Universidad de las Ciencias Informáticas

Facultad 6

Título: "Librería paralela de algoritmos evolutivos aplicados a problemas computacionalmente costosos".

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autor: Rigoberto Leander Salgado Reyes

Tutor(es): Lic. Liesner Acevedo Martínez
Lic. Rafael Arturo Trujillo Rasúa

Ciudad de la Habana, Junio del 2009

Declaración de autoría

Declaro ser autor de la presente tesis y reconozco a la UCI los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Rigoberto Leander Salgado Reyes

_____

Asesor del Departamento de Programación
Lic. Liesner Acevedo Martínez

_____

Asesor del Departamento de Programación
Lic. Rafael Arturo Trujillo Rasúa

Datos de contacto

Tutores:

Lic. Liesner Acevedo Martínez
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: frodo@uci.cu

Lic. Rafael Arturo Trujillo Rasúa
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: trujillo@uci.cu

Agradecimientos

Agradecimientos especiales a mis queridos y estimados tutores, Liesner y el trujo, primero por aceptarme en su grupo y luego por AGUANTARME, mis agradacimientos a Tay por sus opiniones sencillas y sinceras, Blanco mi hermano, mis agradecimientos para ti también, por AGUANTARME y enseñarme a tratar con todo esto. Le agradezco a Pedro por facilitarme su plantilla de tesis en LaTeXy estar presente siempre, a Maikel Rosabal por ayudarme cada vez que necesitaba demostrar algo y AGUANTARME en el apto. Medalla a la resistencia otorgada a Adrian, Cesar, Ana y Daniel por RESISTIRME todos los días en el laboratorio y estar presente siempre. Agradecimiento a todos mis compañeros del apto. y del aula desde 1^{ro} hasta 5^{to} año, Michel agradecimientos para ti también mi hermano, y para todo aquel que de una forma u otra siempre me apoyo y ayudo.

A todos muchas gracias.

Dedicatoria

Gracias Madre...

No existen palabras para describir lo mucho...

te dedico esta Tesis y todas la que puedan venir.

Resumen

En la presente tesis de diploma se desarrolló una librería paralela, portable, eficiente y de propósito general de algoritmos evolutivos. Su campo de aplicación es la resolución de problemas de optimización computacionalmente costosos. La librería se aplicó de forma eficiente al problema de Selección del Portafolio de Proyecto y al problema del Cálculo Inverso Aditivo de los Valores Singulares de una matriz. En ambos casos se obtuvieron muy buenos resultados de rendimiento de la librería paralela en comparación con la equivalente secuencial. En el problema de Selección de Cartera de Proyecto se definió un nuevo modelo computacional no reportado en la literatura, o al menos no conocido por el autor. Para el problema inverso aditivo de valores singulares se propone una hibridación de Algoritmos genéticos (AG) con métodos de Newton, con la cual se logró una solución que converge en un alto porciento de los casos, resultado notable para un problema de optimización donde los métodos basados en gradiente convergen en situaciones muy particulares.

Palabras Claves: *algoritmos evolutivos paralelos, multiobjetivo, selección del portafolio o cartera de proyecto, problema inverso aditivo de valores singulares.*

Abstract

In this diploma thesis a portable and efficient library of parallel evolutionary algorithms has been developed. With this library, a user can solve by evolutionary algorithms any computationally expensive optimization problem. The library was applied to the Project Portfolio Selection problem and to the Inverse Additive Singular Values Problem (IASVP). In both cases the parallel library showed very good performance results in terms of speed-up and efficiency. Also in the Project Portfolio Selection problem we defined a new computational model not reported in the literature, at least not known by the author. To solve the Inverse Additive Singular Values Problem we propose a hybrid evolutionary algorithm with Newton Methods, and in most of the cases this solution converged to a global minimal, despite the Newton methods that converge only in particular conditions.

Keywords: *parallel evolutionary algorithm, multiobjective, project portfolio selection, inverse additive singular values.*

Índice

Índice de figuras

Índice de tablas

Índice de algoritmos

Introducción

Los *algoritmos evolutivos* son métodos de optimización y búsqueda basados en los principios de selección natural y supervivencia de los individuos más aptos, (Darwin, *On the Origin of Species by Means of Natural Selection, 1959*). A finales de los sesenta y principios de los setenta, John Holland[1] desarrolla los algoritmos genéticos, que son el primer tipo de algoritmos evolutivos. Desde entonces los algoritmos evolutivos se han consolidado como técnicas robustas de optimización y búsqueda, aplicándose para resolver un amplio abanico de problemas.

En la actualidad se presentan problemas de optimización que pueden ser de difícil resolución para los métodos tradicionales. Los métodos tradicionales basados en Newton o cuasi-Newton[2] necesitan la información de todas las derivadas de la función para su cálculo y la estimación de ellas no arroja resultados muy precisos. El cálculo de las derivadas es un proceso lento, además estos métodos presenta un convergencia local. Existen alternativas a los métodos basados en Newton como los métodos de búsqueda directa, como el método Simplex[2], estos son asintóticamente más lentos que los métodos de máximo descenso y a medida que el tamaño del problema crece, la ejecución de la búsqueda directa se deteriora.

Los algoritmos evolutivos permiten realizar optimización de ámbito global[3], pueden integrarse con optimizadores locales[4, 5], permitiendo acelerar la convergencia del algoritmo y la exactitud de los resultados. Para agregar estos optimizadores locales, se debe tener conocimiento adicional del problema a resolver.

Los algoritmos evolutivos son muy usados para realizar optimización multiobjetivo[6, 7, 8, 9], cuando un problema de optimización tiene más de una función objetivo o algunas restricciones, se conoce como optimización multiobjetivo o multicriterio, en la misma se analizan varios criterios o funciones objetivos y restricciones sobre el espacio de solución. Tradicionalmente se resolvían estos problemas por medio de la suma ponderada, la programación por metas, etc. Estos métodos[8] de optimización convierten las múltiples funciones objetivos y restricciones en una sola función mediante la asignación de preferencias a las mismas. Esto no es conveniente porque los resultados varían para diferentes preferencias, además de que deben de encontrarse antes de realizar la ejecución del algoritmo evolutivo[3].

En la actualidad los problemas de optimización multiobjetivo se resuelven por métodos evolutivos multicriterio basados en el concepto de *Pareto óptimo* o *Pareto optimalidad*. El concepto de Pareto fue formulado

por Vilfredo Pareto en el siglo XIX, el mismo plantea que una solución es mejor que otra, si la misma presenta una mejor evaluación en todas las funciones objetivos que la solución restante. Estos métodos evolutivos multicriterio son fuertemente elitistas, los individuos más fuertes son los responsable de crear la próxima generación. Existen varios métodos que han demostrado ser los más eficientes[10] para la generalidad de problemas multiobjetivos, eje. NSGA-2[11], SPEA-2[12], PAES-2[13], etc.

Los algoritmos evolutivos presenta una convergencia más lenta que los métodos que utilizan el cálculo de la derivada, además son algoritmos poblacionales que deben evaluar sus individuos en cada iteración o generación, haciendolos lentos en sentido general, por tanto nos planteamos con el desarrollo de esta tesis la implementación de una librería paralela, eficiente, portable y de propósito general de algoritmos evolutivos que permita reducir la complejidad computacional de varios problemas de uso frecuente en nuestra universidad.

Luego de realizar el diseño, implementación y pruebas de rendimiento, nos proponemos hacer uso de la librería para abordar el problema inverso aditivo de valores singulares[14], y selección de la cartera de proyecto[15]. Ambos problemas presentan una complejidad computacional elevada, por ejemplo: en el cálculo inverso aditivo de valores singulares debe resolverse una multiplicación matriz-matriz(orden n^3), la evaluación de cada individuo en la función objetivo es costosa; en el caso del problema selección de la cartera de proyecto, se deben seleccionar varios procesos dentro de un conjunto no muy grande, pero la cantidad de procesos a seleccionar no es fija, sujetos a varias funciones objetivos y restricciones.

Para dar cumplimiento a la presente tesis nos hemos propuestos, realizar un estudio de los Algoritmos evolutivos (AE), dentro de estos, los algoritmos genéticos y Algoritmos de estimación de distribución de la población (EDA), realizar el diseño e implementación de la librería secuencial, debemos realizar además un estudio de los esquemas paralelos para los AE, realizar la implementación y pruebas de eficiencias de la librería paralela, por último, abordaremos los problemas antes mencionados y realizamos las pruebas de aplicación.

El trabajo está dividido en tres capítulos:

El **Capítulo 1**: Revisión bibliográfica. Se realiza una revisión de los algoritmos genéticos, los algoritmos de distribución de la población, técnicas evolutivas para optimización multiobjetivo y los algoritmos evolutivos en entornos paralelos.

El **Capítulo 2**: Programas y metodologías. Se plantean los programas y las metodologías usadas en la elaboración de la tesis.

El **Capítulo 3**: Resultados y discusión. Se plantea como será implementada la librería de AE de forma secuencial y paralela, el tratamiento que se le dará a los problemas multiobjetivos, además se realizará un análisis de su complejidad. Se aborda la aplicación de la librería, tanto secuencia como paralela, en el Problema inverso aditivo de valores singulares y la Selección de la cartera o portafolio de proyecto.

Capítulo 1

Revisión bibliográfica.

La optimización es el proceso que realiza algo se forma eficiente, si algún ingeniero o científico plantea una idea, la optimización esta implícita en la misma. La optimización es la herramienta matemática que nos permite obtener las soluciones más acertadas, la optimización es la actividad que ajusta las entradas a cierto proceso matemático, o experimento para encontrar la máxima o mínima salida o resultado. Los AE son una herramienta que permite a estos ingenieros y científicos plantear sus funciones objetivos, variables de entrada y salida; y un medio para demostrar la exactitud de sus ideas.

1.1. Problemas de optimización.

Los problemas de optimización sin restricciones consisten en minimizar el valor real de una función f de N variables. Por este medio podemos plantear que para encontrar el *mínimo local*, será encontrar un punto x^* tal que:

$$f(x^*) \leq f(x) \; para \; todo \; x \; cercano \; x^*. \tag{1.1}$$

Es estándar expresar el problema de la siguiente forma:

$$min_x f(x) \tag{1.2}$$

Entendemos por 1.1 como un optimizador local, nos referimos a f como la *función objetivo* y a $f(x^*)$ como el mínimo local. Este problema es diferente del problema de encontrar el *mínimo global*, donde el mismo es encontrado de la siguiente forma:

$$f(x^*) \leq f(x) \; para \; todo \; x. \tag{1.3}$$

Algunas veces la maximización tiene más sentido, para maximizar una función, simplemente invertimos el signo de la propia función, por ejemplo, maximizar $1 - x^2$ en el intervalo $-1 \leq x \leq 1$, es el mismo problema que minimizar $x^2 - 1$ sobre el mismo intervalo.

El problema de optimización con restricciones es minimizar una función f sobre un conjunto $U \subset \Re^N$, luego un minimizador local es un $x^* \in U$ tal que:

$$f(x^*) \leq f(x) \; para \; todo \; x \in U \; cercano \; x^*. \tag{1.4}$$

de forma similar se expresa,

$$min_{x \in U} f(x) \tag{1.5}$$

el minimizador global es un punto $x^* \in U$ tal que

$$f(x^*) \leq f(x) \; para \; todo \; x \in U. \tag{1.6}$$

1.2. Problemas de optimización multiobjetivo.

Cuando un problema de optimización tiene más de una función objetivo, entonces se llama *optimización multiobjetivo*. La mayoría de los problemas de optimización en el mundo real tiene múltiples objetivos, los cuales suelen entrar en conflicto; es decir, mejorar un objetivo significa empeorar otro. Los algoritmos tradicionales resuelven problemas de optimización multiobjetivo, transformando el problema en un problema de optimización simple utilizando algún criterio de preferencias. Los algoritmos evolutivos permiten obtener múltiples soluciones óptimas en una sola ejecución del algoritmo, donde podemos escoger aquella solución que nos satisfaga según algún criterio de preferencias.

Otro punto importante en la optimización es la definición de *restricciones*, los algoritmos evolutivos han resultado eficaces para resolver problemas de optimización simple y multiobjetivo con restricciones.

Un problema de optimización multiobjetivo sujeto a restricciones se define como:

$$Minimizar f_i(x), i = 1, \ldots, n \tag{1.7}$$

$$sujeto \; a : g_i(x) \leq 0, j = 1, \ldots, m \tag{1.8}$$

donde $x = (x_1, \ldots, x_p)$ es un vector de parámetros reales $x_k \in \Re$.

1.3. Algoritmos evolutivos.

Los algoritmos evolutivos son métodos estocásticos de búsquedas, que han sido aplicados satisfactoriamente a una gran cantidad de problemas de búsqueda, optimización y aprendizaje, los mismos mantienen

una población de tentativas soluciones que han sido manipuladas competitivamente por medio de la aplicación de operadores de variación. En los siguientes sub-epígrafes haremos énfasis en dos clases de algoritmos evolutivos: los Algoritmos genéticos y los EDAs.

1.3.1. Algoritmos genéticos.

Los Algoritmos genéticos (AG), son una técnica de optimización y búsqueda, basada en los principios de la genética y la selección natural. Los AG presentan poblaciones compuestas por varios individuos, que evolucionan sujetos a una regla de selección especifica, hasta alcanzar la aptitud necesaria. Este método fue desarrollado por John Holland, en la década del 1960 y parte del 1970, y finalmente popularizado por uno de sus alumnos, David Goldberg.

Los elementos básicos de los AG, son la selección de los elementos con más aptitud, la reproducción por cruzamiento de genes y la mutación, que permite cambiar aleatoriamente los genes. Todos estos elementos repetidos iterativamente sobre una población ofrece un acercamiento a la solución del problema. Los AG toman prestado conceptos como genes, cromosomas, selección, reproducción, mutación, evolución, entre otros que permiten construir un modelo matemático más sencillo de implementar computacionalmente.

La aplicación más común de los algoritmos genéticos ha sido la solución de problemas de optimización, en donde han mostrado ser muy eficientes y confiables. Sin embargo, no todos los problemas pudieran ser apropiados para la técnica, y se recomienda en general tomar en cuenta las siguientes características del mismo antes de intentar usarla:

- Su espacio de búsqueda (i.e., sus posibles soluciones) debe estar delimitado dentro de un cierto rango.
- Debe poderse definir una función de aptitud que indique qué tan buena o mala es una cierta respuesta.
- Las soluciones deben codificarse de una forma que resulte relativamente fácil de implementar en el computador.

Esta función de aptitud es la función objetivo del problema de optimización, dicha función debe ser capaz de premiar los buenos resultados y penalizar los malos, permitiendo que las buenas cualidades se propaguen a la siguiente generación.

Se han utilizado una gran variedad de lenguajes para implementar sistemas evolutivos como LISP, PASCAL, C, y muchos otros menos conocidos [16], e incluso dependientes de la máquina. Las tendencias actuales

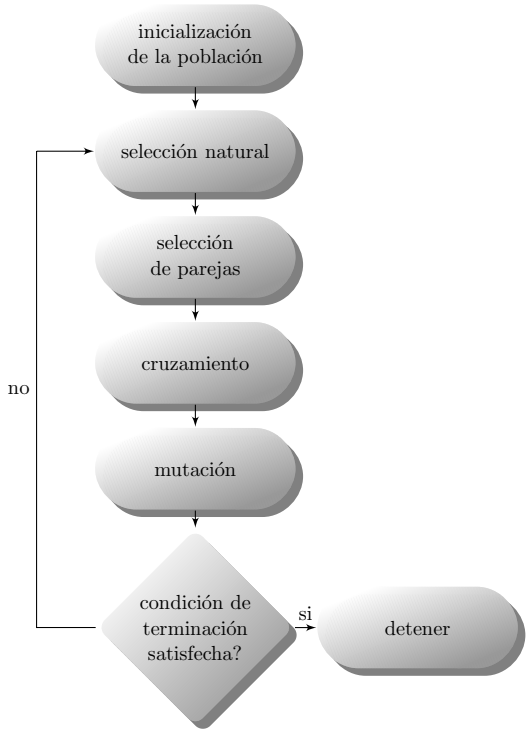
conducen a usar lenguajes orientados a objetos donde todas las características mencionadas están presentes para permitir desarrollar sistemas complejos como los incluidos en GAGS, GAME, GAlib, TOLKIEN, EVOLVER y varios más.

Estructura de los Algoritmos Genéticos.

Los genes son la unidad fundamental de los AG, estos pasan a componer los cromosomas, los cuales a su vez, serán una solución del problema en cuestión, sobre la población de cromosomas aplicamos ciertos operadores, los cuales permiten que las buenas cualidades de los cromosomas se transmitan de generación en generación, además de permitirles que adquieran cualidades provocadas por el medio externo, mediante la mutación. Utilizando una función de aptitud o función objetivo, evaluamos cada individuo de la población, conociendo así, las mejores soluciones, las cuales son los cromosomas más adaptados para continuar en la evolución de la población.

Supongamos que estamos maximizando la función $F(x) = x^2 + 4y - 3z$, los cromosomas para este problema especifico tienen esta forma $[[gen1][gen2][gen3]]$, donde cada gen es del tipo especificado y se asocia con cada variable de la función. Si fuéramos a representar este problema con genes binarios, un ejemplo de cromosoma sería, [[10101][00101][11100]], suponiendo que las variables del problema no alcanzaran valores mayores a 31. En el genoma encontraremos la población de cromosomas además de los operadores que transformaran dicha población, el genoma es el encargado de realizar las iteraciones, donde los cromosomas evolucionarán hasta alcanzar la convergencia, la misma puede definirse como una cantidad de iteraciones máximas a realizar o teniendo en cuenta que la población se haya estabilizado(i.e., cuando todos o la mayoría de los individuos tengan la misma aptitud).

En esencia un algoritmo genético funciona de la siguiente forma:



Algoritmo genético secuencial

1: $t := 0$
2: *inicializar:* $P(0) := \{\overrightarrow{a}_1(0), \ldots, \overrightarrow{a}_\mu(0)\} \in \mathcal{G}^\mu$
3: *evaluar:* $P(0) : \{\Phi(\overrightarrow{a}_1(0)), \ldots, \Phi(\overrightarrow{a}_\mu(0))\}$
4: **mientras** $\iota(P(t)) \neq$ **cierto hacer**
5: *seleccionar:* $P'(t) := s_{\Theta_s}(P(t))$
6: *recombinar:* $P''(t) := \otimes_{\Theta_c}(P'(t))$
7: *mutar:* $P'''(t) := m_{\Theta_m}(P''(t))$
8: *evaluar:* $P'''(t) : \{\Phi(\overrightarrow{a}'''_1(0)), \ldots, \Phi(\overrightarrow{a}'''_\lambda(0))\}$
9: *reemplazar:* $P(t+1) := r_{\Theta_r}(P'''(t) \cup Q)$
10: $t := t + 1$
11: **fin mientras**

Figura 1.1: Secuencia de un algoritmo genético.

Selección natural.

La selección natural es el proceso donde se seleccionan los cromosomas más capaces de la población, los que pasarían al proceso de selección de parejas, el porciento X_{tasa}, que sobrevive es indicado al iniciar la corrida del algoritmo genético, se seleccionan los mejores, ordenando por costo de menor a mayor o por aptitud de mayor a menor, tomando los $N_{sobreviven}$ primeros y descartando el resto, $N_{sobreviven} = N_{total} * X_{tasa}$, estas capacidades son reemplazadas por las descendencias que llegaran luego del proceso del cruzamiento, esta operación se realiza en cada iteración del AG.

Cromosomas	Costos
00101111000110	−12359
11100101100100	−11872
00110010001100	−13477
00101111001000	−12363
11001111111011	−11631
01000101111011	−12097
11101100000001	−12588
01001101110011	−11860

Tabla 1.1: Ejemplo de población inicial de 8 cromosomas.

Cromosomas	Costos
00110010001100	−13477
11101100000001	−12588
00101111001000	−12363
00101111000110	−12359

Tabla 1.2: Ejemplo de población seleccionada.

Selección de parejas.

La selección de parejas es la operación encargada de seleccionar dentro de los más aptos, los padres necesarios para el cruzamiento, esta selección puede realizarse de varias formas, algunas de estas más factibles que otras, en dependencia de como se comporten las descendencias que produzcan estos padres. La selecciones más usadas son:

- **Selección simple**: Es el método de escoger las parejas más simple, se basa en seleccionar los padres de forma consecutiva, por ejemplo, si fueramos a realizar el cruzamiento de dos padres, uniríamos el cromosomas uno con el dos, el tres con el cuatro y así sucesivamente.

- **Selección aleatoria:** Mediante este método se eligen los padres al azar, imitando así, la selección aleatoria que puede ocurrir en el mundo biológico. Para el caso de que se repitan los padres en la pareja escogida, podemos mantenerlos o volver a escoger a otro padre.

- **Selección aleatoria ponderada.**

  - **Selección ponderada por rango:** Para realizar este tipo de selección debemos tener en cuenta la probabilidad de cada cromosoma dentro de la población, la misma se calcula por la fórmula siguiente:

$$P_n = \frac{N_{sobreviven} - n + 1}{\sum_{n=1}^{N_{sobreviven}} n} \tag{1.9}$$

  luego la probabilidad acumulada de cada uno, nos permite encontrar los padres necesarios que pasaran al cruzamiento, cada padre es seleccionado generando un numero aleatorio y verificando el rango donde se encuentra dentro del la probabilidad acumulada.

n	Cromosomas	P_n	$\sum_{i=1}^{n} Pi$
1	00110010001100	0,4	0,4
2	11101100000001	0,3	0,7
3	00101111001000	0,2	0,9
4	00101111000110	0,1	1

Tabla 1.3: Selección de parejas, ponderado por rango.

  Si el numero generado aleatoriamente es $r = 0{,}5325$; $0{,}4 < r <= 0{,}7$, dicho numero se encuentra en el intervalo correspondiente al cromosoma 2, los números menor o iguales a 0.4 seleccionaran a el cromosoma 1 como padre y así sucesivamente.

  - **Selección ponderada por costo:** Esta variante de la selección, se realiza de manera similar a la anterior, solo difiere la forma de determinar la probabilidad, la misma se basa en el costo normalizado de cada cromosoma $C_n = c_n - c_{N_{sobreviven}}$, restándole a cada cromosoma el costo del cromosoma de menor costo de los desechados, se asegura que el costo normalizado sea negativo, luego la probabilidad es la siguiente:

$$P_n = \left| \frac{C_n}{\sum_{n=1}^{N_{sobreviven}} C_n} \right| \tag{1.10}$$

- **Selección por torneo:** En esta selección se escogen dos o tres elementos y de estos se elige el más apto, esta operación se realiza cada vez que se necesite un padre.

Cruzamiento.

El operador de cruzamiento está muy relacionada con la estructura del cromosoma, en esta sección tomaremos los ejemplo del cromosoma binario y real para mostrar dos tipos de cruzamiento.

Para genes binarios se selecciona uno o dos números aleatorios, en dependencia de si el cruzamiento sera en uno o en dos puntos. Si el cruzamiento es en un punto, P_1, se toman los primeros bits del padre uno (desde el bit 1 hasta el bit P_1), y se completa con los restantes del padre dos (del bit P_1 hasta la cantidad total de bits), para formar la descendencia uno y viceversa para la descendencia dos.

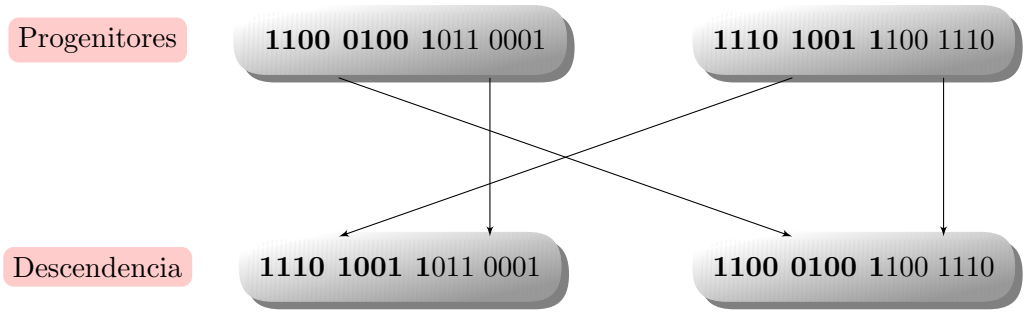


Figura 1.2: Cruzamiento en un punto.

Si el cruzamiento es en dos puntos, P_1 y P_2, para crear la descendencia uno se sustituyen los bits desde P_1 hasta P_2 del padre dos en el padre uno, y de forma contraria para crear la descendencia dos.

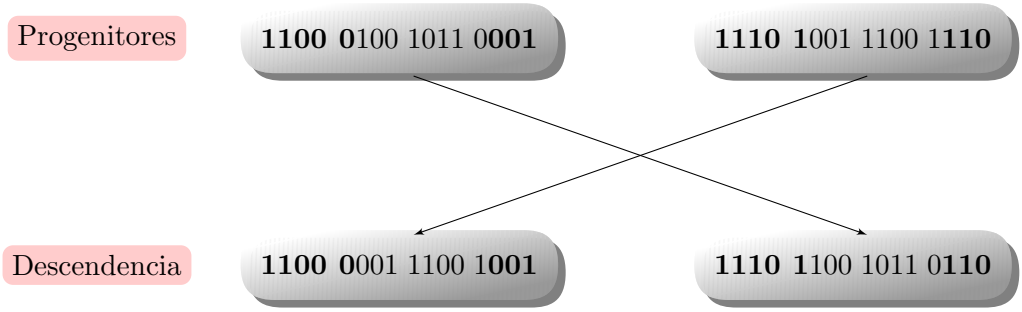


Figura 1.3: Cruzamiento en dos puntos.

Para realizar el cruzamiento usando genes decimales, supongamos que tenemos los progenitores:

$$padre_1 = [p_{m1}p_{m2}...p_{m\alpha}...p_{mN_{var}}] \tag{1.11}$$

$$padre_2 = [p_{d1}p_{d2}...p_{d\alpha}...p_{dN_{var}}] \tag{1.12}$$

donde

$$\alpha = random(1, N_{var}), \tag{1.13}$$

p_{mn} y p_{dn} son las n-ésimas variables del los cromosomas padres respectivamente. Para crear la descendencia debemos encontrar las variables de corte,

$$p_{corte1} = p_{m\alpha} - \beta[p_{m\alpha} - p_{d\alpha}] \tag{1.14}$$

$$p_{corte2} = p_{d\alpha} + \beta[p_{m\alpha} - p_{d\alpha}] \tag{1.15}$$

donde β es un valor en el intervalo $[0, 1]$.
Luego la descendencia quedaría:

$$descendencia_1 = [p_{m1}p_{m2}...p_{corte1}...p_{dN_{var}}] \tag{1.16}$$

$$descendencia_2 = [p_{d1}p_{d2}...p_{corte2}...p_{mN_{var}}] \tag{1.17}$$

si la variable seleccionada es la primera, entonces se cambian las demás, si la seleccionada es la ultima, se cambian las anteriores. Estos son dos ejemplo de cruzamiento de cromosomas compuestos por genes específicos, cuando definamos el gen que vamos a usar, debemos definir como se van a cruzar los cromosomas.

Mutación.

La mutación tanto como el cruzamiento, es un operador especifico para nuestros genes, con el mismo se introduce un cambio en la estructura del cromosoma en cuestión, lo cual posibilita que el algoritmo amplíe su busqueda en la superficie de costo del problema, además de no permitir que se estanque en un extremo local. La mutación en cromosomas binarios ocurre cambiando el bit de una posición especifica por su opuesto, el numero de mutaciones esta dado por,

$$\#mutaciones = \mu \times (N_{pob} - 1) \times N_{bits}, \tag{1.18}$$

donde μ es el por ciento de mutaciones que se le especifico al algoritmo.
luego conocer cuantas mutaciones deben ocurrir, se escogen al azar tantos cromosomas como mutaciones se necesiten, luego se escoge cual sera el bit que mute, puede ocurrir que un cromosomas mute varias veces.

Para cromosomas decimales este proceso ocurre de forma similar, tenemos el por ciento de mutaciones que deben ocurrir,

$$\#mutaciones = \mu \times (N_{pob} - 1) \times N_{var}, \tag{1.19}$$

luego de escoge el cromosomas y del mismo el gen que debe mutar, si elige un número aleatorio que reemplazará dicho gen.

La próxima generación.

La próxima generación se compone de los cromosoma que sobrevivieron a la selección natural, a estos se unen las descendencias generadas mediante el cruzamiento, y los cromosomas que mutaron, la población en cada generación se estará estabilizado, hasta alcanzar la convergencia del algoritmo.

Convergencia.

La convergencia es el proceso que determina cuando el algoritmos genético debe parar, esta vendría siendo la etapa final de cada iteración del algoritmo, las misma puede realizarse de varias formas:

- **Cantidad de iteraciones:** Se determina de forma empírica cuantas veces debe iterar el algoritmo para que converja en el extremo deseado.
- **Convergencia estadística:** Este es un tema bastante difuso dentro de los algoritmos genéticos, los metodos más aceptados se basan en al distribución normal de la población, en su varianza media y estándar, estos se fundamentan con el teorema esquema de Holland, y el teorema de Prince[17].
- **Población estabilizada:** Cuando la mayoría de la población tenga la misma aptitud, se puede decir que el algoritmo esta sobre un extremo, este puede ser una buena solución o que el algoritmo esta estancado en un extremo local, debemos ser cuidadosos con esto ultimo, ya que los buenos resultados suelen estancarse antes de que el próximo cruzamiento o mutación lo lleve a un mejor individuo.

Parámetros.

Los parámetros de entrada más significativos de los algoritmos genéticos son:

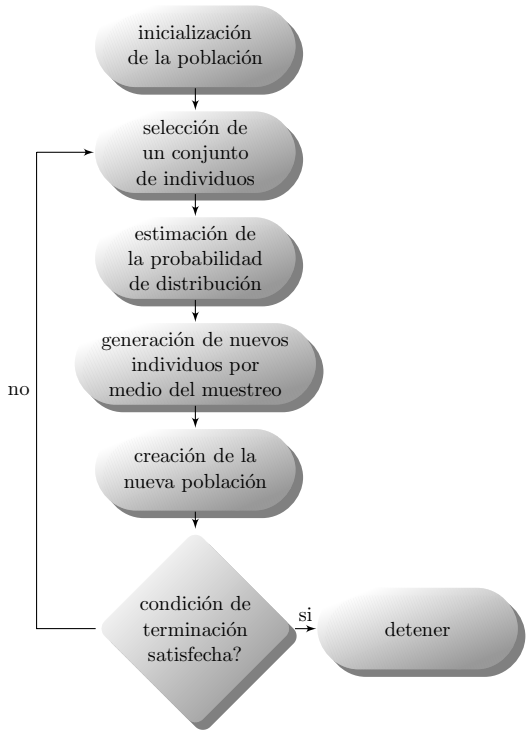
- Tamaño de la población.
- Número de generaciones.
- Probabilidad de cruce.
- Probabilidad de mutación.

En [18] se estima que le número esperado de generaciones hasta la convergencia es una función logarítmica del tamaño de la población. Para las probabilidades de cruce y mutación, los estudios realizados en el tema apuntan a probabilidades de cruce altas y probabilidades de mutación bajas. Los siguientes valores han sido considerados como aceptables para un buen rendimiento de los algoritmos genéticos para funciones de optimización [19]:

- Tamaño de la población: 50 – 100.
- Probabilidad de cruce: 0.60.
- Probabilidad de mutación: 0.001.

1.3.2. Algoritmos de estimación de distribución de la población.

Los algoritmos de estimación de distribución de la población son algoritmos evolutivos que operan sobre un conjunto de poblaciones de posibles soluciones o puntos. La figura 1.4 ilustra una aproximación del funcionamiento de los EDAs



EDA

1: $t := 0$
2: *generar M puntos aleatoriamente*
3: **repetir**
4: *evaluación de los puntos usando la función de aptitud*
5: *seleccionar un conjunto S de $N \leq M$ de acuerdo al método de selección*
6: *estimar el modelo probabilístico para S*
7: *generar M nuevos puntos de la distribución representada*
8: $t := t + 1$
9: **hasta que** *el criterio de parada sea encontrado*

Figura 1.4: Secuencia de un EDA.

Los investigadores son conscientes de que este modelo simple, generalmente realiza poca demanda de recursos y es fácil de entender, pero presenta una capacidad limitada de representar las interacciones, es decir, modelos más complejos pueden requerir sofisticadas estructuras de datos con un costo de aprendizaje mayor.

Otro criterio de decisión que debe ser tomado en consideración a la hora de escoger un EDA, es el conocimiento que se tenga respecto a la estructura del problema, cual modelo probabilista es el mejor a la hora de representar este conocimiento, los siguiente modelos intentan ayudar en esta selección.

Los EDAs están divididos de acuerdo a la complejidad del modelo probabilístico usado para capturar las interdependencias entre variables: *univariate*, *bivariate*, *multivariate*. Los EDAs que están dentro de la definición de *univariate*, como PBIL[20], cGA[21], UMDA[22], asumen que todas las variable son independientes y factorizan la unión de las probabilidades de los puntos seleccionados como un producto de probabilidades para una sola variable. Esta es la representación más simple de los EDAs y ha sido aplicada a problemas con representaciones continuas[23].

La definición del modelo *bivariate* puede representar un orden bajo de dependencia entre las variables. MIMIC[24], BMDA[25], y los EDAs basados en árboles de dependencias[26] y los árboles basados en la estimación de distribución de la población(Tree-EDA)[27], son todos miembros de esta subclase. Los dos últimos son las factorizaciones basadas en árboles, recomendados para problemas con una alta cardinalidad de las variables y donde las interacciones juegan un papel importante. Los árboles y bosques pueden combinarse para representar un alto numero de interacciones, usando modelos de distribución mixta[27].

Los EDAs que pertenecen al modelo *multivariate*, factorizan la unión de la probabilidad de distribución usando estadísticas de orden mayor a dos[28]. Como la dependencia entre las variables es mayor que en las categorías anteriores, la complejidad de la estructura probabilista, así como el tiempo computacional requerido para encontrar los mejores puntos es mayor, por tanto este modelo presenta un complejidad de aprendizaje mayor.

1.3.3. Técnicas basadas en conocimiento.

Los algoritmos evolutivos en su forma más simple son algoritmos ciegos de búsquedas; utilizan únicamente la codificación y el valor de la función objetivo para determinar qué solución se mantienen en la siguiente generación. Esta indiferencia hacia la información específica del problema dota a los AE de una gran robustez y versatilidad, pero, el no usar todo el conocimiento disponible de un problema particular pone a los AE en desventaja competitiva con respecto a los métodos que sí usan tal información.

En [29, 18] se discuten varias formas de combinar información específica del problema con AE, una vía es usar la codificación del problema, que permite representar el conocimiento del experto de igual forma a como este lo observa, estas técnicas son conocidas como técnicas de hibridación, donde los AE pueden ser combinados con las técnicas que exploten el conocimiento que se tiene del problema en cuestión, creando

un algoritmo que presenta la perspectiva global de los AE y la convergencia de las técnicas específicas del problema.

La forma de hacer uso de estas técnicas de hibridación es adicionar un operador más al algoritmo, encargado de realizar la optimización local, esta optimización local puede ser una técnica de búsqueda directa, ramificación y poda, etc.

1.3.4. Técnicas evolutivas para la optimización multiobjetivo.

El primer trabajo donde se sugiere utilizar un algoritmo evolutivo para resolver un problema multiobjetivo lo realiza Rosenberg a finales de los 60 [30]; desde entonces esta nueva área de investigación (actualmente optimización evolutiva) ha tenido cada vez más relevancia y efectividad en la resolución de problemas de optimización multiobjetivo.

Existen diferentes técnicas evolutivas para resolver problemas multiobjetivos. Una posible clasificación es la siguiente[8]:

- Articulación de preferencias *a priori*.
- Articulación de preferencias *a posteriori*.

Articulación de preferencias a priori.

En estas técnicas, se deben tomar decisiones de preferencia antes de ejecutar el algoritmo; normalmente estas decisiones se realizan en forma de pesos, prioridades o preferencias asignadas a cada uno de los objetivos, convirtiendo el problema multiobjetivo en un problema de un solo objetivo. Entre las técnicas que se encuentran englobadas en este conjunto tenemos:

- Método de la Suma Ponderada.
- Método de la Multiplicación.
- Programación por Metas.
- Consecución de Metas.
- Aproximación Min-max con pesos.

- Orden Lexicográfico.

Articulación de preferencias a posteriori.

En estas técnicas el uso de los algoritmos evolutivos alcanza su plena realización, puesto que la optimización se realiza sin tener en cuenta preferencias, evolucionando una población donde todos los objetivos se tienen en cuenta, obteniendo al final un conjunto de soluciones no dominadas. Por otra parte estas técnicas son las que aprovechan la ventaja del paralelismo de los algoritmos evolutivos para resolver problemas multiobjetivos. Podemos distinguir las siguientes:

- Métodos no basados en el concepto de Pareto
    - Algoritmo Genéticos sin Generaciones
    - Método de las restricciones ϵ
    - Algoritmo Genético de Vector Evaluado (VEGA)
    - Algoritmo Genético Basado en Pesos (WBGA)
    - Uso de la Teoría de Juegos
    - Uso del Género para identificar objetivos
    - Estrategia de Evolución Presa-Depredador
    - Algoritmo Genético con Participación Distribuida
    - Método del Aprendizaje por Refuerzo Distribuido
    - Sistema Neuro-Evolutivo Multiobjetivo(MONESSY)
- Métodos no elitistas basados en el concepto Pareto
    - Algoritmo de Ordenación Pareto
    - Algoritmo Genético con Múltiples Objetivos(MOGA)
    - Algoritmo Genético Pareto con Nichos(NPGA)
- Métodos elitistas basados en el concepto Pareto
    - Algoritmo Genético de Ordenación No Dominada Elitista(NSGA-II)
    - Algoritmo Genético Pareto basado en la Distancia(DPGA)

- Algoritmo Evolutivo con Fuerzas Pareto(SPEA-II)
- Algoritmo Genético Termodinámico(TDGA)
- Estrategia de Evolución con Archivo Pareto(PAES-II)
- Algoritmo Genético Desordenado Multiobjetivo(MOMGA)
- Micro-Algoritmo Genético Multiobjetivo(MμGA)
- Algoritmo Evolutivo Multiobjetivo Elitista con Participación Co-evolutiva(ERMOCS)

En la actualidad los mejores algoritmos evolutivos para resolver problemas de optimización multiobjetivo son los considerados algoritmos elitistas y basados en el concepto de Pareto; el concepto de *Pareto óptimo* o *Pareto optimalidad* se formula por Vilfredo Pareto en el siglo XIX y constituye por sí mismo el origen de la investigación en optimización multiobjetivo.

Se dice que una solución $x \in F$ *domina* a otra solución $x' \in F$ si:

$$f_i(x) \leq f_i(x') \text{ para todo } i = 1, \ldots, n \tag{1.20}$$

$$y\ f_i(x) < f_i(x') \text{ para algún } i = 1, \ldots, n \tag{1.21}$$

Una solución $x \in F$ es *Pareto óptima* si, para todo $x' \in F$:

$$o \text{ bien } f_i(x) = f_i(x') \text{ para todo } i = 1, \ldots, n \tag{1.22}$$

$$o \text{ bien } y\ f_i(x) < f_i(x') \text{ para algún } i = 1, \ldots, n \tag{1.23}$$

Es decir, una solución $x \in F$ es *Pareto óptima* en F, si no existe ninguna otra solución $x' \in F$ que mejore a x en algún criterio sin empeorarlo simultáneamente en otro criterio. dicho de esta forma, una solución $x \in F$ es Pareto óptima si no existe ninguna otra solución $x' \in F$ que la domine.

1.4. Introducción a la programación paralela.

La computación paralela se puede definir como una técnica de programación, en la que el problema a resolver se divide en tareas cuyas resoluciones se ejecutarán a un mismo tiempo, consiguiendo con ello disminuir el tiempo total necesario para resolver tal problema. En los últimos años ha ocurrido un aumento de los problemas que requieren un alto grado de cómputo para su resolución, la existencia de estos problemas complejos computacionalmente, exigen la fabricación de computadores potentes para solucionarlos en un tiempo adecuado[31].

1.4.1. Arquitecturas paralelas.

Existen dos clasificaciones típicas [32], una atendiendo al mecanismo de control en los diferentes procesadores, y otra atendiendo a cómo se ve el espacio de direcciones en cada procesador.

Clasificación según el mecanismo de control.

La clasificación de los computadores paralelos más conocida y citada es la clasificación de Flynn, las misma se basa en la ausencia o presencia de un control global que permite regular el flujo de instrucciones y datos del procesador. En esta clasificación aparecen cuatro clases:

- **SISD** (Simple Instruction Simple Data): Un flujo único de instrucciones se ejecuta sobre un único conjunto de datos. En esta clase se engloban de forma evidente los computadores secuenciales.
- **SIMD** (Simple Instruction Multiple Data): Un flujo único de instrucciones se ejecuta sobre diferentes conjuntos de datos.
- **MISD** (Multiple Instructions Simple Data): Diferentes flujos de instrucciones se ejecutan sobre el mismo conjunto de datos.
- **MIMD** (Multiple Instructions Simple Data): Diferentes flujos de instrucciones se ejecutan sobre diferentes conjuntos de datos.

Clasificación según la organización del espacio de direcciones.

Según la organización del espacio de dirección, los computadores paralelos se clasifican en:

- **Multiprocesadores de memoria compartida**: Todos los elementos de proceso tienen acceso a una memoria común. Las restricciones de acceso a la memoria suelen ser únicamente de tipo físico. Una red de interconexión une los distintos elementos de proceso con los módulos de memoria, compartida por todos ellos. La comunicación entre los distintos elementos de proceso se hace utilizando la memoria. Para ello basta con que un determinado procesador escriba una variable en memoria y esta es leída por otro procesador.
- **Multiprocesadores de memoria distribuida**: En los multiprocesadores de memoria distribuida cada elemento de proceso tiene su propia memoria local donde almacena sus propios datos, no exis-

tiendo una memoria global común. Una red de interconexión une los distintos elementos de proceso entre sí. La comunicación entre los procesadores se realiza mediante paso de mensajes.

1.4.2. Evaluación de los algoritmos paralelos.

La evaluación de los algoritmos puede hacerse teórica o experimentalmente. Cualquiera sea el caso, es necesario contar con varias métricas de evaluación de prestaciones, que consideren el tamaño de la entrada al algoritmo n y el número de procesadores P. Las métricas más comúnmente utilizadas son [32]:

Tiempo de ejecución, denotado por la función
$T(n, P) = T_A(n, P) + T_C(n, P)$, donde

$T_A(n, P)$,
es el tiempo empleado por el algoritmo para realizar operaciones aritméticas y
$T_C(n, P)$,
es el tiempo empleado por el algoritmo para realizar comunicaciones entre procesadores.

Ganancia en velocidad de ejecución(SpeedUp), que mide la ganancia de velocidad del algoritmo paralelo, respecto al mejor algoritmo secuencial correspondiente, y está dado por:

$$S(n, P) = \frac{T(n, 1)}{T(n, P)}.$$

Eficiencia, que mide el porcentaje del tiempo que los procesadores se mantienen útilmente empleados. La eficiencia se define así:

$$E(n, P) = \frac{S(n, P)}{P}.$$

Escalabilidad, es la capacidad de un determinado algoritmo de mantener sus prestaciones cuando aumenta el número de procesadores. Nos indica la capacidad del algoritmo de utilizar de forma efectiva un incremento en los recursos computacionales.

1.4.3. Entorno de programación paralela.

Para hacer un uso efectivo de la computación paralela es necesario considerar los siguientes aspectos [32]:

Contar con computadores paralelos que se puedan escalar a un número grande de procesadores y que sean capaces de soportar comunicación rápida, así como compartir datos entre procesadores.

Diseñar algoritmos paralelos eficientes. Las metodologías de diseño de este tipo de algoritmos pueden resultar radicalmente diferentes de las utilizadas en el diseño de los correspondientes algoritmos secuenciales, ya que es necesario cuidar los aspectos de balance de carga, distribución de datos, división efectiva de tareas, minimización de tiempo de comunicación y de acceso a memoria.

Para poder evaluar las prestaciones del sistema que resulta de implementar algoritmos paralelos sobre computadoras paralelas, es necesario poder medir qué tan rápido puede resolverse un problema utilizando el sistema paralelo, qué tan eficientemente se utilizan los procesadores de la computadora paralela, etc.

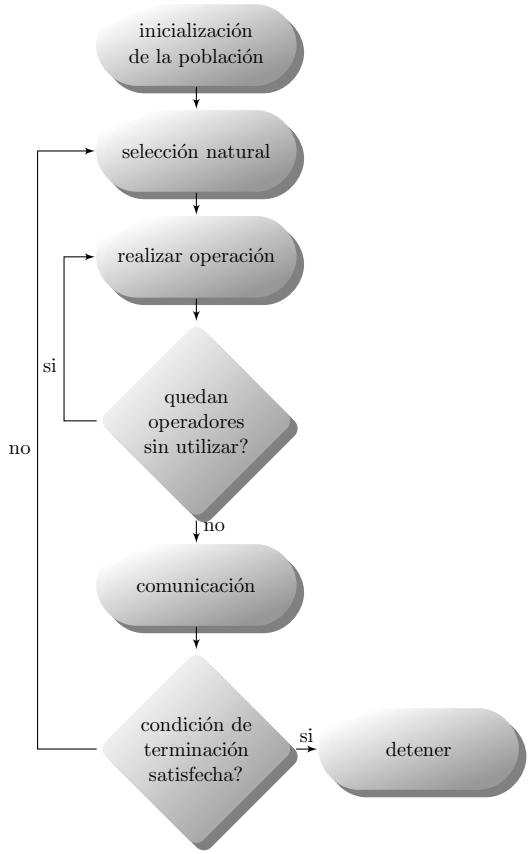
Es necesario, para la implementación de los algoritmos paralelos, contar con lenguajes de programación adecuados para ello.

1.5. Algoritmos evolutivos en entornos paralelos.

Un algoritmo evolutivo paralelo permite utilizar poblaciones de mayor tamaño, resolver problemas de mayor dimensión y complejidad, y reducir el tiempo real de espera por una solución. El funcionamiento de un algoritmo evolutivo paralelo es el siguiente(figura 1.5).

Como puede observarse, un algoritmo evolutivo paralelo extiende la versión secuencial incluyendo una fase de comunicación con un vecindario de otros algoritmos evolutivos[33, 34, 3]. La forma de realizar esta comunicación, el tipo de operaciones del resto de algoritmos en paralelo y otros detalles de funcionamiento determinan su comportamiento global. Sin embargo, podemos observar que este modelo permite también una fácil composición con algoritmos no evolutivos, lo que constituye otra rama interesante de trabajo.

Existen dos modelos principales de AE en paralelos, los **modelos de grano grueso**[35, 36] y los **modelos de grano fino**[37, 38], según sea la relación entre el tiempo de la fase de computación y la fase de comunicación sea alta o baja. En un algoritmo de grano grueso el tamaño de la población es apreciable



Algoritmo evolutivo paralelo

1: $t := 0$
2: *inicializar:* $P(0) := \{\overrightarrow{a}_1(0), \ldots, \overrightarrow{a}_\mu(0)\} \in \mathcal{G}^\mu$
3: *evaluar:* $P(0) : \{\Phi(\overrightarrow{a}_1(0)), \ldots, \Phi(\overrightarrow{a}_\mu(0))\}$
4: **mientras** $\iota(P(t)) \neq$ **cierto hacer**
5: $\quad i := 0$
6: $\quad$ **mientras** $\ell(\eth) \neq$ **cierto hacer**
7: $\quad\quad$ *aplicar operador:* $P'(t) := \eth(i, P(t))$
8: $\quad\quad i := i + 1$
9: $\quad$ **fin mientras**
10: $\quad P'''(t) : \{\Phi(\overrightarrow{a}'''_1(0)), \ldots, \Phi(\overrightarrow{a}'''_\lambda(0))\}$
11: $\quad P''''(t) := comunicar[P'''(t) \cup vecindad :: P'''(t)]$
12: $\quad P(t+1) := selecc\_entorno[P'''(t) \cup P''''(t)]$
13: $\quad t := t + 1$
14: **fin mientras**

Figura 1.5: Secuencia de un algoritmo evolutivo paralelo.

$|P^i(t^i)| = \frac{\mu}{d}$, donde d es el número de subalgoritmos, mientras que en uno de grano fino cada subalgoritmo contiene una sola estructura $|P^i(t^i)| = 1$.

Un algoritmo evolutivo tradicional de granularidad combinada es un modelo AE estructurado en dos o más niveles. Esto puede ser ventajoso debido a que es posible combinar algunas de las ventajas relativas de cada modelo sin, al menos, acentuar sus respectivos inconvenientes (Figura 1.6).

Para el desarrollo de AE de forma paralela existen librerías implementadas principalmente en C, C++ y Fortran, las mismas permite la integración algoritmos de búsqueda local, además de poder utilizarlas en cualquiera de los dos modelos principales, grano fino y grano grueso; es el caso de GAUL, PGAL, PGAPack, etc.

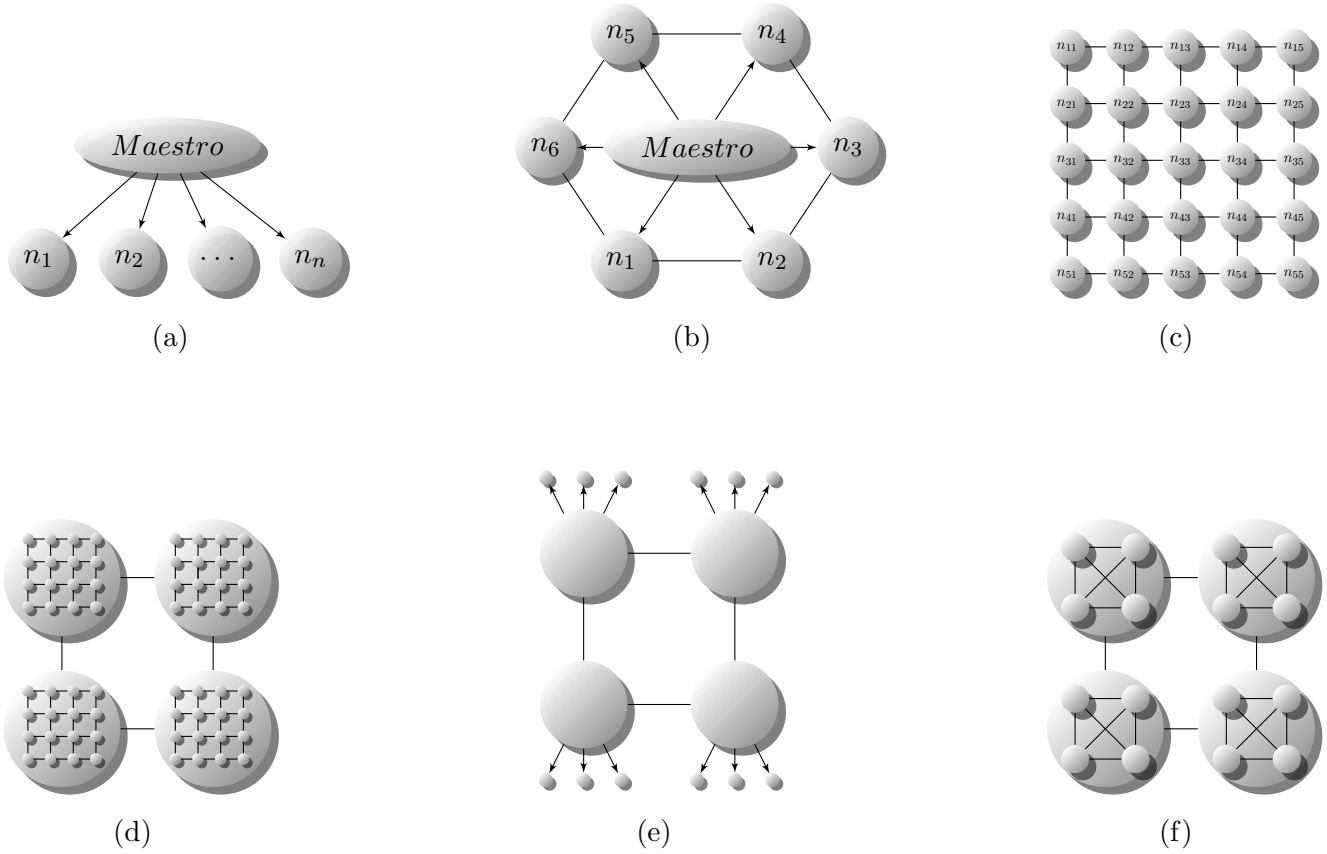


Figura 1.6: Esquemas de un AE (a) con paralelización global, (b) paralelo de grano grueso, y (c) paralelo de grano fino. Existen numerosos modelos paralelos híbridos que combinan a alto y bajo nivel (d) grano grueso y grano fino, (e) grano grueso y paralelización global y (f) grano grueso tanto a alto como a bajo nivel (nomenclatura tradicional).

Conclusiones del capítulo.

En el presente capítulo se realizó un revisión a la definición del problema de optimización monobjetivo y multiobjetivo, con restricciones y sin las mismas, además se realizó un introducción a los algoritmos evolutivos, específicamente, algoritmos genéticos y algoritmos de estimación de distribución de la población. Se realizó una introducción a la programación paralela y a los algoritmos evolutivos paralelos.

Del estudio realizado a los algoritmos evolutivos, observamos como pueden aplicarse a un gran número de problemas, problemas que pueden tener uno o varios criterios de decisión, es válido aclarar que los algoritmos evolutivos dan un acercamiento al extremo global o al frente de Pareto, tratándose de problemas que

necesiten de optimización multiobjetivo, este acercamiento puede mejorarse introduciendo a la modelación del problema, el conocimiento que se tiene del mismo, teniendo en cuenta que la introducción de este concocimiento a la modelación podría impedir que el algoritmo evolutivo realice una búsqueda global y se estanque en un extremo local.

Podemos concluir diciendo que para todo tipo de problema de optimización el cual quiera abordarse con los AE, los algoritmos evolutivos deben adaptarse al problema, es decir, observar que operadores son los necesarios, cual es la forma correcta de expresar las necesidades del problema, cual será su criterio de terminación, sumemos a esto que la mayoría de los problemas de la ingeniería y la ciencia necesitan de grandes poblaciones que abarquen grandes áreas del espacio de soluciones, presentan varios objetivos, los cuales pueden necesitar de bastante tiempo computacional para calcularse, luego es una necesidad incluir la paralelización en la modelación del problema, la cual puede orientarse a dos aspectos específicos, la exactitud del resultado y la reducción del tiempo computacional.

Capítulo 2

Programas y metodologías.

Para resolver el problema científico planteado en la tesis, se hace uso de algunas aplicaciones y herramientas, sobre las cuales descansa el peso del desarrollo de la misma, estas aplicaciones y herramientas son: el IDE Code::Blocks, la colección de compiladores GCC, dentro de esta colección se hizo uso del lenguaje de programación C++; la escritura del documento se realiza en el entorno integrado de LaTeXpara KDE apoyado por el paquete de ofimática Openoffice3. La investigación se basó principalmente en los métodos empíricos, observación, experimento y medición; nos apoyamos en los métodos estadísticos, específicamente en la estadística descriptiva, para clasificar los indicadores cuantitativos obtenidos en la medición, revelándose a través de estos tanto propiedades como problemas imposibles de detectar a simple vista.

2.1. IDE de desarrollo Code::Blocks.

IDE de desarrollo para el lenguaje C y C++, ligero, multiplataforma, reconoce varios compiladores para dichos lenguajes de programación; el código de la aplicación es escrito completamente por medio de este IDE.

2.2. Colección de compiladores GCC.

Colección de compiladores del sistema operativo GNU, dentro de dicha colección se encuentran compiladores para ADA, JAVA, C, C++, FORTRAN, entre otros; el compilador para C y C++ es uno de los que más se acerca al estándar de C y C++, además ser distribuido bajo la licencia GPL versión dos y tres, está portado para otros sistemas operativos.

2.3. Lenguaje de programación C.

Lenguaje de programación creado para realizar modificaciones al sistema operativo Unix, rápido y eficiente, junto al FORTRAN son los lenguajes que con más frecuencia son usados para la creación de aplicaciones de optimización, su versión orientada a objeto C++, permite agregar un nivel más de abstracción a nuestra solución.

2.4. BLAS

La librería BLAS (Basic Linear Algebra Subprograms)[39] se compone de funciones de alta calidad que se basan en la construcción de bloques para efectuar operaciones básicas con vectores y matrices. BLAS está construido en tres niveles: el nivel 1 que efectúa operaciones vector-vector, el nivel 2 que efectúa operaciones matriz-vector y el nivel 3 que efectúa operaciones matriz-matriz. Esta herramienta es eficiente, portable y de amplia disponibilidad, por lo que se utiliza comúnmente en el desarrollo de software del álgebra lineal de altas prestaciones.

2.5. LAPACK

La librería LAPACK (Linear Algebra PACKage)[40] proporciona rutinas para resolver sistemas de ecuaciones, problemas de mínimos cuadrados, problemas de valores propios, vectores propios y valores singulares. También proporcionan rutinas para factorización de matrices, tales como LU, Cholesky, QR, SVD, Schur; tanto para matrices con estructuras específicas o matrices generales.

2.6. Kile.

Entorno integrado para el desarrollo de artículos en LaTeX, ligero y fácil de usar para crear documentos científicos.

2.7. OpenOffice.

Aplicación de ofimática como reemplazo al Microsoft Office, permite usar las bondades del Openoffice.org-calc clon del Microsoft Excel. Herramienta con facilidades para la creación de gráficos y estadísticas.

2.8. Herramientas hardware.

Como herramienta hardware se usó la máquina paralela Atropos. Atropos es una red de PCs que funcionan bajo un ambiente GNU/Linux Debian Lenny 5.0 y consta de 8 nodos biprocesadores Intel(R) Core(TM)2 Duo CPU E4500 2.20GHz, interconectados mediante una red Fast-Ethernet con una topología Beowulf o de anillo. Cada nodo consta de 1 Gigabyte de memoria RAM. Todos los nodos están disponibles para cálculo científico.

2.9. Procedimientos.

El procedimiento utilizado para el cumplimiento de los objetivos de la tesis, es la observación, mediante la cual se realizan modificaciones precisas en la modelación de los casos de estudio, permitiendo obtener buenos resultados.

Luego de realizar una configuración inicial de acuerdo a cantidad de cromosomas, operadores, número de iteraciones, porcentajes de cruzamiento y mutación, se realizaron modificaciones en esta configuración de acuerdo a las necesidades del problema.

Capítulo 3

Resultados y discusión.

En este capítulo se realizará una revisión de los resultados alcanzados con el desarrollo de la tesis. Comenzaremos presentado la modelación e implementación de la librería, desde las estructuras de datos usadas que permiten optimizar el funcionamiento de la librería, hasta la forma de implementar las características esenciales de los algoritmos evolutivos. Seguidamente plantearemos la modelación, implementación y pruebas de los casos de estudios: Optimización de Cartera de Proyecto y Cálculo del problema inverso aditivo de los valores singulares.

3.1. Estructura de los algoritmos evolutivos en la librería.

Para modelar un problema de optimización mediante los AE y para que sea ejecutado por la librería, debemos definir primeramente como estará conformado nuestro gen, eslabón fundamental en nuestra modelación, la librería provee una interfaz, `PEAGene` que nos especifica como deben ser definidos los mismos. Luego debemos definir nuestros operadores, los cuales serán encargados de evolucionar la población de cromosomas, además debemos definir nuestras funciones objetivos y restricciones, en caso de que nuestro problema necesite de las mismas.

Tanto para los operadores como para las funciones objetivos y restricciones, la librería presenta las interfaces `PEAOperator` y `PEAFunction`, encargadas de estandarizar la modelación de dichos artefactos. Los elementos antes mencionados son los fundamentales, podemos además definir como será la convergencia de nuestro problema, como evolucionará la población de cromosomas, pero en estos casos la librería presenta una propuesta genérica para estas instancias(`PEAConvergence`,`PEAEvolution`).

Como recordaremos de la revisión bibliográfica, los AE paralelos presentan un elemento de comunicación, encargado de enviar y recibir los individuos más adaptados de la población a la vecindad, la librería presenta

una definición genérica de este elemento, el cual será analizado posteriormente, permitiendo además utilizar una definición propia.

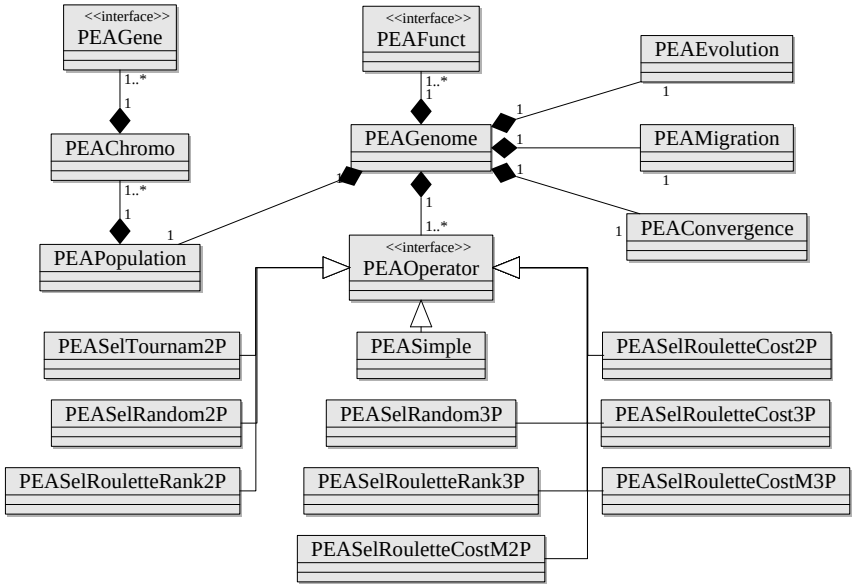


Figura 3.1: Diagrama de clases de la Librería.

Los demás elementos de la librería son el genoma(`PEAGenome`) y la población(`PEAPopulation`). El genoma es el encargado de crear y controlar los elementos necesarios en nuestro problema(figura 3.1). La población es la encargada de contener los cromosomas(`PEAChromo`) que participarán en la evolución. Para almacenar los cromosomas en la librería se usa un árbol rojo-negro(figura 3.2), el cual nos facilita el acceso a los cromosomas. La búsqueda de información sobre este tipo de dato abstracto tiene como cota superior complejidad de $\Theta(logn)$, la inserción de elementos dentro del mismo, también presenta dicha complejidad computacional[41]. Utilizamos este tipo de datos abstracto porque en conjunto las operaciones aleatorias sobre el mismo tienen complejidad superior de $\Theta(logn)$.

Debemos tener presente que este tipo de dato abstracto no almacena elementos repetidos, para solucionar esto se incorpora un lista, que contendrá todos los elementos que coincidan en una posición. Para poder acceder a una posición directamente en el árbol, como si estuviéramos accediendo a una lista, debemos incorporar dos atributos más, que lleven la cantidad de elementos en total que tiene un nodo determinado tanto por la derecha como por la izquierda. Los demás tipos de datos abstractos los provee la librería estándar de plantillas del C++, en este caso del GCC, estos tipos de datos abstractos son los encargados

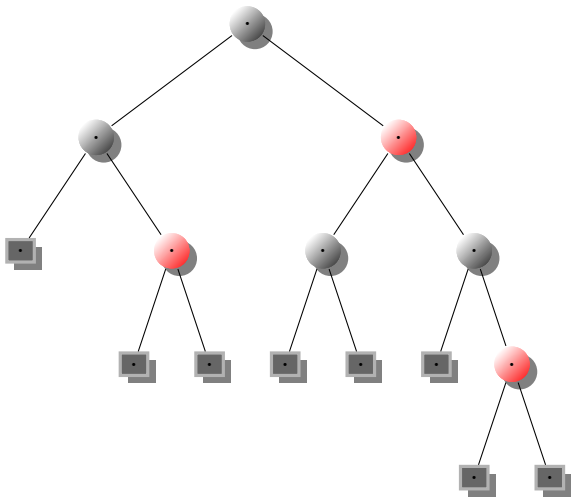


Figura 3.2: Árbol rojo-negro.

de almacenar la restante información de la librería, dígase, operadores, funciones, restricciones, etc.

3.1.1. Selección natural dentro de la librería.

Ya se mencionó que la librería utiliza como tipo de dato abstracto un árbol rojo-negro para almacenar los cromosomas. Un árbol rojo-negro tiene, por definición, los elementos ordenados en entreorden, realizar la selección natural es simplemente eliminar una parte del subárbol derecho, ya que los elementos con más mala aptitud estarán a la derecha; dependiendo del X_{tasa} que se defina puede ser el subárbol derecho completo, permitiendo así que sobrevivan los cromosomas más aptos. Esta operación es costosa ya que se debe podar a lo sumo la mitad del árbol.

La forma de realizar esta operación en la librería presenta complejidad de $\Theta(N_{Total} * (1 - X_{tasa}))$, siendo $N_{Total} * (1 - X_{tasa})$ la cantidad de elementos a eliminar. En caso de realizar esta actividad con otros tipos de datos abstractos, la complejidad en el peor de los casos se mantendría $\Theta(N_{Total} * (1 - X_{tasa}))$, ya que necesariamente se deben eliminar esta cantidad de elementos.

3.1.2. Operadores.

Según el diseño de la librería, no se especifica una cantidad fija de operadores, podemos tener cuantos operadores necesitemos en nuestro problema. Los operadores deberán heredar de la interfaz operador y serán específicos para nuestros genes. Los operadores son almacenados en una lista doblemente enlazada

implementada en la librería estándar de plantillas, los mismos son contenidos en este tipo de dato abstracto porque sobre ellos no se realizarán selecciones aleatorias.

3.1.3. Selección de parejas dentro de la librería.

En esta operación se puede observar la necesidad de un tipo de dato abstracto que optimice el acceso a los cromosomas, si utilizaramos otro tipo de estructura, por ejemplo, una lista doblemente enlazada, la complejidad aumentaría hasta $\Theta(\frac{N}{2})$ siendo N la cantidad de elementos en la estructura. La librería propone la implementación de algunos tipos de selecciones, los cuales son:

- **Selección aleatoria:** Dado que los cromosomas en la librería esta organizados en un árbol rojo-negro, el cual modificamos para seleccionar elementos en entreorden de manera eficiente, la complejidad de obtener un elemento del mismo es $\Theta(logn)$.
- **Selección ponderada por rango:** Para realizar la selección de parejas utilizando el método de la ruleta por pesos, tenemos la siguiente fórmula que dado el numero consecutivo asignado a cada cromosoma, encontramos su probabilidad entre los n cromosomas de la población.

$$P_n = \frac{N_{sobreviven} - n + 1}{\sum_{n=1}^{N_{sobreviven}} n} \tag{3.1}$$

Donde $N_{sobreviven}$, son los cromosomas que sobrevivieron a la selección natural, n el numero del cromosoma en cuestión y $\sum_{n=1}^{N_{sobreviven}} n$, la sumatoria de todos los identificadores asignados a los cromosomas consecutivamente.

La suma incremental de las probabilidades nos permite hallar la probabilidad acumulada, sobre la cual se trabaja para encontrar los padres. Cada padre se encuentra generando un numero aleatorio en el intervalo $[0, 1]$, el rango donde se encuentre el número indicará, el padre seleccionado.

n	Cromosomas	P_n	$\sum_{i=1}^{n} Pi$
1	00110010001100	0,4	0,4
2	11101100000001	0,3	0,7

3	00101111001000	0,2	0,9
4	00101111000110	0,1	1

Tabla 3.1: Selección de parejas, ponderado por rango.

Si el numero generado aleatoriamente es $r = 0{,}5325$; $0{,}4 < r <= 0{,}7$, dicho numero se encuentra en el intervalo correspondiente al cromosoma 2, los números menor o iguales a 0.4 seleccionaran a el cromosoma 1 como padre y así sucesivamente.

Usando a (3.1), podemos encontrar la probabilidad acumulada para cada cromosoma, tomemos $N_{sobreviven}$ como K, y $\sum_{n=1}^{N_{sobreviven}} n$ como S:

$$PA_1 = \frac{K-1+1}{S} = \frac{K}{S}$$

probabilidad acumulada para el cromosoma 1.

$$PA_2 = \frac{K-2+1}{S} + \frac{K}{S} = \frac{2K-1}{S}$$

probabilidad acumulada para el cromosoma 2.

$$PA_3 = \frac{K-3+1}{S} + \frac{2K-1}{S} = \frac{3K-3}{S}$$

probabilidad acumulada para el cromosoma 3.

$$PA_4 = \frac{K-4+1}{S} + \frac{3K-3}{S} = \frac{4K-6}{S}$$

probabilidad acumulada para el cromosoma 4.

$$PA_5 = \frac{K-5+1}{S} + \frac{4K-6}{S} = \frac{5K-10}{S}$$

probabilidad acumulada para el cromosoma 5.

Si nos fijamos con detenimiento observamos como el numero X que multiplica a K va a coincidir con el numero del cromosoma en cuestión, el valor de la S no se altera, solo queda la incógnita del valor que se le sustrae a la $X * K$ en el numerador, si observamos la sucesión $1, 3, 6, 10, ...$, cada elemento de dicha sucesión equivale a la sumatoria de los números menores que X en cada paso de la demostración, conociendo esto podemos encontrar cualquier elemento de dicha sucesión por la fórmula $\frac{(X-1)X}{2}$, demostrando como evolucionan los elementos dentro de la sucesión inicial, la probabilidad acumulada para cualquier cromosoma esta dada por:

$$PA_x = \frac{XK - [\frac{(X-1)X}{2}]}{S} \tag{3.2}$$

si observamos la probabilidad acumulada desde el primer cromosoma hasta el ultimo de cualquier muestra, nos percatamos que presenta una arreglo ordenado,

0.4	0.7	0.9	1

Tabla 3.2: Probabilidad acumulada de la muestra.

para encontrar el rango donde se encuentra el número aleatorio generado con anterioridad R, y conociendo que las probabilidades acumuladas están ordenas, podemos simular una busqueda binaria y preguntar en cada intercepción en cual intervalo se encuentra R.

Otra vía de solución es la siguiente, utilizando a (3.2), si sustituimos la probabilidad acumulada por el número generado aleatoriamente, trataremos de encontrar cual es el padre seleccionado,

$$R = \frac{XK - [\frac{(X-1)X}{2}]}{S}$$

despejando adecuadamente, quedaría $X^2 - (2K+1)X + 2RS = 0$, el discriminante de dicha ecuación quedaría, $D = (2k+1)^2 - 8RS$, si sustituimos S por $\frac{K(K+1)}{2} = \frac{K^2+K}{2}$ obtenemos:

$$D = 4K^2 + 4K + 1 - 4R(K^2 + K)$$
$$D = 4K^2 + 4K + 1 - 4RK^2 - 4RK$$

si realizamos la sustitución $R = 1$ obtenemos $D = 1$, demostrando que para el peor de los casos el discriminante $D > 0$, luego podemos decir que:

$$X_{1,2} = \frac{2K+1 \pm \sqrt{4K(1-R)(K+1)+1}}{2}$$

teniendo en cuanta que,

$$2K + 1 > 2K$$
$$\frac{2K}{2} = K$$

es decir cualquier valor que le sumemos a $2K$ hará que que le resultado se salga dominio del problema, solo nos interesa la fórmula:

$$X_2 = \frac{2K+1-\sqrt{4K(1-R)(K+1)+1}}{2}$$

luego de calcular X_2, comprobamos usando la parte entera de X_2 en (3.2), si el valor coincide con R, $PA_{X_2} = R$, entonces la parte entera de X_2 es el identificador del cromosoma que debemos seleccionar, en caso contrario se le adicionara 1 a la parte entera de X_2.

Podemos concluir diciendo que la complejidad máxima utilizando la primera vía se resume a un $\Theta(logn)$, por basarse en la teoría de la busqueda binaria, por tanto la segunda vía, aparte de realizar más operaciones matemáticas, su complejidad máxima queda como $\Theta(1)$, siendo esta solución una mejora de la librería al método de selección de parejas, ponderado por rango.

- **Selección ponderada por costo:** Para realizar la selección de parejas por el método ponderado por costo, debemos encontrar la probabilidad que presenta cada comosoma dentro de la población,

$$P_n = \left| \frac{C_n}{\sum_{n=1}^{N_{sobreviven}} C_n} \right| \tag{3.3}$$

donde C_n es el costo normalizado del cromosoma en cuestión, el mismo se obtiene restándole al costo real del cromosoma el menor costo de los cromosomas que se desecharon en la selección natural, $C_n = c_n - c_{N_{sobreviven}}$, posibilitando que el costo normalizado siempre sea negativo, de forma idéntica a la variante por rango, utilizamos la probabilidad acumulada de cada cromosoma para encontrar cuales serán seleccionados como padres. La probabilidad acumulada de cualquier cromosoma esta dada por:

$$PA_x = \left| \frac{\sum_{n=1}^{x} C_n}{\sum_{n=1}^{N_{sobreviven}} C_n} \right| \tag{3.4}$$

la sumatoria de todos los costos normalizados, $\sum_{n=1}^{N_{sobreviven}} C_n$, no tenemos que calcularla, ya que cada vez que insertemos o eliminemos un cromosoma, se tendrá un control de dicha suma, luego de tener los números generados con aleatoriedad, solamente debemos recorrer los cromosoma de forma ordenada por costo e ir llevando la suma de los costos anteriores, seleccionar un padre consiste en tener en cuanta el rango donde se van ubicando cada uno de los números aleatorios. Como observamos esta variante de la selección aleatoria ponderada, es más costosa que la ponderada por rango, puesto que al tener que recorrer, en el peor de los casos todos los cromosoma dentro de la estructura de datos, su cota máxima es $\Theta(n)$.

- **Variante de la selección ponderada por costo:** Para realizar esta variante de la selección por costo, nos basamos en la versión anterior, recordemos la fórmula 3.3, si realizamos la sustitución $C_n = -C_n$, quedaría:

$$PA_x = \frac{\sum_{n=1}^{x} C_n}{\sum_{n=1}^{N_{sobreviven}} C_n} \tag{3.5}$$

recordando que los cromosomas están ordenados en entreorden dentro de la estructura de datos, podemos plantear que la sumatoria acumulada de costos hasta un cromosoma cualquiera es a lo sumo:

$$\sum_{n=1}^{x} C_n \leq C_{actual} * X \tag{3.6}$$

siendo X la posición del cromosoma dentro de la población y suponiendo que los valores anteriores de costo sean iguales a el, para el cromosoma en la posición anterior (en el recorrido en entreorden, recordemos que están dentro de un ABB.) quedaría:

$$\sum_{n=1}^{x-1} C_n \leq C_{anterior} * (X-1) \tag{3.7}$$

luego podemos plantear que nuestra suma real se encuentra entre la suma máxima del anterior y la suma máxima del actual:

$$C_{anterior} * (X-1) < \sum_{n=1}^{x} C_n \leq C_{actual} * X \tag{3.8}$$

podemos aproximar la suma real a la mitad de este intervalo, como recordaremos de la versión anterior el rango para la selección de un cromosoma determinado es entre la probabilidad acumulada anterior y la propia, la cual encontraremos de la siguiente forma:

$$PA_x = \frac{C_{anterior} * (X-1) + C_{actual} * X}{2 * \sum_{n=1}^{N_{sobreviven}} C_n} \tag{3.9}$$

conociendo esto podemos plantear que esta variante de la selección por costo reduce la complejidad a $\Theta(logn)$, mejorando así la complejidad de la variante original, la cual debía recorrer a lo sumo todos los elemento del árbol.

- **Selección por torneo:** La complejidad máxima de esta selección es $\Theta(logn)$, puesto que escoger un elemento dentro del árbol se reduce a dicha complejidad.

3.2. Optimización multiobjetivo.

Comentábamos en la revisión bibliográfica que los mejores métodos basados en los algoritmos evolutivos para resolver optimización multicriterio, eran los métodos elitistas basados en la teoría de Pareto optimalidad, dentro de este grupo se encuentra, el ordenamientos por elementos no dominados(NSGA), algoritmo por fuerzas Pareto(SPEA), entre otros. En la librería la optimización multicriterio funciona de la siguiente forma:

Esquema de funcionamiento:

1. Crear elementos iniciales de la población.
2. Adicionar elementos a la población.
3. Realizar la selección natural, elimina los elementos mal adaptados al medio.
4. Aplicar operadores: cruce, mutación, cualquier otro que necesitemos.
5. Volver al paso 2 mientras no se encuentre la convergencia.

En el paso 1 se realiza la inicialización de los individuos, se evalúan cada uno en las funciones objetivos y restricciones, luego se realiza la inserción en la población y por tanto en el árbol rojo-negro, mediante un operador de comparación basado en el principio de Pareto optimalidad, el mismo funciona de la siguiente forma:

1. Se evalúan las restricciones del cromosoma i con el cromosoma que aparece en la primera posición de la lista del nodo raíz[1].
    - Si i presenta más violaciones de las restricciones que el cromosoma antes mencionado, se desciende al subárbol de derecho y se aplica nuevamente el operador.
    - Si i presenta menos violaciones, se desciende al subárbol izquierdo.

[1]los cromosomas que aparezcan en la lista, son no dominados entre ellos.

- Si presentan la misma cantidad de violaciones a las restricciones, i se inserta en el inicio de la lista del nodo correspondiente[2].

2. Si i no presenta violaciones a las restricciones, ni el cromosoma que aparece en la primera posición de la lista del nodo correspondiente.

   - Si i domina al cromosoma en cuestión, se desciende al subárbol izquierdo y se aplica nuevamente el operador
   - Si i es dominado por el cromosoma mencionado, se desciende al subárbol derecho.
   - Si el cromosoma en la primera posición e i son no dominados entre sí, el cromosoma i se adiciona a la lista del correspondiente nodo, luego se actualiza la distancia euclídea entre i y los demás cromosomas de la lista, para el cálculo de dicha distancia solamente se tienen en cuenta los valores de los objetivos, luego se ordena la lista en orden descendente dada la distancia antes mencionada.

En caso de no existir nodo a la izquierda o derecha, se crea e inicializa con el nuevo cromosoma. Como podemos observar los cromosomas mejor adaptados quedarán más a la izquierda y los peores a la derecha, luego de aplicar la selección natural, quedarán los mejores y más aptos, encargados de crear la próxima generación, brindándole el rasgo elitista al algoritmo[3].

3.3. Algoritmos evolutivos paralelos.

En otras secciones se explicaba que los AE paralelos agregaban un elemento más a los tradicionales, este elemento se encarga de enviar los elementos más adaptados a la vecindad. La librería propone un comunicador genérico, el cual funciona para los cromosomas con cantidades variables y constantes de genes. El comunicador(`PEAMigration`) permite especificar la cantidad de cromosomas a enviar y recibir, permite enviar elementos aleatorios o los más adaptados, posibilitando escoger la vía más conveniente para nuestro problema.

[2] se toma como hipótesis que los cromosomas en generaciones futuras son mejores que los actuales.

[3] los algoritmos evolutivos multiobjetivos se consideran elitistas, solo sí los individuos mejor adaptados son los encargados de crear la nueva generación.

En el caso de ser la evaluación de un cromosoma menos costosa que enviar los propios valores de la evaluación, es posible especificarle una de las dos opciones. Como estrategia evolutiva tal vez no sea necesario enviar y recibir elementos en todas la generaciones, es posible indicarle la frecuencia con que se realizarán los envíos. El propio comunicador luego de encontrar la convergencia, se utiliza para recoger el resultado una vez finalizada la corrida del algoritmo evolutivo.

El modelo usado en el comunicador es el modelo de grano grueso, donde se envía una parte del problema para cada nodo del cluster. Cada nodo del cluster o procesador enviará su selección al vecino siguiente basado en la topología de anillo o circular (figura 1.6). Dentro del comunicador se utilizan las funciones de envío y recepción asincrónico, además de envío y recepción con reemplazo; funciones especificadas en el estándar de MPI[42]. La funcionalidad del comunicador se basa en la implementación que le demos a nuestro gen. En la propia interfaz del gen(`PEAGene`) están definidos los métodos a redefinir encargados de realizar el empaquetamiento de los atributos necesarios para nuestro problema.

Si deseamos crear nuestro propio comunicador este debe heredar de la clase comunicador, debemos tener en cuenta que el propio comunicador es el encargado de recolectar de todos los procesos los mejores cromosomas una vez alcanzada la convergencia del algoritmo.

3.4. Pruebas de eficiencia, discusión.

Luego de realizar la implementación de la librería con todo lo mencionado en las secciones anteriores, creamos un caso de prueba. El gen de dicho caso de prueba presenta un vector de longitud variable como atributo. Las pruebas iniciales se realizaron con una longitud de 100 elementos y en cada cromosoma 1 gen. Los operadores que se utilizaron fueron la selección de parejas de forma aleatoria, el cruzamiento por reemplazo, y una mutación. La mutación consiste en eliminar primeramente la misma cantidad de elementos mal adaptados que elementos a mutar, luego los elementos a mutar no se extraen de la población, solo se copian y se realizan los cambios en las copias.

El porciento de la población que sobrevive a la próxima generación es el 70 %, solo realizan la mutación un 30 %, se realizaron 200 iteraciones del algoritmo tanto para el secuencial como para el paralelo. En las pruebas del paralelo la migración se realiza en todas las generaciones enviando y recibiendo 15 elementos cada vez. Las funciones objetivos definidas son funciones sencillas que nos permitieron explorar la funcio-

nalidad de la librería. Los elementos dentro del vector en cada gen son números enteros en el rango de $[-1000, 1000]$.

Se definieron cuatro funciones objetivos: la primera calcula la suma total de los cien elementos, la segunda calcula la suma de los elementos múltiplos de dos, la tercera la suma de los múltiplos de tres y la cuarta la suma de los múltiplos de cinco. Se definió una restricción, en la misma se limitaban los elementos negativos en cada cromosoma, se realizaron varias pruebas tanto con un procesador como con ocho procesadores. Luego de finalizar las pruebas se hizo una evaluación inicial de los resultados mediante las métricas por las cuales se evalúan los algoritmos paralelos.

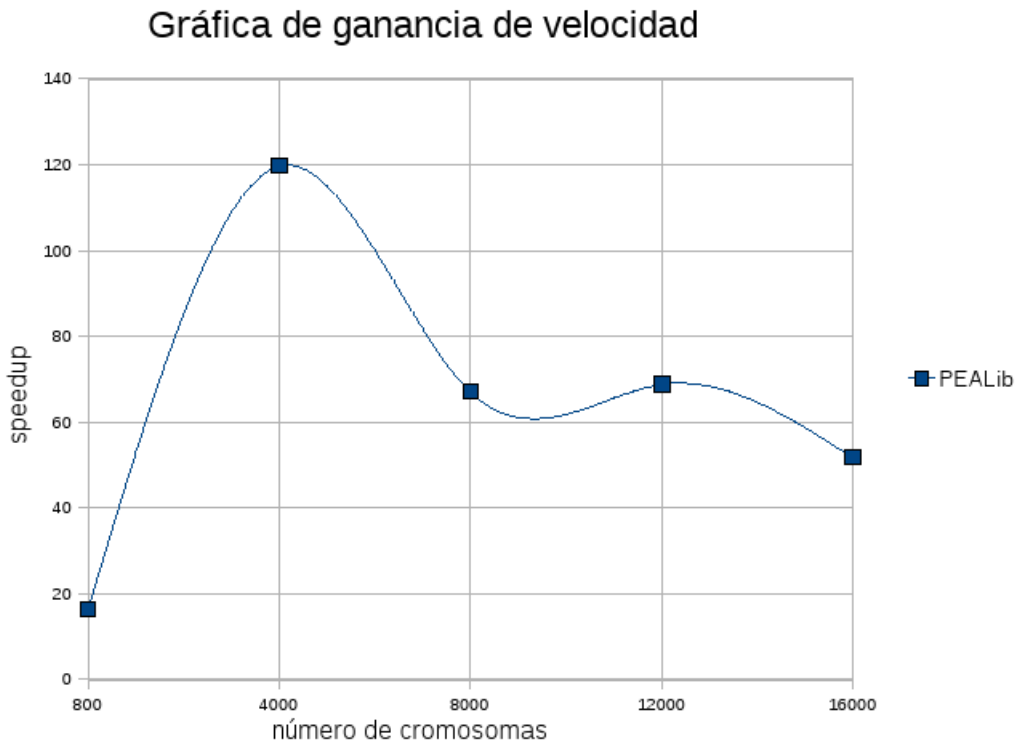


Figura 3.3: SpeedUp de las pruebas iniciales.

Recordaremos que el speedup(3.3) es una métrica que indica que tan rápido es el algoritmo paralelo respecto a la versión secuencial. La evaluación de la métrica es aceptada como buena en cualquier algoritmo paralelo, si es cercana a la cantidad de procesadores en los cuales se realizó la corrida del algoritmo paralelo, en nuestro caso fueron ocho procesadores. Podemos observar como el speedup más bajo esta por encima de

ocho, realizando un pico de hasta 120, este fenómeno es conocido como super-speedup. El super-speedup se debe al uso adecuado de la cache y otros paramétos. Esta prueba inicial fue satisfactoria porque demuestra la ventaja en cuanto a rapidez del algoritmo paralelo.

Si observamos la gráfica de eficiencia(3.4) del algoritmo paralelo respecto al secuencial, nos percatamos que presenta una eficiencia excelente, recordemos que la evaluación de la eficiencia es un valor cercano a uno.

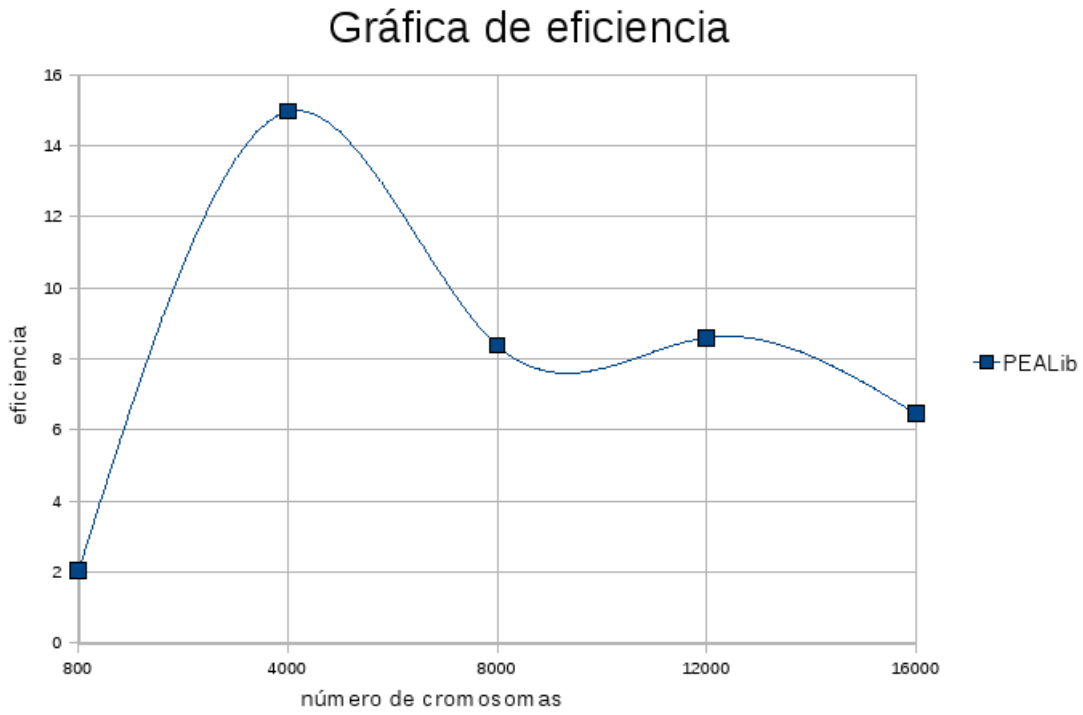


Figura 3.4: Eficiencia de las pruebas iniciales.

En la gráfica de tiempo(3.5) se observa el tiempo total promedio de todas la pruebas realizadas tanto con un procesador como con ocho.

Estas pruebas iniciales demostraron la funcionalidad de la librería a largo plazo. Si observamos con detenimiento nos falto expresar la escalabilidad del algoritmo paralelo de forma gráfica, realmente el algoritmo paralelo es muy bueno en ciertas configuraciones de las pruebas, pero fue incapaz de mantener esa ganancia de velocidad y eficiencia cuando el número de cromosomas se acerca al infinito. Este fenómeno se debe a

Figura 3.5: Tiempo paralelos y secuencial de las pruebas iniciales.

la complejidad del árbol rojo-negro, recordaremos que esta estructura se modificó para que aceptara elementos repetidos, esto se logró adicionando a cada nodo un lista doblemente enlazada que contrendría los elementos que coincidan en un posición. Por parte de la optimización multiobjetivo recordamos que dicha lista se debía recorrer inicialmente y luego ordenarla, alcanzando complejidad máxima de $\Theta(nlogn)$, por tanto según aumentaran los elementos en la misma, mayor sería el tiempo de espera por esta operación.

El modelo matemático sobre el cual estaba soportado la optimización multiobjetivo no podía mejorarse más, por la parte computacional se pensó en acotar el tamaño que podría tener la lista en cada nodo, acotando a su vez el tiempo de espera. Esta operación no incurría en deficiencia sobre el modelo matemático que lo sustentaba, ya que si un cromosoma quedaba no dominado en un nodo cualquiera del árbol y la lista del mismo estaba en su máxima extensión se tomaba el cromosoma como peor que los ya existentes y se pasaba el control del mismo al subárbol derecho, si los elementos en el mismo son peores que el propio cromosoma se pasaba el control al subárbol izquierdo, si es peor por un motivo o por otro, se sede el control nuevamente al subárbol derecho, esto implica que solo pasarán al subárbol izquierdo aquellos cromosomas realmente buenos, si pensamos que por motivos de normales en cualquier árbol rojo-negro[41], solo se puede puede descender una cantidad limitada de veces antes de que la propia estructura de datos realice

las rotaciones, esto no afectaría el modelo, porque siempre quedarían los mejores delante, intermedio los no tan buenos y finalmente los malos, si realizaramos un recorrido en entreorden sobre el árbol.

Nuevamente se realizaron las pruebas, podemos observar un speedup(3.6) más normalizado sobrepasando aún las expectativas, esto se debe a que mejorar el paralelo implicaba mejorar el secuencial, siendo este en casos de escasa población más rápido que el propio paralelo.

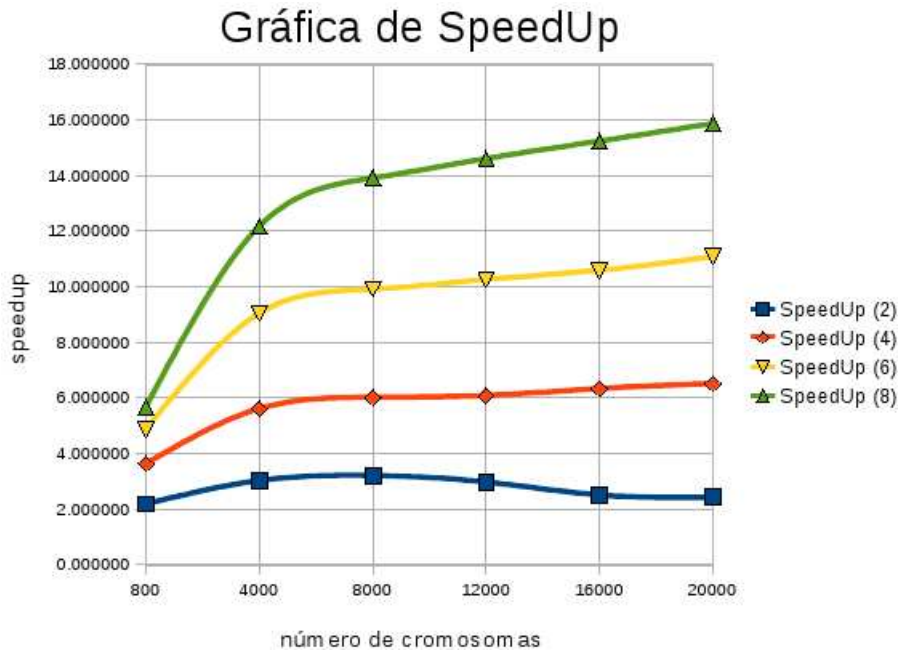


Figura 3.6: SpeedUp de las pruebas de rendimiento.

En la nueva gráfica de eficiencia(3.7) observamos como a medida que aumenta el número de cromosomas a evaluar, la misma se mantiene por encima de uno.

En la gráfica de escalabilidad o speedup escalado(3.8) se puede observar como para cantidades crecientes de procesadores y del tamaño del problema, la librería se comporta de forma eficiente y mantiene esa eficiencia cuando el número de procesadores aumenta conjunto con la cantidad de cromosomas a evaluar.

Finalmente podemos observar como la gráfica de tiempo(3.9) del paralelo respecto al secuencial, donde el tiempo paralelo promedio se mantiene muy por debajo, podemos ver además como el secuencial para la configuración de las pruebas ya no es tan costoso.

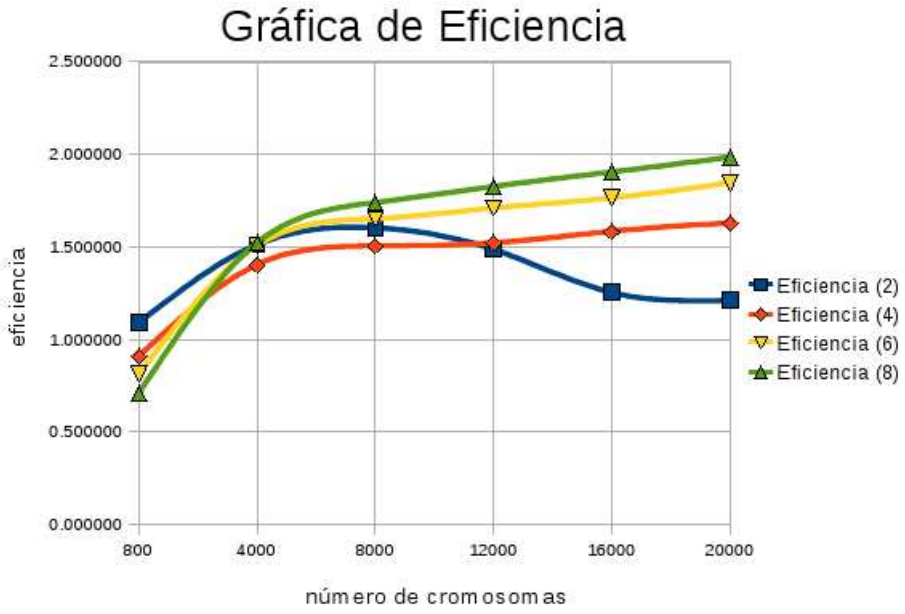


Figura 3.7: Eficiencia de las pruebas de rendimiento.

Estas pruebas de rendimiento aseguran que la librería esta lista para ser utilizada en la resolución de los casos de estudios propuestos en la tesis.

3.5. Selección del portafolio de proyecto.

Las organizaciones se enfrentan fundamentalmente al problema de cómo invertir sus recursos escasos entre un conjunto de alternativas o proyectos candidatos. Para ayudar a dar respuesta a este problema se presenta el siguiente estudio[15], en el se persigue seleccionar, en función de un conjunto de criterios y recursos existentes, una cartera de proyectos en la que los proyectos que la compongan estén distribuidos en el tiempo de la manera más eficiente posible teniendo en cuenta aspectos como las interacciones entre proyectos y la existencia de múltiples criterios en los que debemos hacer hincapié para que dicha selección sea la adecuada. Hay muchos y muy diferentes métodos para la evaluación y selección de una cartera de proyectos, cada uno con sus ventajas e inconvenientes, sin embargo, no existen modelos que de manera integrada agreguen todos aquellos aspectos que tradicionalmente han sido considerados fundamentales. En esta investigación se presenta un modelo global que integre todos estos aspectos y que pretende dar una respuesta universal al problema de la selección y programación de cartera de proyectos.

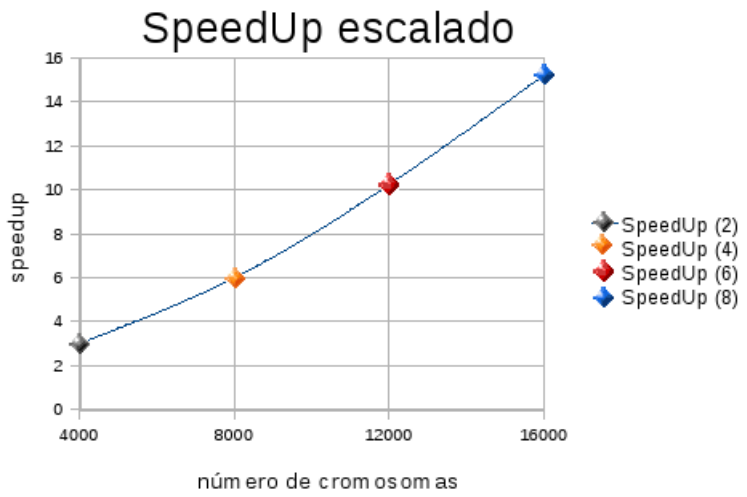


Figura 3.8: SpeedUp escalado de las pruebas de rendimiento.

Siguiendo las pautas del artículo[15] podemos percatarnos que podemos definir la cartera de proyectos como una lista de proyectos, donde solo los que estén serán los activos, dicha lista es la estructura que mantendrá los genes presente en el cromosoma, donde los genes estarán representados por proyectos. Un gen(Proyecto) esta compuesto por un instante de tiempo que indica el comienzo, el identificador del proyecto, y los demás atributos necesarios, ingresos, gastos, personas, instantes de tiempos de duración, etc.

Dado el instante de tiempo del comienzo y los instantes de duración se puede conocer cuantos períodos de tiempos atraviesa el proyecto y su instante de culminación. Este problema en general es multiobjetivo y cargado de restricciones, ya que en dependencia de nuestro caso real, puede que necesitemos estrechar el espacio de busqueda de nuestra solución.

En la resolución del caso de estudio la creación de las funciones objetivos no fue complicada, por el contrario las restricciones necesitaron de un nivel de profundización en el propio problema de la selección de la cartera de proyectos. Existen varios tipos de restricciones, están las restricciones relativas a recursos disponibles en la organización, como pueden ser las personas, la formulación de esta restricción es complicada porque se debe de tener en cuenta los períodos de tiempo sobre los cuales se desarrollarán los proyectos dentro de la cartera, además de la cantidad disponible en cada período del recurso en cuestión(5). Están las restricciones de dependencia entre proyectos, de tiempo de duración y restricciones globales.

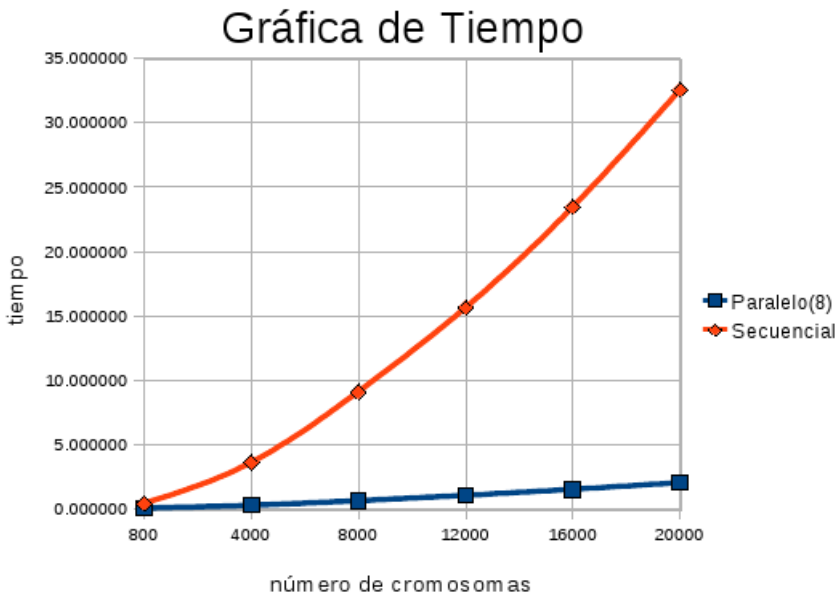


Figura 3.9: Tiempo paralelos y secuencial de las pruebas de rendimiento.

El caso de estudio presenta la siguiente estructura:

Proyecto	Costo	Ingresos	Días	Personas
$P1$	48	47	60	29
$P2$	38	41	189	48
$P3$	40	18	105	51
$P4$	43	38	174	23
$P5$	35	342	191	20
$P6$	25	375	55	51
$P7$	26	122	196	47
$P8$	41	152	120	34
$P9$	53	81	208	51
$P10$	81	67	59	33
$P11$	97	71	224	28
$P12$	51	153	82	40
$P13$	89	74	66	52
$P14$	78	29	176	24

$P15$	97	62	109	28
$P16$	90	90	235	50
Restricciones globales	≤ 500		≤ 1300	≤ 200

Tabla 3.3: Caso de estudio para la selección de la cartera de proyecto.

donde se definió como única función objetivo el maximizar las ganancias, se definieron como restricciones las siguientes:

- Solo ocurrirán dos períodos de tiempo de 180 días cada uno.
- Solo se pueden ocupar 200 personas en cada período.
- El proyecto $P16$ solo se puede hacer si se hace $P2$.
- El proyecto $P15$ solo se puede hacer si se ha hecho $P10$.
- No pueden existir proyectos repetidos.
- El costo total de todos los proyectos en la cartera no debe sobrepasar las 500 unidades.

Para dar cumplimiento a la primera restricción solo debemos buscar que no existan proyectos que su instante de culminación sea mayor que el instante final del último período(1).

Algoritmo 1 Periodos disponibles para la realización del portafolio

Entrada: $periodos = periodo_1, \ldots, periodo_n$

1: **para todo** *proyecto en la cartera de proyectos* **hacer**
2: **si** $proyecto_i(comienzo) + proyecto_i(duración) > periodo_n(final)$ **entonces**
3: *indicar que existe la violación*
4: **fin si**
5: **fin para**
6: *indicará que no ocurre la violación*

La segunda restricción se cumplió mediante el algoritmo(5). Para dar cumplimiento a la tercera restricción debemos percatarnos que solo se puede desarrollar $P16$ de forma correcta sí a la vez ocurre $P2$, como en el cromosoma los proyectos no están ordenados por el instante de comienzo, debemos recorrer la lista de genes y separar los proyectos $P16$ y los $P2$ en dos estructuras, luego buscar que para cada proyecto $P16$ exista un $P2$ que comience en el mismo instante(4). Para la siguiente restricción el proyecto $P15$ solo se puede hacer si se ha hecho $P10$, simplemente debemos recorrer la lista de genes y verificar que no sea ejecutado ningún proyecto $P15$ sin que haya finalizado algún proyecto $P10$(3). Para conocer si existen proyectos repetidos en cada cromosoma o cartera de proyecto, debemos buscar que cada proyecto no sea igual a los restantes proyectos del portafolio(2). Para conocer el costo total de la cartera, simplemente es recorrer la lista de proyectos y verificar que la suma total no supere las 500 unidades.

Algoritmo 2 Restricción de proyectos repetidos

1: *contador por cada proyecto en el caso de estudio = inicializado vacio*
2: **para todo** *proyecto en la cartera de proyectos* **hacer**
3: **si** $contador(proyecto_i(identificador)) = vacio$ **entonces**
4: $contador(proyecto_i(identificador)) = ocupado$
5: **si no**, **si** $contador(proyecto_i(identificador)) = ocupado$ **entonces**
6: *indicará que ocurre la violación*
7: **fin si**
8: **fin para**
9: *indicará que no ocurre la violación*

Podemos observar que cada cromosoma será una cartera de proyecto en potencia, la misma se evaluará y comparará con las demás carteras. Sobre la población de cromosomas se aplicará el operador de cruzamiento, el mismo seleccionará dos números aleatorios uno para cada padre, que dividirán a ambos en dos partes, luego con la primera parte del primer padre y la última del segundo se formará el primer descendiente, con las partes restantes el segundo. Se aplicarán además dos operadores de mutación, los cuales cambiarán aleatoriamente un gen dentro de una copia del cromosoma seleccionado, en el gen cambian el identificador del proyecto o el instante de comienzo, luego se realiza un torneo binario con ambos cromosoma y se ingresa a la población el mejor.

La cantidad de individuos que sobreviven a la próxima generación es el 50 %, y solo un 0,03 % de los individuos realizarán las mutaciones. Al ser la selección de la cartera de proyecto un problema combinatorial, se realizaron 150 generaciones para alcanzar un resultado aceptable, en el caso de realizar la corrida en varios procesadores la migración se realizó cada 75 generaciones enviando y recibiendo 2 cromosomas de forma aleatoria por cada llamada al operador.

Explicábamos en secciones anteriores que el paralelismo es necesario si queremos disminuir el tiempo de procesamiento de un problema dado, o aumentar su exactitud; en el caso de la selección de la cartera de proyecto el paralelismo se usa tanto para disminuir el tiempo de cómputo como para aumentar la exactitud de los resultados. En las pruebas realizadas para una población de 100000 individuos y los parámetros antes mencionados se obtienen buenos resultados, pero al ser este caso de estudio un problema combinatorial, pueden existir muchas soluciones que sean aceptadas como buenas. Si realizamos las corridas del algoritmo sobre varios procesadores, esto puede aumentar el área revisada del espacio de solución, aumentando por tanto las posibles soluciones, brindándole a los especialistas una mayor cantidad de variantes para escoger su portafolio de proyecto ideal.

Para aumentar el área revisada podemos aumentar la población, esto llevaría consigo un aumento del tiempo de cómputo, realizar varias corridas del propio problema o ejecutar la aplicación de forma paralela. Las pruebas fueron realizadas para una población de 800000 individuos en un procesador y la misma cantidad de cromosomas pero sobre una cantidad creciente de procesadores. En la gráfica(3.10) se observa como para una misma cantidad de cromosomas y escalando el número de procesadores podemos obtener buenos resultados y disminuir el tiempo de cómputo.

Recomendamos no asignarle menos de 100000 individuos a cada procesador para obtener buenos resultados en cuanto a calidad de las soluciones. Siguiendo lo planteado anteriormente se puede observar como el speedUp escalado(3.11) del caso de estudio se mantiene de forma creciente según aumenta el tamaño del problema y el número de procesadores.

Podemos observar la eficiencia(3.12) del algoritmo paralelo respecto al secuencial, reiterando la ventaja de realizar la ejecución de problema selección de la cartera de proyecto en entornos paralelos.

Algoritmo 3 Restricción del proyecto P15

1: $finP10 = inicializado\ como\ el\ mayor\ instante\ de\ tiempo$
2: $iniP15 = inicializado\ como\ el\ mayor\ instante\ de\ tiempo$
3: **para todo** *proyecto en la cartera de proyectos* **hacer**
4: **si** $proyecto_i\,(identificador) = 10\ y\ proyecto_i\,(comienzo + duración) < finP10$ **entonces**
5: $finP10(proyecto_i(comienzo + duración))$
6: **si no**, **si** $proyecto_i\,(identificador) = 15\ y\ proyecto_i(comienzo) < iniP15$ **entonces**
7: $iniP15(proyecto_i(comienzo))$
8: **fin si**
9: **fin para**
10: **si** $iniP15 = mayor\ instante\ de\ tiempo$ **entonces**
11: *indicar que no ocurre violación*
12: **fin si**
13: **si** $finP10 = mayor\ instante\ de\ tiempo$ **entonces**
14: *indicar que ocurre violación*
15: **fin si**
16: **si** $iniP15 \leq finP10$ **entonces**
17: *indicar que ocurre violación*
18: **fin si**

3.6. Problema inverso aditivo de los valores singulares(PIAVS)

Dado un conjunto de matrices $A_0, A_1, \ldots, A_n \in \Re^{m\times n}(m > n)$ y un conjunto de números reales $S^* = \{S_1^*, S_2^*, \ldots, S_n^*\}$, tales que $S_1^* \geq S_2^* \geq \cdots \geq S_n^* > 0$, este problema consiste en encontrar un vector de números reales $c = [c_1, c_2, \ldots, c_n]^T$, tal que los valores singulares de $A(c) = A_0 + c_1A_1 + \cdots + c_nA_n$ sean los elementos de S^*. El PIAVS se caracteriza por tener más de una solución, es decir, pueden existir c_X y c_Y tales que los valores singulares de $A(c_X)$ y $A(c_Y)$ sean S^*. Por ejemplo, suponer $m = n = 3$, con

$$\begin{matrix} A_0 \\ \begin{bmatrix} 1 & 3 & 5 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{bmatrix} \end{matrix} \quad \begin{matrix} A_1 = e_1e_1^t \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} A_2 = e_2e_2^t \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} A_3 = e_3e_3^t \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{matrix},$$

existen $c_X = [3, 2, 1]^T$ y $c_Y = [1, 2, 3]^T$ tales que los valores singulares de $A(c_X)$ y $A(c_Y)$ son $S^* =$

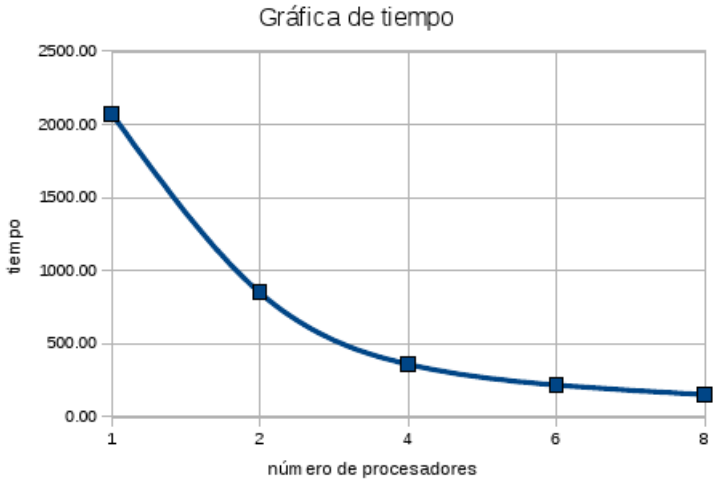


Figura 3.10: Tiempo de cómputo para una población de 800000 cromosomas.

$\{9.5, 1.8, 0.9\}$. Se denota con c^* a una de las soluciones del PIAVS, es decir el vector de números reales $c^* = [c_1^*, c_2^*, \ldots, c_n^*]^T$, tal que los valores singulares de $A(c^*)$ son S^*. Típicamente el PIAVS se resuelve con aproximaciones calculadas por métodos iterativos, por lo que se denota con $c^{(0)}$ la aproximación inicial.

De forma general, el objetivo básico de un problema inverso de valores singulares es reconstruir los parámetros de cierto sistema a partir del conocimiento de su comportamiento dinámico. Estos comportamientos dinámicos dependen de las aplicaciones donde surjan estos problemas. Algunas de estas aplicaciones son:

En *tomografía*, se ocupa sobre la reconstrucción de una función, usualmente una distribución de densidad. Esta aplicación es de gran importancia en aplicaciones médicas y en pruebas no-destructivas. Matemáticamente, esto se conecta con la inversión de la Transformada Radon[43].

Los métodos de SVD son aplicados en antenas. El diseño de antenas se presenta como el problema inverso de encontrar la estructura que brinda crecimiento a un patrón específico de radiación a campo abierto. Los patrones de radiación del cable de una antena pueden ser fácilmente calculados a partir de la distribución a través de su longitud[44].

En la *teoría del transporte de neutrones*, también se tiene aplicación de los problemas inversos de valores propios y valores singulares, donde se trabaja en la dinámica de una red neuronal aditiva, buscando llegar

Algoritmo 4 Restricción del proyecto P16

1: $proyectos_{p16} = inicializado\ vacio$
2: $proyectos_{p2} = inicializado\ vacio$
3: **para todo** *proyecto en la cartera de proyectos* **hacer**
4: **si** $proyecto_i\,(identificador) = 16$ **entonces**
5: $proyectos_{p16}(proyecto_i)$
6: **si no**, **si** $proyecto_i(identificador) = 2$ **entonces**
7: $proyectos_{p2}(proyecto_i)$
8: **fin si**
9: **fin para**
10: **para todo** *proyecto en* $proyectos_{p16}$ **hacer**
11: **si** *proyecto*(*comienzo*) *no existe en* $proyectos_{p2}$(*comienzos*) **entonces**
12: *indicar que existe la violación*
13: **fin si**
14: **fin para**
15: *indicará que no ocurre la violación*

a un estado de equilibrio por medio de los valores propios de la matriz Jacobiana[45].

Para programar el caso de estudio se creo un gen que presenta un vector de longitud variable como atributo. Con el vector se construye la matriz de Toeplitz perteneciente a cada cromosomas y a la cual se le calculan los valores singulares. Nos apoyamos en la librerías BLAS y LAPACK para la realización de operaciones con matrices y vectores. El caso de estudio consiste en crear inicialmente una matriz de Toeplitz, de la cual se hallarían los valores singulares λ^*, estos valores singulares se fijan y luego en cada generación del algoritmo evolutivo se tratará de encontrar la matriz que genere dichos valores singulares. La función objetivo de nuestro problema de optimización es minimizar la distancia euclídea entre los valores singulares de la matriz en cada cromosoma de la población $\lambda(A(x))$ y los valores singulares esperados λ^*, es decir $F(x)_{min} = |\lambda(A(x)) - \lambda^*|$.

Se asignaron 80 individuos a cada procesador, la selección natural fue del 50 % y la mutación del 25 %, la migración se realizó cada 10 generaciones donde enviaban los 5 mejores individuos, la congervencia se alcanzaría siempre que en cualquiera de los procesadores existiera algún individuos cuya evaluación de la

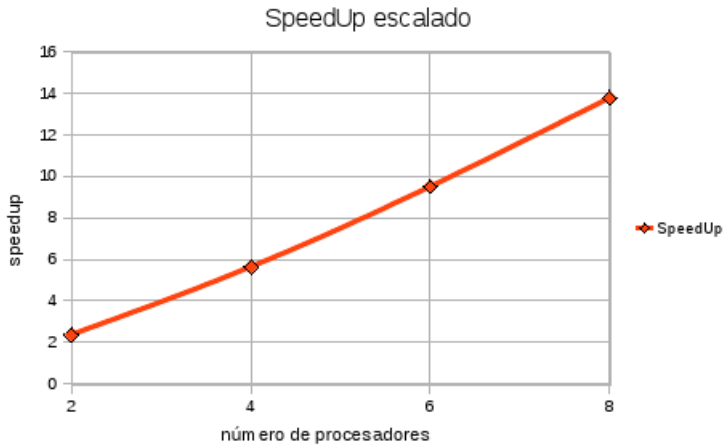


Figura 3.11: SpeedUp escalado de la selección del portafolio de proyecto.

Figura 3.12: Eficiencia de la selección del portafolio de proyecto.

función objetivo fuera menor que 0.01. Los operadores que se utilizaron fueron la selección de parejas por el método de la ruleta, el cruzamiento y una mutación híbrida con el método de Newton-Bisección[2, 46], el cual optimiza todas las coordenadas del vector en cada individuo seleccionado. Se realizaron pruebas para matrices de dimensiones 20×20, 30×30 y 40×40, para cada dimensión se fijaron 3 valores singulares distintos, se realizaron con cada uno de los valores singulares fijados, 10 corridas del algoritmo, alcanzando las siguientes estadísticas.

MxN	Procesadores	Prom. de iteraciones	Npop	Prom. Tiempo	Convergencia	Error R. Prom.
20x20	8	1.43	640	13.86	-3.25E-004	2.26E-001

20x20	1	1.33	640	93.73	-3.60E-008	6.46E-001
30x30	8	4.40	640	86.09	-2.83E-005	7.92E-002
30x30	1	4.38	640	660.29	-3.08E-004	2.68E-006
40x40	8	22.90	640	893.95	-4.63E-004	1.37E-001
40x40	1	17.52	640	5735.01	-2.72E-007	7.17E-002

Tabla 3.4: Pruebas del PIAVS para 8 procesadores.

MxN	Procesadores	Prom. de iteraciones	Npop	Prom. Tiempo	Convergencia	Error R. Prom.
20x20	6	1.4	480	13.24	-3.97E-008	4.61E-001
20x20	1	1.4	480	77.74	-8.14E-005	2.69E-001
30x30	6	5	480	96.79	-1.08E-006	2.00E-001
30x30	1	4	480	457.94	-1.75E-007	1.57E-001
40x40	6	34.37	480	1307.62	-2.32E-004	1.55E-001
40x40	1	26.67	480	5762.06	-2.86E-004	5.50E-001

Tabla 3.5: Pruebas del PIAVS para 6 procesadores.

Se puede observar que los valores de la ganancia de velocidad y la eficiencia son valores discretos con respecto al caso de estudio anterior, precisamente por eso este problema es ideal correrlo sobre una maquina paralela, ya que es un problema complejo, la evaluación de la función objetivo es costosa y es un problema de optimización que puede llegar a la convergencia en dependencia de que tan cerca estén los elementos a evaluar del criterio de parada, es decir, al aumentar los procesos que busquen de forma aleatoria sobre el área de solución se puede reducir considerablemente el tiempo de computo.

Conclusiones del capítulo.

En este capítulo se demostró la eficiencia de la librería y su desempeño exitoso en los casos de estudios propuestos. Se planteó la técnica multiobjetivo que usa la librería para la optimización de este tipo de

Algoritmo 5 Recursos disponibles en la organización

Entrada: $periodo = periodo_1, \ldots, periodo_n$

Entrada: $recursos = recurso_1, \ldots, recurso_n$

1: *línea de tiempo = cada instante inicia en* 0
2: **para todo** *proyecto en la cartera de proyectos* **hacer**
3: *obtener el periodo de inicio del proyecto*
4: **mientras** *queden días por trabajar* **hacer**
5: **si** *el proyecto no se finaliza dentro del periodo donde comienza* **entonces**
6: *se piden recursos en la línea de tiempo*
7: *se reducen los días de duración*
8: *se comienza el proyecto nuevamente en el próximo periodo*
9: **si no**
10: *se piden recursos en la línea de tiempo*
11: *se reducen los días de duración a* 0
12: *se liberan los recursos ocupados*
13: **fin si**
14: **fin mientras**
15: **fin para**
16: **para todo** *instante en la línea de tiempo* **hacer**
17: *se comprueba por periodo que no se soliciten más recursos que los disponibles*
18: **fin para**
19: *indicará si ocurre o no la violación*

problemas, como es el caso de la selección del portafolio de proyecto, se realizó una hibridación de los algoritmos evolutivos con un método cuasi-Newton para darle una solución eficiente al problema inverso aditivo de valores singulares. Se logró en este capítulo aplicar la librería a problemas costosos computacionalmente, alcanzando una reducción considerable del tiempo de procesamiento.

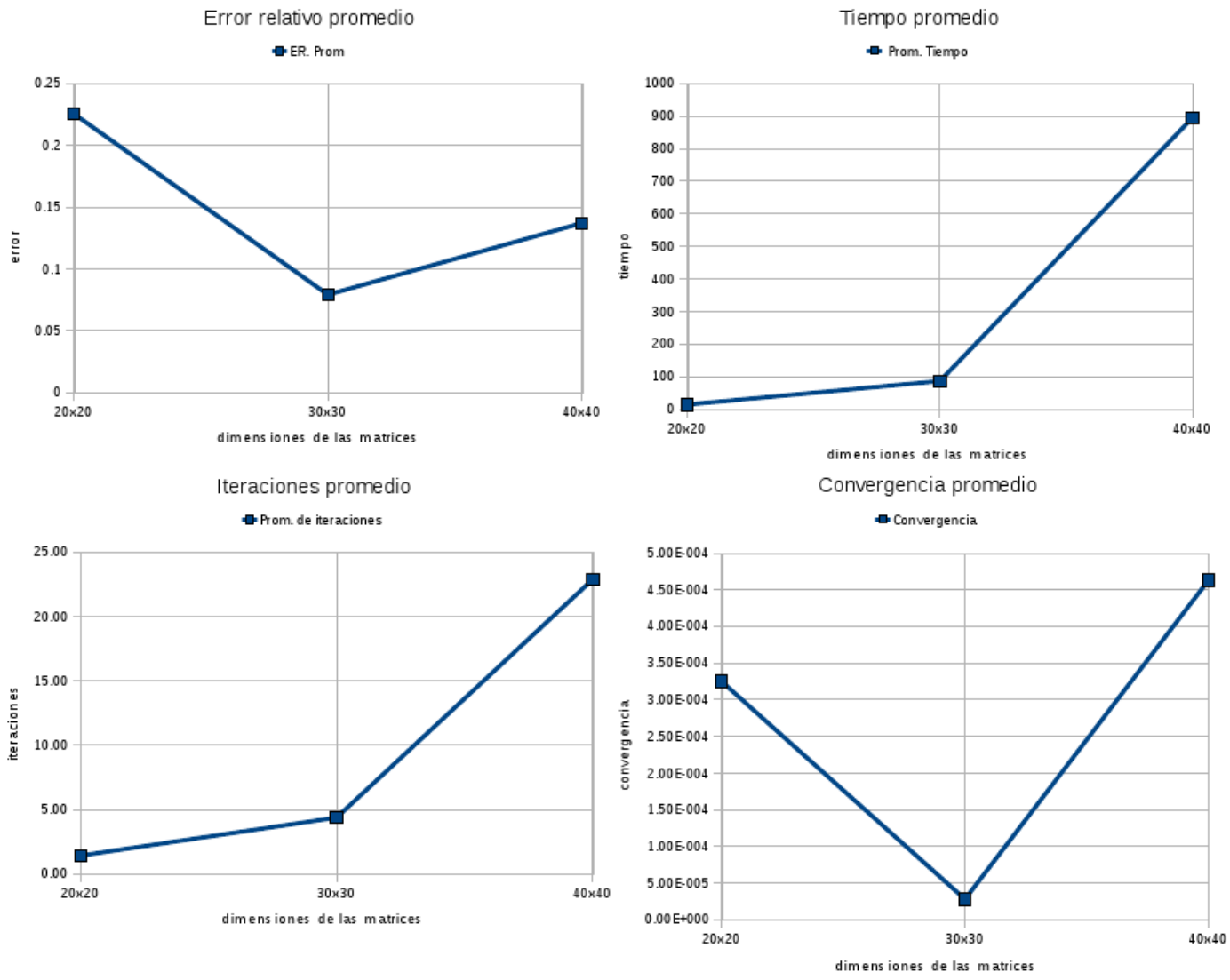


Figura 3.13: Pruebas para 8 procesadores.

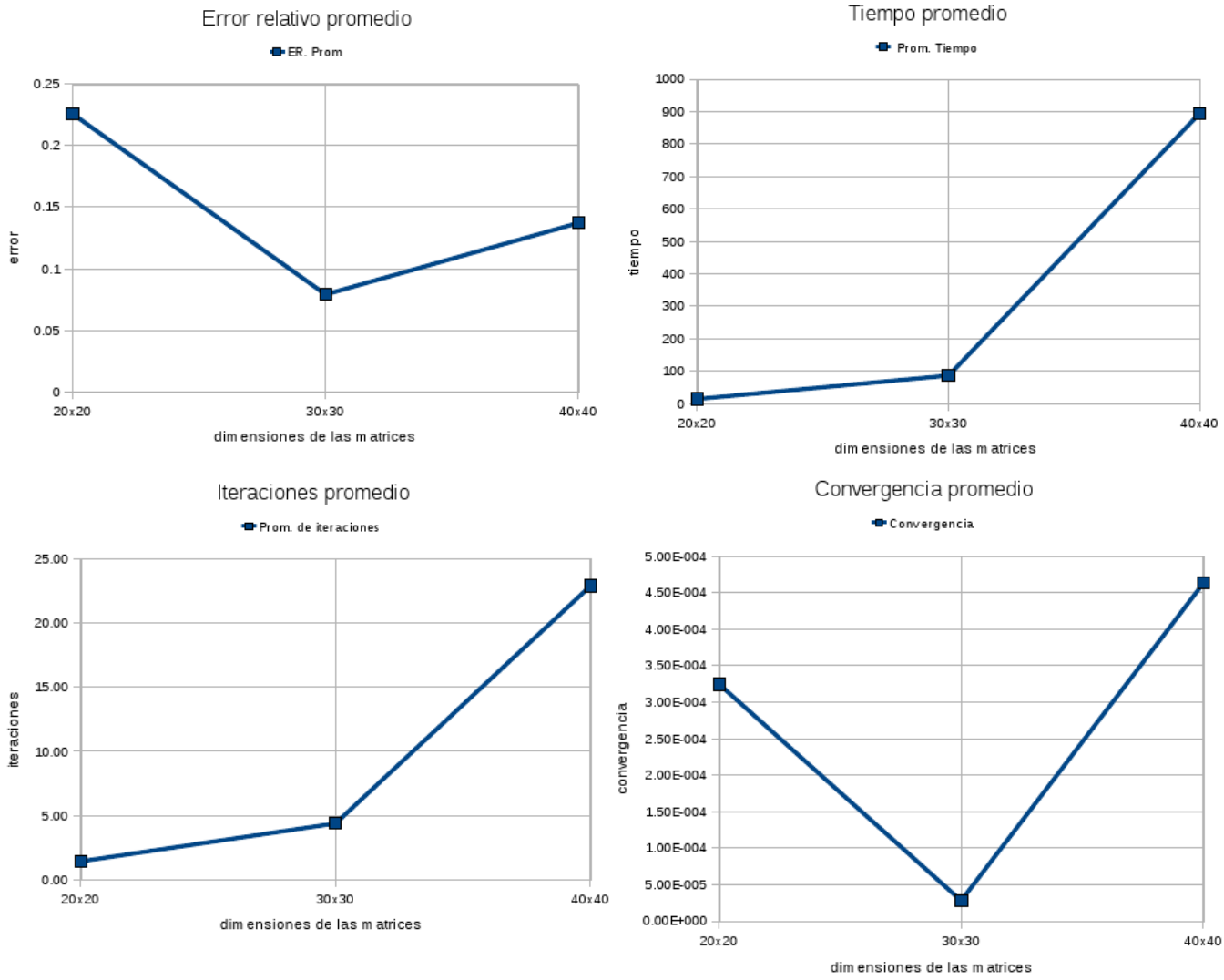


Figura 3.14: Pruebas para 6 procesadores.

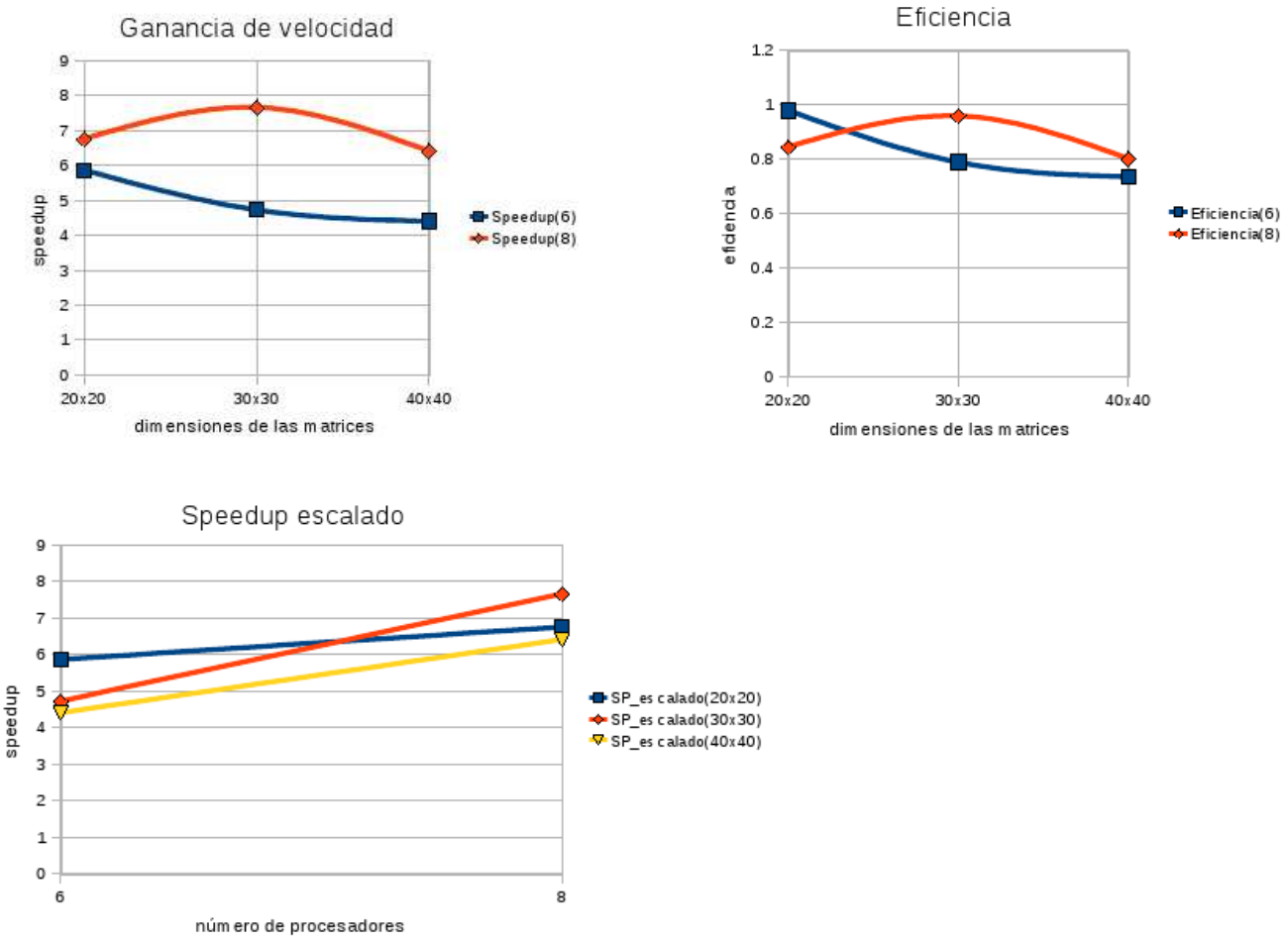


Figura 3.15: Gráficas de ganancia de velocidad y eficiencia.

Conclusiones

Se realizó el diseño y la implementación de la librería paralela de algoritmos evolutivos, en la cual se redujo la complejidad de los métodos de selección de parejas y las demás funciones de dichos algoritmos. Se demostró que es eficiente su aplicación a problemas de optimización computacionalmente costosos, como es el caso del problema multiobjetivo, selección del portafolio de proyecto y el caso del problema inverso aditivo de valores singulares.

Para demostrar la efectividad de la librería se realizaron pruebas de rendimiento en la sección *pruebas de eficiencia, discusión*; luego en la resolución de cada uno de los casos de estudio se realizaron pruebas de ganancia de velocidad y eficiencia, demostrando ser óptima la aplicación de la librería en dichos problemas.

Para el caso de estudio selección del portafolio de proyecto, se necesitó revisar las técnicas evolutivas para la optimización multiobjetivo, de las misma no se escogió ninguna y se realizó una propuesta para el cálculo multiobjetivo, demostrando ser eficiente. En el caso de estudio problema inverso aditivo de valores singulares se realizó una hibridación de los algoritmos evolutivos con un optimizador local basado en Newton, especificamente una híbrido de Newton con el método de Bisección, demostrando ser la hibridación del algoritmo evolutivo con el optimizador local una mejor opción que el algoritmo evolutivo puro, o mezclado con cualquier otro optimizador local.

Referencias bibliográficas

[1] Holland J. Adaptation in Natural and Artificial Systems. 1975;p. 183.

[2] Rasúa RAT. Paralelización de Algoritmos de Optimización por Búsqueda Directa. Universidad Politécnica de Valencia; 2005.

[3] Randy L Haupt, Sue Ellen Haupt. PRACTICAL GENETIC ALGORITHMS. A JOHN WILEY & SONS, INC., PUBLICATION; 2004.

[4] Oh IS, Lee JS, Moon BR. Hybrid Genetic Algorithms for Feature Selection. 2004 Nov;26(11).

[5] Whitley D. Modeling Hybrid Genetic Algorithms. 2007;.

[6] Alvarado C, Herazo I, Ardilla C, Donoso Y. Aplicación de NSGA-II y SPEA-II para la optimización multiobjetivo de redes multicast. Ingeniería & desarrollo. 2005 Enero-Junio;(17).

[7] Banka H, Mitra S. Evolutionary Biclustering of Gene Expressions;.

[8] Carpena GS. DINEÑO Y EVALUACIÓN DE ALGORITMOS EVOLUTIVOS MULTIOBJETIVOS EN OPTIMIZACIÓN Y MODELACIÓN DIFUSA. Universidad de Murcia; 2002.

[9] Juan Carlos Montoya M, Yezid Enrique Donoso M, Ramón Fabregat G, Edwing Montoya M, Diego Echeverria S. Optimización multiobjetivo para enrutamiento multicast en overlay network utilizando algoritmos evolutivos. 2008 junio;4(7):87–111.

[10] Zitzler E, Deb K, Thiele L. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. 2000;.

[11] Deb K, Agrawal S, Pratap A, Meyarivan T. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In: Schoenauer M, Deb K, Rudolph G, Yao X, Lutton E, Merelo JJ, et al., editors. Parallel Problem Solving from Nature – PPSN VI. Berlin: Springer; 2000. p. 849–858.

[12] Zitzler E, Laumanns M, Thiele L. SPEA2: IMPROVING THE STRENGTH PARETO EVOLUTIONARY ALGORITHM FOR MULTIOBJETIVE OPTIMIZATION. CIMNE; 2002. .

[13] Corne DW, Jerram NR, Knowles JD, Martin J O. PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization. 2004;.

[14] Becerra GF. Algoritmos Secuenciales y Paralelos para la Resolución del Problema Inverso de Valores Singulares. Universidad Politécnica de Valencia; 2006.

[15] Fernández Carazo A, Gómez Núñez T, Molina Luque J, García Hernández-Díaz A, Guerrero Casas FM, Caballero Fernández R. Selección de cartera de proyectos: Formalización de un modelo genérico. XV Jornadas de ASEPUMA y III Encuentro Internacional. 2007;.

[16] Troya JM, Aldana JF. Extending an Object Oriented Concurrent Logic Language for Neural Network Simulations. Springer-Verlag; 1991.

[17] Lee Altenberg. The Schema Theorem and Price's Theorem;.

[18] Goldberg DE. Sizing populations for serial and parallel genetics algorithms. (ed) Proceedings of the Third International Conference on Genetics Algorithm JDS, San Mateo CMK, editors; 1989.

[19] Schaffer JD. A study of control parameters affecting online performance of genetic algorithms for funtion optimization; 1989.

[20] Baluja S. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. 1994;.

[21] Harik GR, Lobo FG, Goldberg DE. The compact genetic algorithm. IEEE Transactions on Evolutionary Computation 1999. 1999;.

[22] Mühlenbein H, Paaß G. From recombination of genes to the estimation of distributions. Binary parameters. Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV. 1996;.

[23] Sebag M, Ducoulombier A. Extending population-based incremental learning to continuous search spaces. Parallel Problem Solving from Nature. 1998;p. 418–427.

[24] Bonet JD, Isbell C, Viola P. MIMIC: Finding optima by estimating probability densities. In Advances in Neural Information Processing Systems. 1997;9.

[25] Pelikan M, Mühlenbein H. The bivariate marginal distribution algorithm. In Advances in Soft Computing – Engineering Design and Manufacturing. 1999;p. 521–535.

[26] Baluja S, Davies S. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Proceedings of the 14th International Conference on Machine Learning. 1997;p. 30–38.

[27] Santana R, de León EP, Ochoa A. The edge incident model. Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99). 1999;p. 352–359.

[28] Larrañaga P. A Review on Estimation of Distribution Algorithms;.

[29] Davis L. Handbook of Genetic Algorithm.; 1991.

[30] Rosenberg RS. Simulation of genetic population with biochemical properties. University of Michigan. Michigan; 1967.

[31] Vidal AM. Presente y futuro de la Computación Paralela. 2000;.

[32] Kumar V, Gramar A, Grupta A, Kerypis G. Introduction to Parallel Computing: Desing and analysis of Algorithms. CA: Redwood City; 1994.

[33] Back T. Parallel Optimization of Evolutionary Algorithms;.

[34] Cantú-Paz E. A Survey of Parallel Genetic Algorithms;.

[35] Belding TC. The Distributed Genetic Algorithm Revisited. Eshe95. 1995;.

[36] Goldberg DE. Critical Deme Size for Serial and Parallel Genetic Algorithms. IlliGAL Report No. 95002; 1995.

[37] Mühlenbein H, Schomisch M. The Parallel Genetic Algorithm as Function Optimizer. Parallel Computing 17. 1991;.

[38] Spiessens P, Manderick B. A Massively Parallel Genetic Algorithm. BB91. 1991;.

[39] Hammarling S, Dongarra J, Du Croz J, J HR. An extended set of fortran basic linear algebra subroutines. ACM Trans Mat Software. 1988;.

[40] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J, Du Croz J, et al. LAPACK User Guide. SIAM. 1995;.

[41] Mark Allen Weiss. Data Structures and Algorithm Analysis in C;.

[42] Web pages for MPI and MPE;. Available from: `http://www.mcs.anl.gov/research/projects/mpi/www/`.

[43] Natterer F. The mathematics of computerized tomography. Teubner. 1986;p. 33.

[44] P N. Svd methods applied to wire antenna;p. 33.

[45] M C; CNR. Inverse Eigenvalue Problems: Theory and Applications. A Series of Lectures to be Presented at IRMA. 2001 June 27;.

[46] Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical Recipes in C. The Art of Scientific Computing. 1992;.

Glosario de términos

EDA Algoritmos de estimación de distribución de la población

AE Algoritmos evolutivos

AG Algoritmos genéticos

GCC GNU Compiler Collection

IDE Entorno integrado de desarrollo

KDE Entorno de escritorio para GNU y Unix

cluster Conjunto de PC's conectados por una red y con determinadas características

ABB Árbol de búsqueda binaria