SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Ciudad Madero

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"POR MI PATRIA Y POR MI BIEN"

ALGORITMO HÍBRIDO PARALELO PARA LA SOLUCIÓN DEL PROBLEMA DE CARTERA DE PROYECTOS MEDIANTE LA ASIGNACIÓN DE RECURSOS

Opción

Tesis

Que para obtener el grado de

Master en Ciencias de la Computación

Presenta

Ing. Federico Gamboa Ruvalcaba

G10070519

Asesor

Dra. Claudia Guadalupe Gómez Santillán

Cd. Madero Tamaulipas Enero 2018

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Ciudad Madero

"Año del Centenario de la Promulgación de la Constitución Política de los Estados Unidos Mexicanos"

Cd. Madero, Tams., a 16 de Enero de 2018

OFICIO No.: U5.013/18
ÁREA: DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS.

C. ING. FEDERICO GAMBOA RUVALCABA
No. DE CONTROL G10070519
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su Examen de Grado de Maestro en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ALGORITMO HÍBRIDO PARALELO PARA LA SOLUCIÓN DEL PROBLEMA DE CARTERA DE PROYECTOS MEDIANTE LA ASIGNACIÓN DE RECURSOS "

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTE :	DRA.	LAURA CRUZ REYES
SECRETARIO:	DR.	NELSON RANGEL VALDEZ
VOCAL:	DRA.	CLAUDIA GUADALUPE GÓMEZ SANTILLAN
SUPLENTE:	DR.	JUAN JAVIER GONZÁLEZ BARBOSA
DIRECTORA DE TESIS :	DRA.	CLAUDIA GUADALUPE GÓMEZ SANTILLAN
CO-DIRECTOR DE TESIS:	DR.	NELSON RANGEL VALDEZ

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"POR MI PATRIA Y POR MI BIEN"®

DRA. ADRIANA ISABEL REYES DE LA TORRE
JEFA DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p.- Archivo
Minuta
AIRT 'RAPR 'mdcoa*

Agradecimientos

Le doy gracias a Dios por permitirme llegar a este momento tan trascendente de mi vida y agradecerle su presencia a lo largo de esta etapa y a mis padres por tener su apoyo incondicional.

Deseo expresar un especial agradecimiento a la directora de esta tesis de Maestría, Dra. Claudia Guadalupe Gómez Santillán, por la dedicación y apoyo brindado a este trabajo, por guiarme de forma rigurosa y por ser parte en mi formación como investigador y profesional del área de las Ciencias Computacionales.

Por su orientación y disposición para la culminación de este trabajo, mi sincero agradecimiento al Co-Director de esta tesis de Maestría, Dr. Nelson Rangel Valdez quien atendió cada una de mis consultas sobre el desarrollo del proyecto.

También, gracias a la Dra. Laura Cruz Reyes y al Dr. Juan Javier González Barbosa por sus valiosas sugerencias para este proyecto.

Agradezco a las instituciones que dieron todas las facilidades para que este trabajo se llevara a cabo: el Consejo Nacional de Ciencia y Tecnología (Conacyt), la Dirección General de Educación Superior Tecnológica (DGEST) y el Instituto Tecnológico de Ciudad Madero, por haberme permitido realizar mis estudios.

Gracias a todos mis compañeros y amigos de la maestría, por haberme dado la oportunidad de conocerlos y compartir nuevas experiencias.

Contenido

Capítulo 1: Introducción

Una organización siempre trata de obtener el mayor aprovechamiento de sus recursos para garantizar su progreso y permanencia en el mercado, la correcta manipulación de los mismos resulta uno de los aspectos más importantes en el contexto de la toma de decisiones. Una adecuada selección de proyectos es fundamental, ya que una mala decisión en la selección de los proyectos impactaría fuertemente en la competitividad de la organización.

El problema de selección de cartera de proyectos es un problema de optimización combinatoria que consiste en asignar los recursos de una organización para llevar a cabo un conjunto de proyectos que conllevan un beneficio a la organización, dicho problema está catalogado como NP-Duro [1]; esto quiere decir, que no se conoce hasta este momento un algoritmo determinista que permita obtener una solución exacta en tiempo polinomial.

Una alternativa para dar solución a problemas NP-Duros es usar métodos aproximados, también llamados métodos heurísticos, que permiten obtener buenas soluciones en tiempos razonables [2]. Los métodos heurísticos pueden ser clasificados en algoritmos de búsqueda local, que generalmente trabajan con una sola solución, y algoritmos evolutivos los cuales trabajan con múltiples soluciones, a esta agrupación de soluciones se le conoce como población que naturalmente puede ser tratada de forma paralela [3], con lo que se puede reducir el tiempo de ejecución del algoritmo.

En este trabajo de investigación se propone la solución del problema de selección de cartera de proyectos, mediante el desarrollo de un algoritmo híbrido que incluya la asignación de recursos como estrategia de solución. Dicha estrategia hará uso de un algoritmo evolutivo, la administración de las tareas mediante el procesamiento paralelo, configuración de parámetros, entre otras; para lograr el objetivo establecido a través de mecanismos que buscan mejorar las estrategias actuales.

1.1 Antecedentes

Desde sus inicios, el problema de selección de cartera de proyectos ha sido estudiado como un problema de decisión, en donde el objetivo es brindar al tomador de decisiones (DM, por sus siglas en inglés) un conjunto de soluciones que den un mejor compromiso a sus preferencias; los primeros trabajos en esta área daban como resultado un conjunto de soluciones con un cardinal muy grande, lo que dificultaba al DM la elección del mejor compromiso o de un compromiso satisfactorio.

La magnitud del problema descarta el uso de métodos exactos de una manera práctica, por ello, se opta comúnmente por el uso de técnicas metaheurísticas. Debido a esto se dio inicio la red académica de Optimización y Apoyo a la Decisión (OAD).

Dicha red está formada por cuatro centros de investigación: Universidad Autónoma de Nuevo León (UANL), Universidad Autónoma de Sinaloa (UAS), Universidad de Occidente (UDO) y el Instituto Tecnológico de Ciudad Madero (ITCM).

La red tiene como finalidad explorar las posibilidades de colaboración conjunta en torno al problema de selección de selección de cartera de proyectos, además de propiciar un acercamiento fructífero entre los miembros de las universidades [4].

Dentro de los trabajos realizados con anterioridad por el grupo de trabajo se destaca el desarrollo de un algoritmo de procesamiento celular realizado por Cerecedo [4] el cual simula paralelismo para la solución del problema de selección de cartera de proyectos.

1.2 Objetivo General

Resolver el problema de selección de cartera de proyectos mediante estrategias de asignación de recursos, hibridación de algoritmos, estrategias de paralelización y estrategias basadas en patrones de descomposición para algoritmos paralelos.

1.3 Objetivos Particulares

- Revisar el estado del arte para identificar trabajos que resuelvan el problema de selección de cartera de proyectos usando estrategias de asignación de recursos.
- Analizar y diseñar estrategias basadas en patrones de descomposición para algoritmos paralelos.
- Diseñar el algoritmo híbrido, evolutivo y paralelo que resuelva el problema de selección de cartera de proyectos.
- Buscar instancias del estado del arte para llevar a cabo la experimentación.
- Evaluar el algoritmo híbrido y buscar algoritmos del estado del arte para la validación de resultados.
- Publicar los resultados mediante la escritura de artículos.

1.4 Justificación

En la realidad, muchas organizaciones renuncian a encontrar la “mejor” cartera, ya que esto es difícil de lograr. Tradicionalmente se han aplicado procedimientos y heurísticas que resultan en “buenas” carteras, pero que muy probablemente desde la óptica de la optimización global están lejos de ser la solución óptima incluso en problemas pequeños. Dichas prácticas son tradicionales en la cultura organizacional que, en parte por costumbre y en parte por simplicidad, continúan vigentes a pesar de su clara incapacidad para resolver estos problemas de manera eficaz.

Se pueden encontrar en la literatura estudios que han abordado esta problemática. Estos sugieren procedimientos de optimización que ofrecen una mejor precisión para la solución del problema de selección de cartera de proyectos. Otro factor importante al estudiar el problema de selección es la cantidad de información que se tiene que procesar, para obtener resultados de calidad en tiempos relativamente cortos.

Es por eso que en este trabajo de investigación se propone el desarrollo de un algoritmo híbrido que incluya la modelación del problema a través del problema de la asignación de recursos como estrategia de solución, además de hacer uso de un algoritmo evolutivo que explore rápidamente grandes espacios de soluciones del problema, y finalmente la administración de las tareas mediante el procesamiento paralelo para que el algoritmo sea ejecutado en diferentes nodos de procesamiento.

1.5 Alcances

Se hará uso de alguna de las estrategias de paralelización como lo es el procesamiento con memoria compartida o procesamiento con memoria distribuida.

Se modelará el problema de cartera como un problema de asignación de recursos.

Se validarán los resultados comparándolos con Cplex.

1.6 Limitaciones

Se hará uso de alguno de los algoritmos del grupo de trabajo como base.

Se hará uso de algoritmos evolutivos para la solución del problema.

Se adaptarán instancias del grupo de trabajo.

Se modelará el problema como mono-objetivo

1.7 Descripción de los capítulos

En el Capítulo 2 se enfoca en el estudio de temas o estrategias que apoyan a la metodología de solución, enseguida en el Capítulo 3 se hará una revisión al Estado del Arte, continuando con en el Capítulo 4 en el cual se hará una propuesta de solución al Problema de Selección de Cartera de Proyectos Mono-Objetivo y finalmente en el Capítulo 5 se mostraran los resultados obtenidos por el algoritmo metaheurístico.

Capítulo 2: Marco Teórico

En esta sección se describen los elementos necesarios para el desarrollo de las temáticas relacionadas con el proyecto de investigación, dentro de esas temáticas se pueden encontrar los problemas de optimización los cuales pueden ser resueltos mediante algoritmos metaheurísticos, algoritmos híbridos paralelos, asignación de recursos, todo lo anterior para resolver el problema de selección de cartera de proyectos, entre otros.

2.1 Problemas de Optimización

En matemáticas la optimización o programación matemática intenta dar respuesta a un tipo general de problemas donde se desea elegir el mejor entre un conjunto de elementos [5]. La optimización combinatoria es una rama de la optimización en matemáticas aplicadas y en ciencias de la computación, relacionada a la investigación de operaciones, teoría de algoritmos y teoría de la complejidad computacional. También está relacionada con otros campos, como la inteligencia artificial e ingeniería de software [6].

Los algoritmos de optimización combinatoria resuelven instancias de problemas que se creen ser difíciles en general, explorando el espacio de soluciones (usualmente grande) para estas instancias. Los algoritmos de optimización combinatoria logran esto reduciendo el tamaño efectivo del espacio, y explorando el espacio de búsqueda eficientemente.

Mediante el estudio de la teoría de la complejidad computacional es posible comprender la importancia de la optimización combinatoria. Los algoritmos de optimización combinatoria se relacionan comúnmente con problemas NP-hard. Dichos problemas en general no son resueltos eficientemente, sin embargo, varias aproximaciones de la teoría de la complejidad sugieren que ciertas instancias (ej. "pequeñas" instancias) de estos problemas pueden ser resueltos eficientemente. Dichas instancias a menudo tienen ramificaciones prácticas muy importantes. En este proyecto de investigación se trabajará con dos problemas NP-hard como son: el problema de selección de cartera de proyectos [7] y el problema de asignación de recursos [8].

2.1.1 Problema de la Selección de Cartera de Proyectos

La selección de una cartera de proyectos es un problema de programación lineal entera [9] de optimización multi-objetivo [10]. Una representación general del problema es la siguiente:

Cerecedo en [4] describe el problema de selección de cartera de proyectos de la siguiente forma. Suponga una cartera de n proyectos $\vec{x} = (x_1, x_2, \dots x_n)$ $donde$ $\vec{x} \in \{0,1\}^n$ para el cual:

$$x_i = \begin{cases} 1 & Si\ x_i \in \vec{x} \\ 0 & Si\ x_i \notin \vec{x} \end{cases}$$

Además considérese un presupuesto B y un vector de costos $c = (c_1, c_2, \dots c_n)$ donde $c_i \in \mathbb{N}$ representa el costo de seleccionar el proyecto i en la cartera. Asimismo un par de vectores $a = (a_{11}, a_{12}, \dots, a_{1m}, a_{21}, \dots, a_{nm})$ y $r = (r_{11}, r_{12}, \dots, r_{1l}, r_{21}, \dots, r_{nl})$ donde $a_{ij} \in \{0,1\}$ representa si el proyecto i pertenece al área j y $r_{ik} \in \{0,1\}$ representa si el proyecto i pertenece a la región k.

Entonces el problema se resuelve al encontrar una cartera $\vec{x}$ que satisfaga las siguientes restricciones mostradas en las Ecuaciones **¡Error! No se encuentra el origen de la referencia.**, (2) y (3), las cuales están relacionadas con las áreas y las regiones:

$$\sum_{i=0}^{n} x_i c_i \leq B \tag{1}$$

$$A_1^L \leq \sum_{i=0}^{n} x_i c_i a_{i1} \leq A_1^U \qquad R_1^L \leq \sum_{i=0}^{n} x_i c_i r_{i1} \leq R_1^U$$

$$A_2^L \leq \sum_{i=0}^{n} x_i c_i a_{i2} \leq A_2^U \qquad R_2^L \leq \sum_{i=0}^{n} x_i c_i r_{i2} \leq R_2^U$$

$$\vdots \qquad\qquad \vdots$$

$$A_m^L \leq \sum_{i=0}^{n} x_i c_i a_{im} \leq A_m^U \tag{2} \qquad R_k^L \leq \sum_{i=0}^{n} x_i c_i r_{ik} \leq R_k^U \tag{3}$$

Dónde:

$$A_j^U = Límite\ superior\ de\ inversión\ en\ área\ j$$

$$A_j^L = Límite\ inferior\ de\ inversión\ en\ área\ j$$

$$R_j^U = Límite\ superior\ de\ inversión\ en\ región\ j$$

$$R_j^L = Límite\ inferior\ de\ inversión\ en\ región\ j$$

Asimismo, incluye el vector $b = (b_{11}, b_{12}, \dots, b_{1o}, b_{21}, \dots, b_{no})$ donde b_{ij} indica el beneficio aportado en el objetivo j por el proyecto i. Entonces, la función objetivo mostrada en las Ecuaciones (4) y (5) se define como:

$$f(x) = (f_1(\vec{x}), f_2(\vec{x}), \dots, f_p(\vec{x})) \tag{4}$$

Donde

$$f_i(x) = \sum_{j=0}^{n} x_j b_{ji} \tag{5}$$

Cabe señalar que si el valor de j solo es 1 el problema solo tendrá un solo objetivo, si j toma valores entre 2 y *N* se considera que el problema es multiobjetivo. En este trabajo se trabajó con el valor de *j* = 1.

La mejor solución se define al resolver $max_{x \in R_F}(f(\vec{x}))$ donde R_f es el espacio de soluciones factibles. La solución del problema nos brinda un conjunto de soluciones consideradas óptimas, este es un conjunto de soluciones que crece exponencialmente con la dimensionalidad del problema.

Ejemplo

Una empresa desea decidir entre un conjunto de proyectos los que mayor beneficio o impacto podrían retribuirle, esto contemplando un conjunto de restricciones como lo es el presupuesto total y un rango mínimo y máximo por cada área y región de la empresa. La empresa tiene un presupuesto total de 80,000 unidades que será distribuido entre 25 proyectos, cada proyecto pertenece a una de las tres áreas y a una de las dos regiones con las que cuenta el problema, cada una de las áreas cuenta con un presupuesto mínimo de 14,810 y máximo de 47,995 unidades mientras que cada una de las regiones tienen un presupuesto mínimo de 23,525 y un máximo de 68,000 unidades.

La información de entrada del problema es una lista de proyectos que cuenta con su costo requerido, su beneficio, el área al que pertenece y su región.

Proyecto	Costo	Beneficio	Área	Región
1	9695	7960	1	1
2	8635	8860	2	1
3	6140	3990	3	1
4	9430	4070	1	2
5	6310	6000	1	2
6	8600	4835	3	1
7	9360	9415	1	1
8	7080	9530	2	1
9	8715	6530	1	2
10	8700	5445	3	1
11	7490	1305	1	1
12	5490	5480	1	2
13	5450	3825	1	2
14	9585	1225	2	2
15	5550	3515	3	1
16	9875	9085	2	1
17	9925	9675	2	1
18	6780	5835	2	1
19	9970	7305	1	1
20	5205	1675	1	2
21	5095	4345	3	2

22	6895	2790	1	1
23	5975	7920	3	1
24	7940	6220	3	1
25	7590	9720	2	2

Ahora se ejemplifica el problema mediante una solución aleatoria generada con los proyectos de la instancia previamente mostrada.

Las soluciones del problema tienen una representación binaria donde un "1" en la i-*ésima* posición de la solución significa que el proyecto que ocupa esa posición recibe apoyo al formar parte de la cartera y un "0" significa lo contrario.

Un ejemplo de cartera factible es la siguiente:

Proyecto	Costo	Beneficio	Área	Región
2	8,635	8,860	2	1
5	6,310	6,000	1	2
7	9,360	9,415	1	1
8	7,080	9,530	2	1
12	5,490	5,480	1	2
13	5,450	3,825	1	2
17	9,925	9,675	2	1
21	5,095	4,345	3	2
23	5,975	7,920	3	1
24	7,940	6,220	3	1
25	7,590	9,720	2	2
	78,850	80,990		

Teniendo un costo total de 78,850 y un beneficio de 80,990 unidades al seleccionar los proyectos enumerados, como se puede apreciar, la solución no excede el presupuesto máximo de 80,000 unidades.

También podemos observar que cada una de las áreas y regiones tienen un presupuesto entre el mínimo y máximo preestablecido, el cual es de un mínimo de 14,810 y un máximo de 47,995 unidades para cada una de las áreas y un mínimo de 23,525 y un máximo 68,000 unidades para cada una de las regiones.

Áreas			
#	Proyectos	Costo	Beneficio
1	5,7,12,13	26,610	24,720
2	2,8,17,25	33,230	37,785
3	21,23,24	19,010	18,485

Regiones			
#	Proyectos	Costo	Beneficio

1	2,7,8,17,23,24	48,915	51,620
2	5,12,13,21,25	29,935	29,370

En breve se hará revisión del problema de asignación de recursos, analizando sus características y aplicaciones del problema.

2.1.2 Problema de Asignación de Recursos

El problema de asignación de recursos (RAP), busca encontrar una asignación óptima de una cantidad limitada de recursos a un número de tareas para optimizar sus objetivos sujetos a las restricciones dadas. Un recurso puede ser una persona, un activo, material, o capital que puede ser usado para conseguir un resultado [11].

El RAP tiene una amplia variedad de aplicaciones, incluyendo Asignación de productos [12], Cartera de Proyectos [13], Pruebas de software [14], Asignación de recursos en la atención médica [15], Proceso de asignación o calendarización de talleres de trabajo [16], entre otras aplicaciones.

Existen diversos modelos para resolver el problema de asignación de recursos, entre los que destaca el propuesto por Gamrath [17] para seleccionar las tareas a ejecutarse en recurso compartido. Considere un conjunto I de tareas y una capacidad de recurso C. Cada elemento $i \in I$ tiene un beneficio p_i, una cantidad de recurso utilizado w_i, un periodo de inicio s_i, y un periodo de inicio e_i. El horizonte de tiempo considerado es el tiempo de inicio más temprano al último tiempo de finalización de las tareas. Se asocia para cada tarea una variable binaria x_i, que, si tiene el valor 1, indica que la tarea se ejecuta desde su tiempo de inicio hasta antes de su tiempo de finalización.

Una tarea consume recursos mientras se ejecuta. La meta es seleccionar las tareas que puedan ejecutarse en orden para maximizar el beneficio mientras no excedan la capacidad de los recursos compartidos. Sea $S = \{s_i | i \in I\}$ definido por el conjunto del tiempo de inicio de todas las tareas, y sea $L_s = \{i \in I | s_i \leq s < e_i\}$ definido por el conjunto de las tareas que son ejecutadas en cada tiempo de inicio $s \in S$. El problema puede ser modelado como se muestra:

$$maximizar \sum_{i \in I} p_i x_i$$

$$sujeto\ a \sum_{i \in L_s} w_i x_i \leq C$$

$$s \in S\ (capacidad)$$

$$x_i \in \{0,1\}$$

$$i \in I$$

Ejemplo

En la formulación, las restricciones (de capacidad) se asegura de que las tareas ejecutándose no exceden la capacidad del recurso. Para ilustrar, considere las siguientes 5 tareas de ejemplo que cuentan con un beneficio p=(6,8,5,9,8), un costo w=(8,5,3,4,3), un tiempo de inicio s=(1,3,5,7,8), un tiempo de finalización e=(5,8,9,17,10), y un presupuesto C=10. La formulación produce una matriz de restricciones que tiene una "estructura de escalera" que es determinada por las tareas ejecutándose.

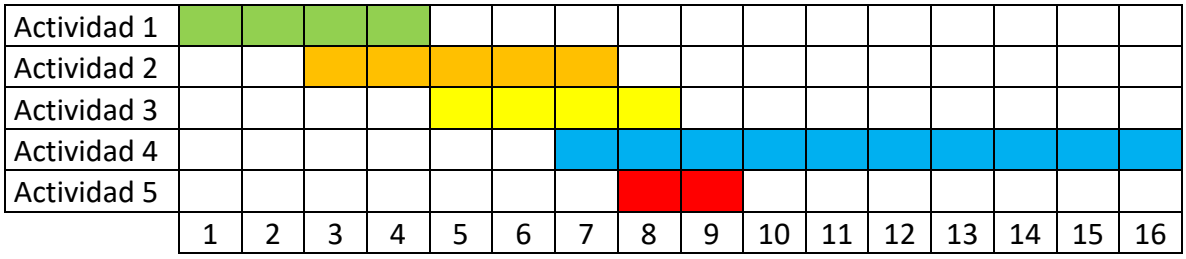


Figura 8. Expansión del problema de Asignación de recursos

Como podemos ver en la Figura 8, las tareas pueden llegar a coexistir entre sí, por lo que tienen que competir por el uso del recurso disponible teniendo que decidir entre cuál de ellas elegir. Con los datos del problema se puede formular un modelo matemático, como el que se muestra a continuación.

$$
\begin{array}{lrcrcrcrcrcr}
\text{Maximizar} & 6x_1 & + & 8x_2 & + & 5x_3 & + & 9x_4 & + & 8x_5 & & \\
\text{Sujeto a} & 8x_1 & & & & & & & & & \leq & 10 \\
 & 8x_1 & + & 5x_2 & & & & & & & \leq & 10 \\
 & & & 5x_2 & + & 3x_3 & & & & & \leq & 10 \\
 & & & 5x_2 & + & 3x_3 & + & 4x_4 & & & \leq & 10 \\
 & & & & & 3x_3 & + & 4x_4 & + & 3x_5 & \leq & 10
\end{array}
$$

$$x_i \in \{0,1\} \quad i \in I$$

Para conocer que tareas elegir se debe dar solución al modelo matemático, siendo una solución factible al problema la elección de las tareas 1, 3, 4 y 5 que dan un beneficio de 28 unidades.

En la siguiente sección se propondrá un modelo matemático que integra características del problema de Selección de Cartera de Proyectos y Asignación de Recursos.

2.1.3 Integración de los Modelos Propuestos

Se integraron los modelos de referencia del Problema de Selección de Cartera de Proyectos y del Problema de Asignación de Recursos dando como resultado el modelo que se presentará a continuación.

Variables:

$x_{i,t}$ = Matriz binaria que representa si el proyecto i es financiado (1) o no (0) en el tiempo t.

$P_{a,t}$ =Presupuesto requerido para el área a en el año t.

$P_{r,t}$ =Presupuesto requerido para la región r en el año t.

Constantes:

n = Número de proyectos.

O = Número de objetivos.

T = Número de años a calcular.

na = Número de áreas.

nr = Número de regiones.

i = Índice para los proyectos en donde $i \in \{1,2,\dots,n\}$.

a = Índice de áreas en donde, $a \in \{1,2,\dots,na\}$.

r = Índice de áreas en donde, $r \in \{1,2,\dots,nr\}$.

o = Índices para los objetivos en donde, $o \in \{1,2,\dots,O\}$.

t = Índices que representa el año o periodo sobre el que se están haciendo los cálculos en donde, $t \in \{1,2,\dots,T\}$.

$A_{i,a}$ = Matriz binaria que indica si el proyecto i pertenece al área a.

$G_{i,r}$ = Matriz binaria que indica si el proyecto i pertenece a la región r.

P_t = Presupuesto anual para el año t.

$P_{a_{min},t}$ = Presupuesto mínimo para el área a en el año t.

$P_{a_{max},t}$ = Presupuesto máximo para el área a en el año t.

$P_{r_{min},t}$ = Presupuesto mínimo para la región r en el año t.

$P_{r_{max},t}$ = Presupuesto máximo para la región r en el año t.

$b_{o,t}^{i}$ = Beneficio del proyecto i, al objetivo j en el tiempo t.

$c_{i,t}$ = Matriz que contiene los costos de cada proyecto i en el tiempo t.

Función objetivo:

$$\max\{z(x)\} \tag{6}$$

En donde:

$$z(x) = < z_1(x), z_2(x), \dots, z_n(x) > \tag{7}$$

$$z_i(x) = \sum_{o=1}^{O} \sum_{t=1}^{T} b_{o,t}^{i} x_{i,t} \tag{8}$$

Restricciones:

$$\left(\sum_{i=1}^{n} x_{i,t}\, c_{i,t}\right) \leq P_t \quad \forall_t \tag{9}$$

$$P_{a_{min},t} \leq P_{a,t} \leq P_{a_{max},t} \quad \forall_{a,t} \tag{10}$$

$$P_{r_{min},t} \leq P_{r,t} \leq P_{r_{max},t} \quad \forall_{r,t} \tag{11}$$

$$P_{a,t} = \sum_{i=1}^{n} x_{i,t}\, c_{i,t}\, A_{i,a} \quad \forall_{l,t} \tag{12}$$

$$P_{r,t} = \sum_{i=1}^{n} x_{i,t}\, c_{i,t} G_{i,r} \quad \forall_{r,t} \tag{13}$$

En las ecuaciones (6), (7), (8) se puede observar la función objetivo del problema que intenta maximizar el beneficio de un proyecto i en un tiempo t con respecto a sus O objetivos.

En la ecuación (9) se limita el número de proyectos financiados en el año t acorde al presupuesto de ese año, en la ecuación (10) se restringe el presupuesto del área a en el año t con respecto a su presupuesto mínimo y máximo para ese año, la misma dinámica se sigue en la ecuación (11) para las regiones mientras que en las ecuaciones (12) y (13) se calcula el presupuesto para cada área y región en cada uno de los años.

2.2 Técnicas de solución a problemas de optimización

El término *optimización*, según [10], es la acción de encontrar una o más soluciones factibles que corresponden a valores extremos de uno o más objetivos. Cuando un problema de optimización involucra solamente una función objetivo, se le llama *optimización mono-objetivo* a la tarea de encontrar la solución óptima. En el caso donde el problema de optimización involucra más de una función objetivo, se le conoce como *optimización multi-objetivo* a la tarea de encontrar una o más soluciones óptimas.

2.3 Optimización estocástica

La programación estocástica consiste en optimizar un problema con parámetros inciertos que tienen o no una distribución de probabilidad conocida. Entre las técnicas más utilizadas y de más estudio se encuentra la programación estocástica bi-etapa (Two-stage stochastic problem) la cual fue propuesta por [18].

2.3.1 Métodos exactos

Los métodos exactos buscan la mejor solución a un problema determinado, con este propósito estos métodos navegan por todo el espacio de soluciones factibles; son considerados imprácticos en problemas de gran tamaño dado los altos tiempos de cómputo necesarios para su terminación. Entre los métodos exactos se puede nombrar métodos de programación lineal entera, Branch and Bound e incluso búsquedas por fuerza bruta.

2.3.2 Métodos heurísticos

Para la mayoría de problemas de interés no se conoce un algoritmo exacto con complejidad polinómica que encuentre la solución óptima a dicho problema. Además, la cardinalidad del espacio de búsqueda suele ser enorme, lo cual hace generalmente inviable el uso de algoritmos exactos, fundamentalmente porque la cantidad de tiempo que se necesitaría para encontrar una solución es completamente inaceptable. Debido a estos dos motivos, se deben utilizar métodos aproximados o heurísticas que permitan obtener una solución de calidad en un tiempo razonable a estos problemas.

El término heurística se debe al matemático G. Polya quien lo empleó por primera vez en su libro *How to solve it* [19]. Con este término Polya quería expresar las reglas con las que los humanos gestionan el conocimiento común. Actualmente, existen bastantes definiciones para el término heurística. Una de las más claras e intuitivas es la presentada por Zanakis et al. [20]:

"Procedimientos simples, a menudo basados en el sentido común, que se supone que obtendrán una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido".

Los métodos heurísticos tienen su principal limitación en su incapacidad para escapar de óptimos locales. Esto se debe, fundamentalmente, a que estos algoritmos no utilizan ningún mecanismo que les permita proseguir la búsqueda del óptimo en el caso de quedar atrapados en un óptimo local. Para solventar este problema, se introducen otros algoritmos de búsqueda más *inteligentes* (denominados metaheurísticas [21]) que evitan, en la medida de lo posible, este problema. Este tipo de algoritmos son procedimientos de alto nivel que guían a métodos heurísticos conocidos, evitando que éstos queden atrapados en óptimos locales.

Clasificación de Métodos Heurísticos

Existen métodos heurísticos (también llamados algoritmos aproximados, procedimientos inexactos, algoritmos basados en el conocimiento o simplemente heurísticas) de diversa naturaleza, por lo que su clasificación es bastante complicada. Duarte [22] sugiere la siguiente clasificación.

1. **Métodos constructivos:** Procedimientos que son capaces de construir una solución a un problema dado. La forma de construir la solución depende fuertemente de la *estrategia* seguida. Las estrategias más comunes son:

- *Estrategia voraz:* Partiendo de una semilla, se va construyendo paso a paso una solución factible.
- *Estrategia de descomposición*: Se divide el problema en subproblemas de menor tamaño.
- *Métodos de reducción*: Identifican características de las mejores soluciones conocidas y se asume que la solución óptima también las tendrá.
- *Métodos de manipulación del modelo:* Consisten en simplificar el modelo del problema original para obtener una solución al problema simplificado. Entre estos métodos se pueden destacar: la linealización, la agrupación de variables, introducción de nuevas restricciones, etc.

2. **Métodos de búsqueda:** Parten de una solución dada que intentan mejorar de manera iterativa. Se pueden clasificar en:
   - Estrategia aleatorizada: Toma de manera aleatoria cualquier solución dentro de su vecindario actualizando la solución original por el vecino elegido favoreciendo la diversidad de las soluciones exploradas.
   - Best Improvement: Examina cada uno de los vecinos de la solución original y la actualiza si encuentra una solución mejor, esta búsqueda consigue buenas soluciones en un número reducido de iteraciones comparado con la estrategia aleatorizada pero siendo propensa a caer en óptimos locales.
   - First Improvement: Realiza una búsqueda dentro del vecindario eligiendo el primer vecino que sea mejor que la solución actual, esto no garantiza encontrar el mejor vecino, sin embargo, permite diversificar las soluciones de la búsqueda teniendo un balance entre exploración y explotación.

En general, los métodos heurísticos tienen problemas cayendo en óptimos locales, por lo que surgió la necesidad de desarrollar métodos más robustos naciendo las metaheurísticas.

2.3.2 Metaheuristicas

El científico Glover acuño el término Meta-heurística en el año de 1986, describiéndolo como “un procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimilidad local” [21]. Por otro lado una de las definiciones más completa que se han formulado es la descrita por J.P. Kelly [23], la cual nos dice que “Las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos”.

Las metaheurísticas se suelen clasificar de acuerdo al número de soluciones que maneja el algoritmo, caracterizándose las metaheurísticas trayectoriales por el manejo de una única solución mientras que las metaheurísticas poblacionales manejan un conjunto de soluciones.

Las metaheurísticas suelen ser divididas trayectoriales y poblacionales. Se describirán a continuación.

Las metaheurísticas trayectoriales es un método de búsqueda iterativo que se caracterizan por mejorar la solución actual mediante la inspección de su vecindario, donde la búsqueda finaliza cuando se alcanza un número máximo de iteraciones o se detecta un estancamiento del proceso, estas metaheurísticas parten de una solución factible, y conforme exploran el espacio de soluciones generan un camino o trayectoria, mediante operaciones de movimiento. Dentro de las principales metaheurísticas trayectoriales se encuentran la Búsqueda Tabú (TS), Recocido Simulado (SA), Búsqueda de Vecindad Variable (VNS), Búsqueda Local Guiada (GLS), Aceptación de Umbral (TA), Métodos Ruidosos (NM), FANS (Fuzzy Adaptive Neighborhood Search) y Búsqueda Local Iterada (ILS).

Por otro lado, se encuentran las metaheurísticas poblacionales, las cuales utilizan un conjunto de soluciones iniciales, a este conjunto se le denomina población la cual es mejorada iterativamente a través de un proceso de exploración inteligente del espacio de búsqueda. Algunas de las metaheurísticas más importantes dentro de esta sub-categoría se encuentran: los algoritmos evolutivos (Algoritmo Genético (GA), Algoritmo Memético (MA)), Búsqueda Dispersa, Re-encadenamientos de Trayectorias (PR), Algoritmos de Estimación de la Distribución, Algoritmos Culturales y por último se tiene el algoritmo de Inteligencia de Enjambre y Optimización por Enjambre de Partículas.

Paradigmas principales de la computación evolutiva:

1. Estrategias Evolutivas (EE): Este paradigma se inspira en la evolución que tiene un solo individuo, cual es influenciado solamente por el operador de mutación, y en una segunda forma es por la recombinación de los elementos del cromosoma que son padres.
2. Programación Evolutiva (PE): Esta técnica se inspira en el principio de la evolución a nivel de las especies. El único operador que utiliza esta técnica es el de mutación, además de utilizar la selección probabilística. La recombinación dentro de esta técnica no es permitida debido a que especies distintas no se mezclan entre si.
3. Algoritmos Genéticos (AG). Estos algoritmos inician su funcionamiento creando una población de individuos de manera aleatoria, posteriormente se les aplica los operadores de cruzamiento y mutación, obteniendo una nueva generación de la población, donde el operador genético principal es la recombinación y el operador secundario es la mutación.

El algoritmo básico es el siguiente:

1. Generar una población inicial.
2. Calcular aptitud de cada individuo.

3. Seleccionar (probabilísticamente) en base a aptitud.
4. Aplicar operadores genéticos (cruza y mutación) para generar nuevos individuos.
5. Repetir hasta que se satisfaga el criterio de paro.

Los individuos se suelen representar como una cadena binaria llamada “cromosoma”. A cada posición de la cadena se le denomina “gene” y al valor dentro de esta posición se le llama “alelo”.

Para poder aplicar el algoritmo genético se requiere de los 5 componentes básicos siguientes:

- Una representación apropiada de las soluciones potenciales del problema.
- Un mecanismo que genere una población inicial de posibles soluciones, comúnmente de manera aleatoria.
- Una función de evaluación que clasifique las soluciones acorde a su “aptitud”.
- Operadores genéticos que modifiquen la composición de los hijos que se producirán para las siguientes generaciones.
- Definir los distintos parámetros de algoritmo genético: tamaño de la población, número máximo de generaciones, probabilidad de cruza, probabilidad de mutación, etc.

Aplicaciones

Algunas aplicaciones de los Algoritmos Genéticos son las siguientes [24]:

- Optimización (estructural, de topologías, numérica, combinatoria, etc.)
- Aprendizaje de máquina (sistemas clasificadores)
- Bases de datos (optimización de consultas)
- Reconocimiento de patrones (por ejemplo, imágenes)
- Generación de gramáticas (regulares, libres de contexto, etc.)
- Planeación de movimientos de robots
- Predicción

No hay algoritmo que lo pueda resolver todo eficientemente, por ello se ha buscado combinar más de una técnica para resolver la problemática.

2.3.4 Hibridación de algoritmos metaheurísticos

Las Metaheurísticas híbridas consisten en combinar dos o más algoritmos, o diferentes metaheurísticas con la finalidad de generar nuevos métodos que aprovechen las ventajas de las estrategias individuales para obtener mejores resultados y la reducción de la complejidad del problema. Talbi [25] presenta una taxonomía de metaheurísticas híbridas y propone dos clasificaciones para este tipo de métodos: jerarquizadas y plana. Las diferentes hibridaciones de metaheurísticas pueden clasificarse de un modo jerárquico en:

- Combinación de bajo nivel: los procedimientos heurísticos están embebidos unos en otros, sustituyendo una función por otra metaheurística. Esta combinación se divide en:
  - Serie *(LRH- Low level Relay Hybrid)* un método se introduce dentro de otro como una función.
  - Paralelo *(LTH- Low level Teamwork Hybrid)* se tiene una población de soluciones de tal forma que sobre cada solución actúa un método distinto.
- Combinación de alto nivel: los métodos heurísticos se combinan están autocontenidos, de forma que no existe interacción entre ellos. Se dividen en:
  - Serie *(LRH- Low level Relay Hybrid)* se tiene una única solución de tal forma que un método se aplica después del otro.
  - Paralelo *(LTH- Low level Teamwork Hybrid)* se tiene una población de soluciones de forma que cada método se aplica independientemente a cada solución cooperando para obtener el óptimo global.

Además, las metaheurísticas hibridas pueden organizarse en una clasificación plana de la siguiente manera:

- Homogéneas o Heterogéneas:
  - Homogéneas: la metaheurística usa la misma combinación de algoritmos.
  - Heterogéneas: la metaheurística usa diferentes combinaciones de algoritmos.
- Globales o parciales:
  - Globales: en las hibridaciones globales, todos los algoritmos buscan en todo el espacio de búsqueda. El objetivo es entonces explorar el espacio más minuciosamente.
  - Parciales: en las hibridaciones parciales el problema se descompone en subproblemas, cada uno definido en su propio espacio de búsqueda dedicándose cada uno de los algoritmos a explorar cada uno de los sub-espacios. Los subproblemas están relacionados entre sí a través de restricciones entre las soluciones encontradas en cada uno. Para respetar estas restricciones los algoritmos se comunican entre ellos construyendo una solución global factible.
- Especializados o generales
  - Especializados: se combinan algoritmos resuelven diferentes problemas de optimización.
  - Generales: todos los algoritmos resuelven el mismo problema de optimización

2.3.5 Paralelización de algoritmos metaheurísticos

La paralelización de las metaheurísticas tienen como objetivo resolver problemas de mayor complejidad en menor tiempo comparado con sus versiones secuenciales [26], siendo paralelizadas con mayor frecuencia las metaheurísticas evolutivas como los algoritmos genéticos o sistemas de colonias de hormigas [26]. Para clasificar este tipo de metaheurísticas se ha propuesto clasificarlas acorde a la estrategia de paralelización [27]:

- Paralelización de operaciones dentro de una iteración del método: Obtiene los mismos resultados que su versión secuencial en menor tiempo.
- Descomposición del dominio del problema o espacio de búsqueda: Se subdivide el problema para ser resuelto en paralelo y posteriormente construir una solución global. Los resultados pueden ser diferentes a su versión secuencial.
- Caminos de búsqueda múltiples con varios grados de sincronización y cooperación. Esta estrategia intensifica la exploración del espacio de búsqueda realizando varios procesos de búsqueda en paralelo sobre el mismo espacio de búsqueda.

2.4 Estrategias de optimización

La palabra “optimizar” se refiere a la forma de mejorar alguna acción o trabajo realizado [28]. En ésta sección se hablará en particular de las formas de optimización desde el punto de vista matemático y las formas de descomponer problemas en subproblemas de menor tamaño que nos permita paralelizar o simplificar el problema que se pretende resolver.

2.4.1 Optimización lineal bietapa y multietapa

Fábián [29] comenta que los problemas de programación estocástica de dos etapas se derivan de modelos en los que se toman las decisiones en dos etapas y la observación de algún evento aleatorio tiene lugar. De ahí que la primera decisión debe ser tomada cuando no se conoce el resultado del evento aleatorio. Por ejemplo, la primera etapa puede representar la decisión sobre el diseño de un sistema; y la segunda etapa, una decisión sobre el funcionamiento del sistema bajo ciertas circunstancias. El objetivo puede ser encontrar un equilibrio entre el costo de la inversión y los costos de operación a largo plazo.

En caso de que los parámetros aleatorios no tienen una distribución finita, se aplican métodos de discretización para hacer que el problema sea tratable por solvers. En el contexto de las distribuciones discretas finitas, los posibles resultados de los eventos aleatorios son llamados escenarios.

Para cada escenario, se incluye un subproblema que describe la decisión de la segunda etapa en caso de que este escenario se realice. Estos subproblemas son llamados problemas del recurso. Los subproblemas están vinculados por las variables de la primera etapa, es decir, las variables que representan la decisión de la primera etapa.

En la formulación del Problema Lineal Estocástico de Multietapa se debe considerar una secuencia de decisiones que evolucionan en el tiempo y puede interpretarse como un encadenamiento de múltiples árboles de dos etapas [30].

El proceso de decisión es *no anticipativo* en el sentido que las decisiones tomadas en cualquier etapa del proceso no dependen de las realizaciones futuras de los parámetros aleatorios o de futuras decisiones. La información histórica o pasada y la especificación probabilística del proceso es en lo que se basa la decisión.

Existen diversas técnicas con las que se puede dar respuesta a los problemas estocásticos, entre ellas destacan las técnicas de descomposición.

2.4.2 Técnicas de descomposición

Ramos y Cerisola han estudiado las técnicas de descomposición en problemas estocásticos observando que estas técnicas hacen que el uso de procesamiento paralelo o distribuido sea muy conveniente [31] permitiendo dar solución a problemas lineales estocásticos de gran tamaño.

Las técnicas de descomposición resuelven problemas de gran tamaño generando subproblemas que son solucionados de manera iterativa. Se suelen aplicar cuando se identifican partes del problema que se pueden resolver con mayor facilidad de manera individual. Se considera una forma flexible y elegante de resolver problemas.

La manipulación del problema tiene tres objetivos [31]: inducir separación entre problemas, inducir linealidad en un problema parcialmente no lineal y aislar partes del problema para utilizar algoritmos más eficientes.

La elección de la técnica depende de las características del problema, usando una técnica generadora de columnas como la Descomposición de Lagrange cuando las restricciones dificultan la solución del problema y un método generador de filas como la Descomposición de Benders cuando las variables dificultan dar solución al problema o cuando existen dos o más conjuntos de variables de diferente naturaleza, por ejemplo, problemas de programación entera mixta.

En la siguiente sección se profundizará en la Descomposición de Lagrange, técnica que podría acoplarse dada la naturaleza del Problema de Selección de Cartera de Proyectos.

2.4.3 Descomposición de Lagrange

Este tipo de descomposición toma una matriz de restricciones con forma de escalera y la transforma en una matriz de bloque angular que pueda facilitar la división del problema. Esto lo realiza agrupando restricciones del problema y duplicando las variables que se encuentren en más de un grupo de restricciones. Por cada variable que se añade se agrega una restricción para forzar que las variables duplicadas tengan el mismo valor.

Para demostrar este método podemos tomar el ejemplo mostrado en el modelado del problema de asignación de recursos. Se cuenta con 5 restricciones que se dividirán en bloques como se ilustra.

Bloque 1	$8x_1$					$\leq$	10
	$8x_1$	+	$5x_2$			$\leq$	10
Bloque 2			$5x_2$	+	$3x_3$	$\leq$	10

$$5x_2 + 3x_3 + 4x_4 \leq 10$$

Bloque 3 $$3x_3 + 4x_4 + 3x_5 \leq 10$$

Se busca crear estos bloques con la finalidad de poder realizar la solución del problema de manera independiente favoreciendo así la paralelización del problema. Como se puede apreciar existen variables que aparecen en más de un bloque de restricciones lo que obstaculiza la solución del problema. Para ello se hace uso de la Descomposición Lagrange.

La descomposición duplica las variables que se repiten en distintos bloques de restricciones y añade una restricción por cada variable que ha sido duplicada, forzando así que cada una de las copias de la variable sean iguales entre sí. Con esto se logra que las restricciones puedan ser escritas en forma de bloque angular.

$$\begin{aligned}
x_2^1 - x_2^1 &= 0\\
x_3^2 - x_3^2 &= 0\\
x_4^2 - x_4^2 &= 0\\
8x_1^1 &\leq 10\\
8x_1^1 + 5x_2^1 &\leq 10\\
5x_2^2 + 3x_3^2 &\leq 10\\
5x_2^2 + 3x_3^2 + 4x_4^2 &\leq 10\\
+ 3x_3^3 + 4x_4^3 + 3x_5^3 &\leq 10
\end{aligned}$$

Con esto se puede dar respuesta de manera independiente a cada bloque de restricciones favoreciendo la solución en paralelo del problema. Existen diversas estrategias que permiten paralelizar algoritmos metaheurísticos que se harán mención a continuación.

2.5 Estrategias de paralelización para metaheurísticas

Los algoritmos paralelos surgen ante la necesidad de cómputo requerida por problemas de extrema complejidad, cuyo tiempo de ejecución utilizando los tradicionales algoritmos secuenciales es prohibitivo [32]. Es por eso que se buscó la manera de poder adaptar este tipo de heurísticas a distintas configuraciones de cómputo paralelo, lo que dio lugar a tres grandes estrategias para el desarrollo de metaheurísticas paralelas: algoritmos maestro-esclavo, algoritmos de grano fino y algoritmos de grano grueso.

2.5.1 Maestro-esclavo

La estrategia Maestro-Esclavo consiste en un algoritmo que contiene una sola población, en la que un nodo maestro lleva a cabo la lógica del algoritmo y los esclavos realizan la evaluación de las soluciones en paralelo [32]. Con esta estrategia se llega a reducir los tiempos de ejecución del algoritmo sin hacer cambios significativos con respecto a su versión

secuencial, por lo que se considera una manera sencilla de paralelizar un algoritmo. Este tipo de estrategia se ajusta a la programación de memoria compartida.

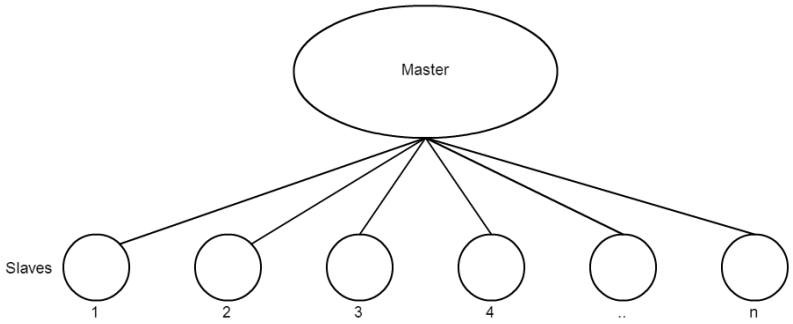


Figura 1. Arquitectura Maestro-Esclavo

2.5.2 Grano grueso

En esta estrategia se divide la población en múltiples subpoblaciones, también llamados demes, que evolucionan de manera aislada la mayor parte del tiempo intercambiando individuos de manera ocasional [33]. A este intercambio de individuos se le llama migración, y se considera como un nuevo operador.

Otro nombre con el que se le conoce a los algoritmos de grano grueso es algoritmos distribuidos, ya que suelen implementarse bajo el paradigma de programación paralela con memoria distribuida.

Se requiere relativamente de poco esfuerzo para convertir un algoritmo serial en uno de grano grueso, debido a que la mayor parte de la de la estructura del algoritmo no se ve afectada, solo es necesario definir los criterios de migración, como el número de de individuos a migrar, con que frecuencia se realizará la migración, que individuos seran migrados y cuales serán remplazados. En la Figura 2 se ilustra la arquitectura previamente expuesta.

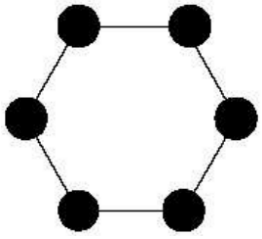


Figura 2. Arquitectura de grano grueso

2.5.3 Grano fino

Otra forma de paralelizar un algoritmo es usando una estrategia de grano fino. En este caso, la población se divide en un gran número de subpoblaciones muy pequeñas. De hecho, el caso ideal sería tener sólo un individuo por cada unidad de procesamiento disponible [33].

Debido al aumento del número de subpoblaciones en los algoritmos de grano fino, puede haber un aumento considerable en la comunicación que puede reducir significativamente la eficiencia del algoritmo. Estos algoritmos suelen modelar las subpoblaciones en forma de una malla bidimensional teniendo comunicación con los individuos aledaños. La Figura 3 ejemplifica la arquitectura previamente mencionada.

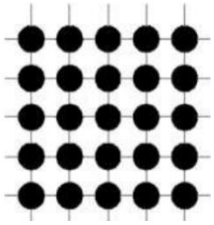


Figura 3. Arquitectura de grano fino

2.5.4 Grano grueso y fino

Consiste en usar un algoritmo de grano fino embebido dentro de un algoritmo de grano grueso. Un ejemplo de este tipo de algoritmo híbrido es el propuesto por Gruau [34], en el cual la población de cada deme se coloca en una malla bidimensional y los demes se conectan entre sí en forma de toroide bidimensional habiendo migración con frecuencia entre los demes vecinos. En la Figura 4 se muestra la arquitectura propuesta.

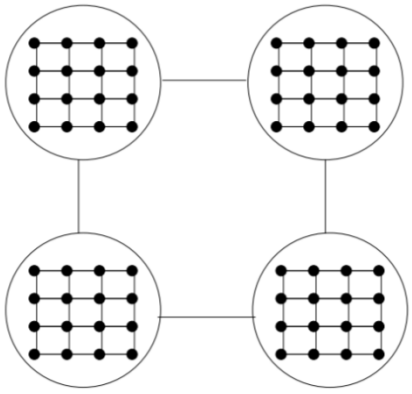


Figura 4. Arquitectura de grano grueso y grano fino.

2.5.5 Grano grueso y maestro-esclavo

Otra alternativa es la hibridación de un algoritmo maestro-esclavo dentro de un algoritmo de grano grueso en el que se divide la población en demes realizando la evaluación de los individuos en paralelo bajo el esquema maestro-esclavo y realizando migraciones

periódicamente entre los distintos demes. En la Figura 5 se aprecia un esquema de este tipo de arquitectura.

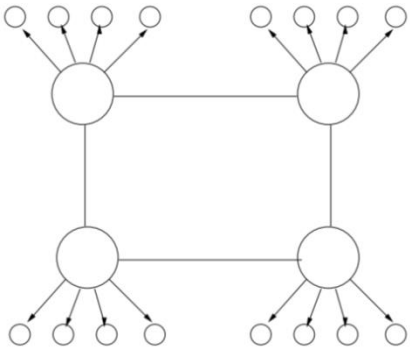


Figura 5. Arquitetura de grano grueso y maestro-esclavo.

2.5.6 Grano grueso a dos niveles

En esta estrategia se hace uso de un algoritmo de grano grueso incrustado dentro de otro algoritmo de grano grueso habiendo mayor comunicación en el algoritmo interno que en el externo. podría consistir en usar un AG de grano grueso tanto a bajo como a alto nivel [33]. En la Figura 6 se muestra la arquitectura previamente mencionada.

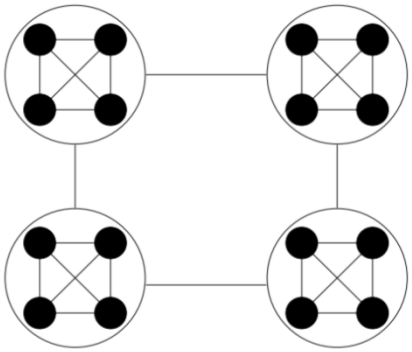


Figura 6. Arquitectura de grano grueso a dos niveles.

2.6 Procesamiento Paralelo

El procesamiento paralelo permite realizar múltiples cálculos de manera simultánea. Existen diversas clasificaciones, tecnologías e implementaciones que permiten realizar procesamiento paralelo, a continuación, se hará mención de alguna de ellas.

2.6.1 Taxonomia de Flynn

Existen diversos esquemas de clasificación del cómputo paralelo, una de ellas es la taxonomía de Flynn, la cual se basa en cómo se organiza el flujo de datos y de instrucciones.

SISD (Single Instruction Stream, Single Data Stream): Se ejecuta una instrucción a un dato. Es una arquitectura de procesamiento secuencial.

SIMD (Single Instruction Stream, Multiple Data Stream): Se ejecuta una instrucción en un conjunto de datos al mismo tiempo. También se le conoce como paralelismo de datos y es especialmente útil para el procesamiento de imágenes ya que la misma instrucción se puede usar en para manipular múltiples pixeles en una unidad de tiempo. La cantidad de pixeles que pueden ser manipulados al mismo tiempo depende del hardware y de la propia implementación de la arquitectura.

MISD (Multiple Instruction Stream, Single Data Stream): Se ejecutan múltiples instrucciones a un dato. Es un modelo teórico que no ha teindo mucha difusión.

MIMD (Multiple Instruction Stream, Multiple Data Stream): Es el modelo más general de paralelismo. Las ideas básicas son que múltiples tareas heterogéneas puedan ser ejecutadas al mismo tiempo, y que cada procesador opere independientemente con ocasionales sincronizaciones entre si. Este modelo se divide principalmente en la forma de gestionar la memoria que se divide en Memoria Compartida y Memoria Distribuida.

2.6.2 Memoria Compartida

En esta arquitectura, el conjunto de procesadores comparte una memoria común memoria en la que cada procesador puede ejecutar una instrucción diferente [35] siendo realizada la comunicación entre los procesadores a través de la memoria. Este tipo de arquitectura tiene la ventaja de poder paralelizar programas secuenciales sin cambios significativos, pero al aumentar el número de procesadores se incrementan los costos de coordinación de la memoria, ya que más de un procesador podría querer acceder a la misma dirección de memoria. El esquema de memoria compartida se ilustra en la Figura 7.

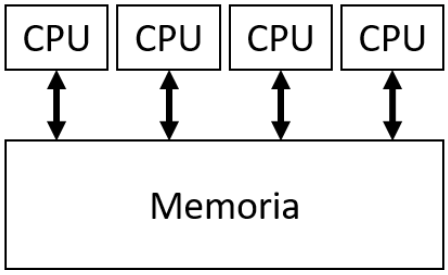


Figura 8. Esquema de memoria compartida.

Comúnmente se utiliza OpenMP para desarrollo de aplicaciones con el paradigma de Memoria Compartida.

2.6.3 OpenMP

Es una Interfaz de Programación de Aplicaciones (API) que es portable y escalable que brinda a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas con memoria compartida [36]. OpenMP es multiplataforma y soporta lenguaje C, C++ y Fortran. La API consiste en un conjunto de directivas de compilador, rutinas, funciones en tiempo de ejecución y variables de entorno.

OpenMP permite paralelizar instrucciones poniendo sus directivas en la instrucción a paralelizar aunque también permite desarrollar acciones más complejas. Otra de las características es que no es necesaria la declaración explicita de hilos ya que las directivas se encargan de ello.

Cabe aclarar que no todo es propenso a paralelizar, la creación y destrucción de los hilos implica un costo computacional que puede aumentar el tiempo de ejecución.

Las directivas de OpenMP inician con el prefijo "**#pragma omp**" y tiene soporte para paralelizar ciclos y secciones de código además de funciones atómicas que garantizan la congruencia de los datos y funciones para la medición de tiempos de ejecución.

Entre las directivas más utilizadas se encuentran [36]:

#pragma omp parallel

Forma un grupo de hilos y empieza la ejecución en paralelo.

#pragma omp single

Especifica que el bloque es ejecutado sólo por un hilo.

#pragma omp for

Especifica que las iteraciones de los ciclos serán ejecutados en paralelo.

#pragma omp master

Especifica que el bloque será utilizado sólo por el hilo maestro.

#pragma omp critcal

Restringe la ejecución del bloque a un solo hilo a la vez.

#pragma omp barrier

Especifica una barrera explicita. El hilo no continuará con la siguiente instrucción hasta que todos los hilos hayan llegado a este punto.

#pragma omp atomic

Se asegura que una locación de memoria es accesar atómicamente.

Entre las funciones más utilizadas se encuentran [36]:

omp_set_num_threads

Define el número de hilos a utilizar

omp_get_num_threads

Retorna el número de hilos ejecutándose.

omp_get_max_threads

Retorna el máximo de hilos soportados.

omp_get_thread_num

Retorna el identificador del hilo actual.

omp_get_wtime

Retorna el tiempo actual.

La creación y destrucción de hilos tiene un costo considerable por lo que se aconseja paralelizar los ciclos más externos, con mayor número de iteraciones o que tenga una carga considerable de trabajo.

OpenMP tiene algunos problemas con la escalabilidad, cuando el número de hilos se incrementa puede haber inconvenientes si más de un hilo quiere acceder a la misma locación de memoria.

2.6.4 Memoria Distribuida

Se caracteriza por un control distribuido y una distribución de los datos. Cada procesador tiene su propia memoria local comunicándose entre sí a través de una red. El rendimiento

está fuertemente ligado a la velocidad de las comunicaciones entre las máquinas, y el paralelismo a utilizar debe ser explícito, lo que hace requerir nuevos sistemas de explotación y ambientes de programación. El esquema de memoria distribuida se puede observar en la Figura 9.

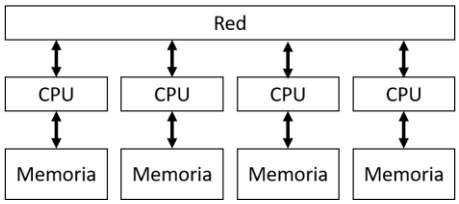


Figura 10. Esquema de Memoria Distribuida.

Su mayor ventaja es la gran escalabilidad mientras que su mayor desventaja son los problemas de sincronización de las memorias. Para implementar el paradigma de memoria distribuida se hace uso de librerías que sigan el estándar MPI.

2.6.5 MPI

MPI ("Message Passing Interface", Interfaz de Paso de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

Su mayor ventaja es la gran escalabilidad mientras que su mayor desventaja son los problemas de sincronización de las memorias. Para implementar el paradigma de memoria distribuida se hace uso de librerías que sigan el estándar MPI [35].

La principal característica de MPI es la comunicación explicita entre procesos por lo cual los métodos de comunicación son de gran relevancia. A continuación se mencionan los métodos de comunicación que son usados con mayor frecuencia [37]:

MPI_Send

Función de envío de mensaje bloqueante de un proceso de origen a uno de destino. Al ser bloqueante significa que hasta que el mensaje no haya sido enviado (que salga del buffer de salida) no se continúa la ejecución.

MPI_Recv

Esta función bloquea el proceso hasta que se reciba un mensaje con las características especificadas.

MPI_Isend

Esta función sirve para realizar el envío de un mensaje desde un origen a un único destino de forma no bloqueante. Esto significa que la función devuelve el control a la ejecución del programa lo antes posible, es decir, hasta que el hardware y el sistema operativo sean capaces

de realizar el resto del envío por sí solos. Con éste método no se puede asegurar que el mensaje se haya enviado excepto consultando a la variable MPI_Request.

Es común utilizar este método para enviar mensajes muy largos, ya que puede llevar tiempo cargarlo en el buffer, se puede comenzar el envío lo antes posible y continuar parte de la ejecución, hasta que sea estrictamente necesario conocer que se ha mandado el mensaje. En este caso, se puede hacer uso de la función MPI_Wait.

MPI_Irecv

Esta función es para comenzar el recibimiento de un mensaje. Lo que hace es bloquear el proceso hasta que se le notifique la llegada de un mensaje. Cuando esto suceda, pedirá que se comience a recibir el mensaje, a la vez que continúa la ejecución del resto del proceso. Una vez nos es necesario utilizar el mensaje, es obligatorio utilizar alguna directiva de MPI para detener la ejecución (como MPI_Wait), o bien comprobar el estado del recibo (por ejemplo con MPI_Test).

MPI_Bcast

Envía un único mensaje desde el proceso raiz a todos los procesos del grupo, incluyendose a sí mismo. Todos los procesos llaman a la función exactamente de la misma manera, usando siempre los mismos argumentos. Cuando se haya completado la función, todos los procesos tendrán una copia en su buffer de entrada de lo enviado por el proceso raíz.

MPI_Scatter

Un proceso raíz parte un mensaje en partes iguales y los envía individualmente al resto de procesos y a sí mismo.

MPI_Gather

Recoge una serie de datos de varios procesos en un único proceso raíz (operación en la cual interviene también el propio proceso raíz).

2.6.6 Ley de Amdahl

No es posible paralelizar el 100% de un programa, por lo tanto, la reducción de tiempo de un programa no es directamente proporcional al número de procesadores con los que se ejecute el programa. Para hacer el cálculo del incremento máximo de la velocidad del programa en paralelo se puede usar la siguiente ecuación:

$$\frac{1}{(1-p)+\frac{p}{N}}$$

Dónde:

p: Es el porcentaje de código que es paralelo ($0<p<1$).

N: Es el número de procesadores disponibles.

Si el 80% del código se puede paralelizar la ecuación quedaría de la siguiente manera.

$$\frac{1}{(1-0.8)+\frac{0.8}{N}} = \frac{1}{(0.2)+\frac{0.8}{N}}$$

Suponiendo que se contara con una cantidad ilimitada de procesadores entonces lo máximo que se podría reducir el tiempo es $1/(1-p)$.

$$\frac{1}{(1-p)+\frac{p}{\infty}} = \frac{1}{(1-p)} = \frac{1}{(1-0.8)} = \frac{1}{0.2} = 5$$

Por lo tanto, en el mejor de los casos sería 5 veces más rápido el programa paralelo comparado con el secuencial. Esto es desde el punto de vista teórico, sin embargo, no siempre se reparte equitativamente los procesos entre los procesadores disponibles lo que evita que se logre la reducción de tiempo de ejecución teórica. Además, entre mayor número de procesadores debe de haber mayor comunicación entre ellos añadiendo tiempo en la ejecución.

2.6.7 Eficiencia del algoritmo paralelo

Para calcular la eficiencia de un algoritmo debemos de calcular la relación entre el tiempo de ejecución de un algoritmo en una unidad de procesamiento con respecto al que le toma en *n* unidades, también es conocido como *speedup* o *aceleramiento*:

$$speedup = \frac{T(1)}{T(n_{procesadores})}$$

Se espera que el *speedup* sea mayor que 1, lo que indicaría que es factible la paralelización del algoritmo.

La *eficiencia* es la relación entre el *speedup* y el número de procesadores con los que se ejecutó:

$$Eficiencia = \frac{speedup}{n}$$

en donde n es el número de procesadores.

2.7 Ajuste de parámetros

La correcta parametrización del problema a resolver es fundamental ya que esto impacta en la eficiencia del proceso de resolución del problema, por ello es importante la *configuración paramétrica* que consiste en la combinación de valores que los parámetros podrían utilizar.

Para la aplicación de un metaheurístico a un problema se debe de definir un esquema de representación y la función objetivo, juntos son un intermediario entre el problema y su algoritmo de solución. Tanto el esquema de representación como la función objetivo necesitan valores para cada uno de sus parámetros, de tal manera que se represente adecuadamente el problema y se logre una mejor eficiencia en el proceso de resolución [38]. A las combinaciones de valores de los parámetros se le llama *configuración paramétrica*, y al problema de seleccionar la mejor configuración se le llama *problema de configuración* o *ajuste de parámetros* [39] [40].

Existen clasificaciones de tipos de parámetros, entre ellas la formulada por Eiben, Hinterding y Michalewicz [41] que se divide en dos tipos de adaptación: Estática (afinación de parámetros) y Dinámica (control de parámetros). La afinación de parámetros se realiza antes de la ejecución del algoritmo, y el control de parámetros se realiza durante la ejecución del algoritmo.

La *Afinación de Parámetros* propone una configuración inicial estática que no es modificada durante la ejecución del algoritmo y en base a los resultados de la experimentación se determina el conjunto de parámetros que tuvieron el mejor desempeño. La estrategia más usual para obtener una configuración inicial de parámetros consiste en probar muchas configuraciones de manera manual y escoger la de mejor resultado [38].

El *Control de Parámetros,* monitorea los cambios en el ambiente al mismo tiempo que considera el estado actual del algoritmo. Eiben [40] sugiere realizar una configuración paramétrica inicial a partir de técnicas de afinación y posteriormente ejercer control sobre los parámetros.

2.8 Diseño metodológico

Para diseñar un algoritmo paralelo es necesario contemplar algunas situaciones que no se presentan en el diseño secuencial de los algoritmos, por ello Foster [42] propone una metodología llamada PCAM que facilita el diseño de los algoritmos paralelos. Esta metodología se divide en cuatro etapas:

1. **Particionamiento:** Los cálculos o información son descompuestos en segmentos más pequeños. Se hace énfasis en reconocer oportunidades de paralelización.
   1.1. **Descomposición del dominio:** Se descompone primero los datos en fragmentos de menor tamaño y se fragmenta el código acorde a la fragmentación de los datos.
   1.2. **Descomposición funcional:** Se centra primero en los cálculos que van a ser efectuados antes que los datos que van a ser calculados.

2. **Comunicación:** Se define las comunicaciones requeridas para coordinar la ejecución, y las estructuras de comunicación apropiadas y los algoritmos son definidos.
3. **Aglomeración:** Es evaluado el particionamiento y las comunicaciones previamente definidas, de ser necesario se pueden fusionar tareas para mejorar el rendimiento o para reducir costos de desarrollo.
4. **Mapeo:** Su meta es maximizar el uso de los procesadores y minimizar los costos de comunicación. El mapeo puede ser estático o determinado en tiempo de ejecución por un algoritmo de balanceo de cargas.

Capítulo 3: Estado del Arte

En esta sección se describen algunos de los trabajos usados para el desarrollo de esta investigación. Los trabajos seleccionados están relacionados con el problema de investigación o su estrategia de solución.

El problema de selección de cartera de proyectos es un problema que se ha estudiado ampliamente desde un enfoque multi-objetivo [43] [44] [45] [46] [47], en la literatura sin embargo, no se han encontrado trabajos que lo aborden como un problema de mono-objetivo. Algunos autores consideran al problema como una variante del problema de la mochila [44], para el cual existen trabajos que tienen un enfoque mono-objetivo que se comentarán en breve.

3.1 Descripción de trabajos relacionados

Comparing between different approaches to solve the 0/1 Knapsack problem

Autores: Ameen Shaheen y Azzam Sleit

En este trabajo [48] se hace una comparación entre 4 estrategias para dar solución al problema de la mochila, entre ellas los algoritmos avaros, programación dinámica, ramificación y poda, y algoritmos genéticos describiendo cada una de las estrategias y ejemplificando cada una de ellas además de proporcionar la complejidad de los algoritmos en su peor caso.

En la experimentación se trabajó con un modelo de programación lineal entera con un solo objetivo y una única restricción, usando instancias de hasta 500 mil variables, exceptuando el caso del método de ramificación debido a falta de memoria RAM en el equipo de cómputo, haciendo la experimentación de este algoritmo con instancias de hasta 60 mil variables. Debido a esta limitante, se dividió la experimentación en 2 conjuntos de instancias, el primer grupo de 20, 30, 40, 50 y 60 mil variables que pueden ser ejecutadas en los 4 algoritmos y el segundo grupo con 100, 200, 300, 400 y 500 mil variables que son ejecutadas por los algoritmos exceptuando a la ramificación y poda.

En cuanto a los tiempos de ejecución en el primer conjunto de instancias, se puede observar en el trabajo que el algoritmo avaro es el que da solución al problema en menor tiempo y el algoritmo de ramificación y poda es el que mayor tiempo necesita, creciendo exponencialmente su tiempo de ejecución al aumentar el número de variables, siendo 10 veces mayor el tiempo requerido por el método de ramificación y poda que por el algoritmo avaro en la instancia de 20 mil variables, creciendo la diferencia a más de 100 veces mayor el tiempo requerido por el algoritmo de ramificación y poda para la instancia de 60 mil variables. El algoritmo avaro tiene tiempos de ejecución muy similares al algoritmo genético y el algoritmo de programación dinámica ronda entre 2 y 2.5 veces mayor tiempo de ejecución.

En el segundo bloque de instancias el algoritmo que menor tiempo de ejecución requiere es el algoritmo genético y el que mayor tiempo consume es la programación dinámica, requiriendo en promedio 10 veces mayor tiempo de ejecución. La calidad de las soluciones no está reportada para los dos grupos de instancias, habiendo un tercer grupo de instancias entre 100 y 1000 variables en el cual, si se reporta la calidad de las soluciones generadas por el algoritmo avaro, genético y de programación dinámica proporcionando este último el óptimo global mientras que los otros dos un óptimo local, habiendo un error máximo en el algoritmo avaro de 3% y de 1% en el algoritmo genético.

Secuential and Parallel Implementation of GRASP for the 0-1 Multidimensional Knapsack Problem

Autores: Bianca de Almeida Dantas and Edson Norberto Cáceres

En este trabajo [49] se hace una hibridación de entre un algoritmo GRASP y una búsqueda local, para el que se implementó una versión secuencial y una paralela en GPU. La experimentación fue realizada usando las instancias de la librería ORLIB, estas instancias fueron en grupos de 5, 10 o 30 restricciones y de 100, 250 y 500 objetos. Se menciona que el número de iteraciones manejado en los programas paralelos fue de 1000 mientras que en la versión paralela en GPU fue de 100 iteraciones pero con 100 bloques de hilos, exceptuando en las instancias de 500 objetos dónde solo se hizo uso de bloques de 20 hilos.

El algoritmo fue comparado contra el algoritmo genético de Chu and Bealey [50], un algoritmo neurogenético de Deane and Agarwal [51] y un algoritmo de colonia de hormigas de Fingler et al [52]. Aunque el algoritmo GRASP tuvo los mejores resultados en las instancias de 100 y 250 elementos, el algoritmo genético y neurogenético tuvieron mejor resultado en las instancias de 250 elementos, a diferencia de las instancias agrupadas por número de restricciones dónde el algoritmo GRASP fue el de mejor rendimiento.

A Parallelization of a Simulated Annealing Approach for 0-1 Multidimensional Knapsack Problem Using GPGPU

Autores: Bianca de Almeida Dantas and Edson Norberto Cáceres

En este trabajo [53] da solución al problema de la mochila multidimensional haciendo una hibridación entre un algoritmo GRASP, el cual genera la solución inicial y un recocido simulado que es paralelizado su ciclo de la cadena de Markov y es descompuesto el dominio del problema, los procesos se comunican tanto de manera síncrona como asíncrona para sincronizarse entre ellos. La paralelización del algoritmo fue realizada en un procesador tipo SIMD (Simple Instruction, Multiple Data) obteniendo mejores resultados comparado con la versión secuencial del algoritmo, teniendo una mejoría promedio del 39% y fue 5 veces más rápida la versión paralela con las instancias de OR Library.

A Parallel Nash Genetic Algorithm for the 3D Orthogonal Knapsack Problem

Daniel Soto, Wilson Soto y Yoan Pinzón

Este trabajo [54] da solución al problema de la mochila de 3 dimensiones, consiste en encontrar el máximo beneficio al empacar un subconjunto de cajas es una caja más grande (contenedor). Las cajas por empacar son rectangulares, pero de diferentes tamaños.

Para dar solución al problema se hace uso de un algoritmo genético con un modelo de islas y con la teoría de equilibrio de Nash. El algoritmo genético distribuye su población en subpoblaciones que ocasionalmente intercambia individuos entre las poblaciones, este intercambio de soluciones entre las poblaciones se le conoce como migración, los individuos que van a ser intercambiados son elegidos de manera aleatoria. Las soluciones de este algoritmo tienen dos tipos de función objetivo, en las subpoblaciones par el algoritmo busca minimizar el volumen vacío y en las subpoblaciones impar busca maximizar el beneficio.

La paralelización del algoritmo se llevó a cabo en un procesador tipo MIMD (Multiple Instruction, Multiple Data) usando OpenMP con asignación dinámica y una estructura de memoria compartida que permite el intercambio entre las subpoblaciones.

3.2 Análisis de trabajos relacionados

En la Tabla 1 se resumen los trabajos que presentan información relevante para esta investigación. Aquí se puede observar que todos los trabajos abordan el problema de la mochila en versión mono objetivo a diferencia de esta investigación que aborda el problema de selección de cartera de proyectos en su versión mono objetivo también. Cabe señalar que el trabajo de estudio de este trabajo pertenece a la familia del problema de la mochila. Otro factor observado es la estrategia de solución que abordan los diferentes trabajos, encontrándose diversas estrategias tales como: deterministas, heurísticas y metaheurísticas como lo son los algoritmos genéticos; ya que estos son los preferidos para solucionar problemas con grandes espacios de soluciones. Otra característica que se identificó en los trabajos relacionados del estado del arte es la hibridación, encontrándose que la mayoría de los trabajos hacen hibridación; siendo estas hibridaciones entre

heurísticas y metaheurísticas a diferencia de este trabajo de investigación que la realiza entre métodos de programación lineal entera y metaheurísticas. Finalmente se analizó quienes de los investigadores hacen uso de estrategias de paralelismo aplicadas a su algoritmo de solución, encontrándose que esta característica también está presente en la mayoría de los trabajos de referencia pero haciendo uso de diferentes arquitecturas como lo son las arquitecturas SIMD (Simple Instruction, Multiple Data) y la MIMD (Multiple Instruction, Multiple Data), y diferentes balanceos de cargas como lo son el estático y el dinámico.

Tabla 1. Trabajos relacionados con esta tesis.

Trabajo	**Problema de estudio**	**Tipo de función objetivo**	**Algoritmo de Solución**	**Incluye Hibridación**	**Paralelismo y/o patrones paralelos**
Shaheen y Sleit [48]	Problema de la Mochila	Mono-objetivo	Branch and Bound, Programación Dinámica, Algoritmo Greedy y Algoritmo Genético	No	
de Almeida y Cáceres [49]	Problema de la mochila multi-dimensional	Mono-objetivo	GRASP con búsqueda local	Sí	SIMD
de Almeida y Cáceres [53]	Problema de la mochila multi-dimensional	Mono-objetivo	GRASP con Recocido Simulado	Sí	SIMD / Descompo-sición del dominio del problema
Soto et al. [54]	Problema de la mochila en 3 dimensiones	Mono-objetivo	Algoritmo Genético con Modelo de Islas y teoría de Nash	Sí	MIMD
Esta tesis	Selección de cartera de proyecto.	Mono-objetivo	Algoritmo genético con estrategias de asignación de recursos.	Sí	MIMD / Descompo-sición de tareas.

Capítulo 4: Propuesta de Solución

En este capítulo se describirá la metodología utilizada para la resolución del problema. La descripción incluye características del algoritmo genético que se utilizó para dar solución al problema, como lo es su selección, cruzamiento y mutación, además, se describe la paralelización del algoritmo.

Propuesta de Metodología de Solución

En este trabajo se propone el desarrollo de un algoritmo paralelo para la solución del problema de selección de cartera de proyectos. Como se describió en el capítulo 2 en la sección 2.8, se establecen los elementos básicos de esta metodología. En este capítulo se retomará la metodología y se describirá el problema de estudio de esta tesis. En la Figura 11 se muestra el diagrama general de la metodología propuesta por Foster [42].

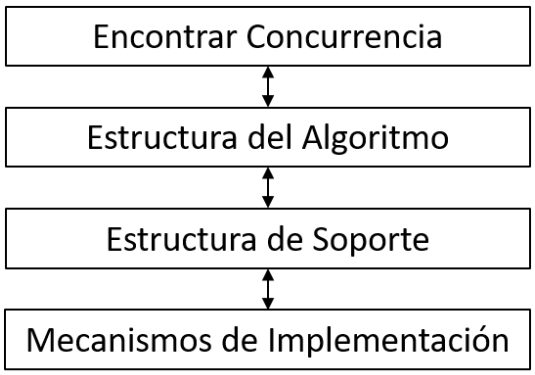


Figura 11. Metodología General de Patrones de Diseño para Algoritmos Paralelos [42].

Enseguida se detallarán cada una de las etapas con la información necesaria para la solución del problema de estudio. Cabe señalar que agrego una etapa 0, donde se hace un análisis de la propuesta de solución secuencial ya que es una parte importante de la solución del problema de estudio.

Etapa 0: Análisis de la propuesta de solución secuencial

En esta etapa se describen los elementos necesarios para establecer los módulos de la propuesta de solución secuencial.

Descripción de Algoritmo: La propuesta de solución trabajará con un algoritmo genético que se establece como una estrategia de solución al problema de estudio, dicho algoritmo contiene los siguientes elementos:

Variables:

La Tabla 2 muestra el conjunto de variables necesarias.

Tabla 2. Variables del problema y de los algoritmos.

N	Número de variables del problema
M	Número de restricciones del problema
A	Matriz de restricciones del problema. Su dimensión es de *N*x*M* elementos de tipo real.
B	Vector de desigualdades de las restricciones del problema. Su dimensión es de M elementos de tipo real.
C	Vector de beneficios de la Función Objetivo. Su dimensión es de N elementos de tipo real.
TP	Tamaño de la población
P	Vector de Solución de tamaño TP, está formado por los siguientes elementos: Solución: Vector de tamaño N que representa la cartera tipo binario. wSol: Vector de costos por restricción de tamaño M tipo real. bSol: Beneficio de la solución tipo real. nInf: Número de restricciones infactibles tipo entero corto. exceso: Sumatoria del recurso excedido en las restricciones tipo real. card: Número de proyectos financiados en la solución tipo entero.
G	Número de generaciones del algoritmo evolutivo
Porcentaje de Población Inicial	Posibilidad de incluir la actividad en la solución inicial.
Porcentaje de Migración	Porcentaje de la población que será migrada a la población contigua.

Enseguida de las variables se mostrarán los algoritmos propuestos para la solución del problema.

Algoritmos

Enseguida se describen los diferentes algoritmos que forman la estrategia de solución. El algoritmo 1 muestra la estrategia de solución general, la cual incluye todos los módulos necesarios para formar el algoritmo genético.

Algoritmo1: Estrategia General del Algoritmo Genético.	
Entrada:	**I: Instancia** **a: Porcentaje de aceptación en la población inicial**
	$\boldsymbol{P = PoblacionInicial(I, a)}$ $\boldsymbol{Ordenamiento(P)}$ $\boldsymbol{for\ g = 1 \rightarrow G}$ $\quad\boldsymbol{for\ k = |P|/2 -> |P|}$ $\quad\quad\boldsymbol{Padre_1 = seleccion(P_0 \cdots P_{N/2})}$ $\quad\quad\boldsymbol{Padre_2 = Padre_1}$ $\quad\quad\boldsymbol{while(Padre_1 == Padre_2)\ //\ Igualdad\ en\ posicion}$ $\quad\quad\quad\boldsymbol{Padre_2 = seleccion(P_0 \cdots P_{N/2})}$ $\quad\quad\boldsymbol{(Hijo1, Hijo2) = Cruza(Padre1, Padre2, I)}$; $\quad\quad\boldsymbol{Hijo_1' = Mutacion\ (Hijo_1, I)}$ $\quad\quad\boldsymbol{Hijo_2' = Mutacion\ (Hijo_2, I)}$ $\quad\quad\boldsymbol{P_k = Hijo_1'}$ $\quad\quad\boldsymbol{P_{k+1} = Hijo_2'}$ $\quad\quad\boldsymbol{Ordenamiento(P)}$

Enseguida el algoritmo2 muestra el módulo de la generación de la población inicial.

Algoritmo2: Generación de la Población Inicial	
Entrada:	a: Porcentaje de aceptación. I: Instancia del problema Salida: P: P soluciones generadas aleatoriamente.
	$\boldsymbol{Población\_Elite(P)}$ $\boldsymbol{for\ i = 1 \rightarrow |P|}$ $\quad\boldsymbol{for\ j = 1 \rightarrow N}$ $\quad\quad\boldsymbol{if(aleatorio() < a)}$ $\quad\quad\quad\boldsymbol{P_{i_{card}} = P_{i_{card}} + 1}$ $\quad\quad\quad\boldsymbol{P_{i_{Sol_j}} = 1}$ $\quad\quad\quad\boldsymbol{P_{i_{bSol}} = P_{i_{bSol}} + C_j}$ $\quad\quad\quad\boldsymbol{for\ k = 1 \rightarrow M}$ $\quad\quad\quad\quad\boldsymbol{P_{i_{wSol_k}} = P_{i_{wSol_k}} + A_{k,j}}$ $\quad\boldsymbol{for\ j = 1 \rightarrow M}$ $\quad\quad\boldsymbol{if\ (P_{i_{wSol_k}} > B_j)}$ $\quad\quad\quad\boldsymbol{P_{i_{nInf}} = P_{i_{nInf}} + 1}$ $\quad\quad\quad\boldsymbol{P_{i_{exceso}} = P_{i_{exceso}} + P_{i_{wSol_k}} - B_j;}$

Como se observa en el algoritmo 2, para las *P* soluciones de la población, se calcula su vector de solución, acorde a un porcentaje de aceptación generando números aleatorios, si el numero aleatorio es menor que el porcentaje establecido entonces se financia el proyecto y/o actividad. Al acabar de conocer los elementos financiados se procede a calcular su factibilidad, en caso de existir restricciones infactibles entonces se contabilizan y se acumula por cuanto fueron excedidas las restricciones.

Después de la generación de la población, esta se ordenará para preparar las soluciones para la estrategia de selección de padres ya que a través de este ordenamiento se genera una lista con los mejores padres, esto se puede ver en el algoritmo 3.

Algoritmo3: Ordenamiento de soluciones	
Entrada:	*P*: P soluciones generadas aleatoriamente. Se ordenan las soluciones en base a los siguientes criterios: 1. Menor número de restricciones infactibles. 2. Menor sumatoria de la cantidad en que fueron excedidas las restricciones. 3. Mayor beneficio de la solución 4. Mayor cardinalidad de la solución.
	$\boldsymbol{for\ i\ = 1 -> N}$ $\boldsymbol{for\ j\ = 0 -> N - i}$ $\boldsymbol{if}\ (\boldsymbol{P_{nInf_j} > P_{nInf_{j+1}}}\ \|\|$ $(\boldsymbol{P_{nInf_j} = P_{nInf_{j+1}}}\ \&\&\ \boldsymbol{P_{exceso_j} > P_{exceso_{j+1}}})\ \|\|$ $(\boldsymbol{P_{exceso_j} = P_{exceso_{j+1}}}\ \&\&\ \boldsymbol{P_{pSol_j} > P_{pSol_{j+1}}})\ \|\|$ $(\boldsymbol{P_{pSol_j} = P_{pSol_{j+1}}}\ \&\&\ \boldsymbol{P_{card_j} < P_{card_{j+1}}}))$ $\boldsymbol{swap}(\boldsymbol{P_{pSol_j}}, \boldsymbol{P_{pSol_{j+1}}})$ $\boldsymbol{swap}(\boldsymbol{P_{wSol_j}}, \boldsymbol{P_{wSol_{j+1}}})$ $\boldsymbol{swap}(\boldsymbol{P_{Sol_j}}, \boldsymbol{P_{Sol_{j+1}}})$ $\boldsymbol{swap}(\boldsymbol{P_{exceso_j}}, \boldsymbol{P_{exceso_{j+1}}})$ $\boldsymbol{swap}(\boldsymbol{P_{nInf_j}}, \boldsymbol{P_{nInf_{j+1}}})$ $\boldsymbol{swap}(\boldsymbol{P_{card_j}}, \boldsymbol{P_{card_{j+1}}})$

Posteriormente se llevará acabo el algoritmo 4, el cual muestra el módulo de selección de padres.

Algoritmo4: Selección de Padres	
Entrada:	P: Vector de Población
	$\boldsymbol{Indice_1 = Aleatorio(\|P\|/2)}$ $\boldsymbol{Indice_2 = Aleatorio(\|P\|/2)}$

```
while Indice1 = Indice2
    Indice2 = Aleatorio(P/2)
if( P_Indice1_nInf < P_Indice2_nInf)
    return Indice1
if(P_Indice1_nInf > P_Indice2_nInf)
    return Indice2
else
    if(P_Indice1_exceso < P_Indice2_exceso)
        return Indice1
    if(P_Indice1_exceso > P_Indice2_exceso)
        return Indice2
    else
        if(P_Indice1_bSol > P_Indice2_bSol)
            return Indice1
        if(P_Indice1_bSol < P_Indice2_bSol)
            return Indice2
        else
            if(P_Indice1_card > P_Indice2_card)
                return Indice1
            if(P_Indice1_card < P_Indice2_card)
                return Indice2
            else
                if(Aleatorio() < 0.5)
                    return Indice1
                else
                    return Indice2
```

En el módulo de selección de padres se cuenta con *P* soluciones ordenadas bajo los criterios previamente comentados y se busca elegir uno de los padres que se utilizará en la cruza, para ello se elige aleatoriamente dos números entre 0 y |*P*|/2, ya que la primer mitad de las soluciones servirán de padres y la segunda mitad será para almacenar los hijos producidos en la cruza, aquí el aleatorio obtenido representa el índice de la solución que se pretende tomar como padre.

Los dos índices aleatorios no deben ser iguales entre si y se seguirá tomando otro número aleatorio hasta que los índices difieran. Una vez que se obtuvieron los dos índices se compara primero el número de infactibilidades de ambas soluciones seleccionando el que menor número de infactibilidades tenga, si hay empate entonces se toma la solución con menor cantidad de exceso, si persiste el empate entonces se toma como tercer criterio la solución que mayor beneficio tenga, si hay empate el cuarto criterio será la solución con mayor cardinalidad y si persiste el empate se elige aleatoriamente entre ambas soluciones.

Ejemplo de la selección de padres:

Se tiene un problema con 6 variables con los siguientes costos y benéficos.

Beneficios					
1	2	3	4	5	6

Costos					
1	2	3	3	2	1

Teniendo el siguiente conjunto de soluciones

Índice	Soluciones						Beneficio	Costo	nInf	exceso	Card
0	0	0	1	0	1	1	14	6	0	0	3
1	1	1	0	1	0	0	7	6	0	0	3
2	0	1	0	1	0	0	6	5	0	0	2
3	0	1	1	0	1	0	10	7	1	1	3
4	1	0	1	1	0	0	8	7	1	1	3
5	0	0	1	1	1	1	18	9	1	3	4
6	0	1	1	1	0	1	15	9	1	3	4
7	1	1	1	1	0	0	10	9	1	3	4

Se tome aleatoriamente dos números

Indice1 = 2

Indice2 = 2

Como Indice1 es igual a Indice2 entonces se genera un nuevo número aleatorio

Indice2 = 0

Ahora se compara en número de infactibilidades del vector $nInf$ en su posición $Indice_1$ e $Indice_2$

$nInf_{\text{Indice1}} \sim nInf_2 = 0$

$nInf_{\text{Indice2}} \sim nInf_0 = 0$

Como el número de infactibilidades es 0 el exceso también lo será, el tercer factor es el beneficio

$Beneficio_{\text{Indice1}} \sim Beneficio_2 = 6$

$Beneficio_{\text{Indice2}} \sim Beneficio_0 = 14$

Como el beneficio ofrecido por la solución 0 es mayor que el beneficio de la solución 2 entonces la solución 0 será uno de los padres de la próxima generación.

Se continúa con el algoritmo 5 donde se muestra el módulo de cruza de soluciones.

Algoritmo 5: Cruza de Soluciones	
Entrada:	Padre1: Solución elegida por el método de selección Padre2: Solución elegida por el método de selección

	I: Instancia del problema a resolver
	$\boldsymbol{for\ i = 1 \rightarrow N}$ $\boldsymbol{if(Aleatorio() < 0.5)}$ $\boldsymbol{if(Padre1_{Solucion[i]} = 1)}$ $\boldsymbol{Hijo1_{card} = Hijo1_{card} + 1}$ $\boldsymbol{Hijo1_{bSol} = Hijo1_{bSol} + C_i}$ $\boldsymbol{Hijo1_{Solucion[i]} = 1}$ $\boldsymbol{for\ j = 1 \rightarrow M}$ $\boldsymbol{Hijo1_{wSol[j]} = Hijo1_{wSol[j]} + A_{j,i}}$ $\boldsymbol{else}$ $\boldsymbol{Hijo1_{Solucion[i]} = 0}$ $\boldsymbol{if(Padre2_{Solucion[i]} = 1)}$ $\boldsymbol{Hijo2_{card} = Hijo2_{card} + 1}$ $\boldsymbol{Hijo2_{bSol} = Hijo2_{bSol} + C_i}$ $\boldsymbol{Hijo2_{Solucion[i]} = 1}$ $\boldsymbol{for\ j = 1 \rightarrow M}$ $\boldsymbol{Hijo2_{wSol[j]} = Hijo2_{wSol[j]} + A_{j,i}}$ $\boldsymbol{else}$ $\boldsymbol{Hijo2_{Solucion[i]} = 0}$ $\boldsymbol{else}$ $\boldsymbol{if(Padre1_{Solucion[i]} = 1)}$ $\boldsymbol{Hijo2_{card} = Hijo2_{card} + 1}$ $\boldsymbol{Hijo2_{bSol} = Hijo2_{bSol} + C_i}$ $\boldsymbol{Hijo2_{Solucion[i]} = 1}$ $\boldsymbol{for\ j = 1 \rightarrow M}$ $\boldsymbol{Hijo2_{wSol[j]} = Hijo2_{wSol[j]} + A_{j,i}}$ $\boldsymbol{else}$ $\boldsymbol{Hijo2_{Solucion[i]} = 0}$ $\boldsymbol{if(Padre2_{Solucion[i]} = 1)}$ $\boldsymbol{Hijo1_{card} = Hijo1_{card} + 1}$ $\boldsymbol{Hijo1_{bSol} = Hijo1_{bSol} + C_i}$ $\boldsymbol{Hijo1_{Solucion[i]} = 1}$ $\boldsymbol{for\ j = 1 \rightarrow M}$ $\boldsymbol{Hijo1_{wSol[j]} = Hijo1_{wSol[j]} + A_{j,i}}$ $\boldsymbol{else}$ $\boldsymbol{Hijo1_{Solucion[i]} = 0}$ $\boldsymbol{for\ j = 1 \rightarrow M}$ $\boldsymbol{if(Hijo1_{wSol[j]} > B_j)}$ $\boldsymbol{Hijo1_{nInf} = Hijo1_{nInf} + 1}$ $\boldsymbol{Hijo1_{exceso} = Hijo1_{exceso} + Hijo1_{wSol[j]} - B_j}$ $\boldsymbol{if(Hijo2_{wSol[j]} > B_j)}$ $\boldsymbol{Hijo2_{nInf} = Hijo2_{nInf} + 1}$ $\boldsymbol{Hijo2_{exceso} = Hijo2_{exceso} + Hijo2_{wSol[j]} - B_j}$

Ejemplo de la estrategia de cruza:

Se tienen dos soluciones de un problema con 6 variables.

	0	1	2	3	4	5
Padre 1	0	0	1	0	1	1
Padre 2	1	1	0	1	0	0

Para realizar la cruza se recorre cada una de las posiciones del vector generando un número aleatorio entre 0 y menor que 1.

Iteración 1:

Aleatorio = 0.73

Como el aleatorio es mayor a 0.5 entonces el primer bit del Hijo 1 será copiado del Padre 2 y el Hijo 2 del Padre 1.

	0	1	2	3	4	5
Hijo 1	1	-	-	-	-	-
Hijo 2	0	-	-	-	-	-

Iteración 2:

Aleatorio = 0.41

Como el aleatorio es menor a 0.5 entonces el primer bit del Hijo 1 será copiado del Padre 1 y el Hijo 2 del Padre 2.

	0	1	2	3	4	5
Hijo 1	1	0	-	-	-	-
Hijo 2	0	1	-	-	-	-

Iteración 3:

Aleatorio = 0.20

Como el aleatorio es menor a 0.5 entonces el primer bit del Hijo 1 será copiado del Padre 1 y el Hijo 2 del Padre 2.

	0	1	2	3	4	5
Hijo 1	1	0	1	-	-	-
Hijo 2	0	1	0	-	-	-

Iteración 4:

Aleatorio = 0.33

Como el aleatorio es menor a 0.5 entonces el primer bit del Hijo 1 será copiado del Padre 1 y el Hijo 2 del Padre 2.

	0	1	2	3	4	5
Hijo 1	1	0	1	0	-	-
Hijo 2	0	1	0	1	-	-

Iteración 5:

Aleatorio = 0.96

Como el aleatorio es mayor a 0.5 entonces el primer bit del Hijo 1 será copiado del Padre 2 y el Hijo 2 del Padre 1.

	0	1	2	3	4	5
Hijo 1	1	0	1	0	0	-
Hijo 2	0	1	0	1	1	-

Iteración 6:

Aleatorio = 0.00

Como el aleatorio es menor a 0.5 entonces el primer bit del Hijo 1 será copiado del Padre 1 y el Hijo 2 del Padre 2.

Y con esto se habrían formado dos soluciones nuevas a partir de Padre 1 y Padre 2.

Padres:

	0	1	2	3	4	5
Padre 1	0	0	1	0	1	1
Padre 2	1	1	0	1	0	0

Hijos:

	0	1	2	3	4	5
Hijo 1	1	0	1	0	0	1
Hijo 2	0	1	0	1	1	0

Finalmente se presenta el algoritmo 6 donde se muestra el módulo de mutación de una solución.

Algoritmo 6: Mutación de una Solución	
Entrada:	Hijo: Solución generada por la mutación. A: Matriz de restricciones del problema. B: Vector de desigualdades. C: Vector de beneficios.
	$\boldsymbol{i = Aleatorio(N)}$ $\boldsymbol{if(Hijo_{Solucion[i]} = 1)}$ $\boldsymbol{Hijo_{Solucion[i]} = 0}$ $\boldsymbol{Hijo_{bSol} = Hijo_{bSol} - C_i}$ $\boldsymbol{Hijo_{card} = Hijo_{card} - 1}$

	$for\ j = 1 \rightarrow N$ $Hijo_{wSol[j]} = Hijo_{wSol[j]} - A_{j,i}$ $else$ $Hijo_{Solucion[i]} = 1$ $Hijo_{bSol} = Hijo_{bSol} + C_i$ $Hijo_{card} = Hijo_{card} + 1$ $for\ j = 1 \rightarrow N$ $Hijo_{wSol[j]} = Hijo_{wSol[j]} - A_{j,i}$ $Hijo_{nInf} = 0$ $Hijo_{exceso} = 0$ $for\ j = 1 \rightarrow M$ $if(Hijo_{wSol[j]} > B_j)$ $Hijo_{nInf} = Hijo_{nInf} + 1$ $Hijo_{exceso} = Hijo_{exceso} + Hijo_{wSol[j]} - B_j$

La mutación implementada en el algoritmo 6, consiste en cambiar el bit de un cromosoma elegido aleatoriamente.

Ejemplo de la estrategia de mutación:

Se cuenta con el siguiente vector de solución.

Solución	0	0	0	1	0	1

Debido a que el vector es de longitud 6 entonces se genera un número aleatorio en el rango [0,6).

Aleatorio = 2

Como la Solución en su posición 2 es igual a 0, entonces se hace 1 representando que ahora se incluirá el proyecto en la solución.

Solución	0	0	0	1	0	1
Solución’	0	0	1	1	0	1

Enseguida del análisis de los elementos de la estrategia de solución se da paso al diseño del algoritmo paralelo que incluye patrones de diseño.

Etapa 1: **Encontrar la Concurrencia**

El objetivo de esta etapa es descomponer el problema en piezas que se pueden ejecutar simultáneamente, para ello esta etapa se realiza en tres pasos que son: la descomposición, análisis de dependencias y validación, se relatarán en detalle a continuación.

Descomposición

La descomposición se aplicará en dos dimensiones: en las tareas y los datos. Es decir, se identificarán las tareas y los datos de cada tarea para encontrar las relaciones entre las tareas.

Descomposición de tareas: El problema se modela como una secuencia de instrucciones que se pueden dividir en secuencias llamadas tareas que se pueden ejecutar simultáneamente. Para que el cálculo sea eficiente, las operaciones que componen la tarea deben ser en gran medida independientes de las operaciones que tienen lugar dentro de otras tareas.

Descomposición de tareas de generación de la población inicial.

- Asignación de los proyectos a las carteras
- Calculo de beneficios de las carteras
- Asignación de recursos de Área/Región
- Calculo de Infactibilidad de las carteras

Descomposición de tareas del ordenamiento

- Comparar soluciones
- Intercambio de soluciones

Descomposición de tareas de la selección de padres

- Elección y comparación de los padres

Descomposición de tareas de la cruza de soluciones

- Generar un número aleatorio
- Copia de cromosoma padre a cromosoma hijo
- Calculo de la función objetivo
- Calculo de infactibilidad de carteras.

Descomposición de tareas de la mutación de una solución

- Selección del elemento a mutar
- Se invierte el proyecto seleccionado
- Calculo de la función objetivo
- Calculo de infactibilidad de carteras.

Descomposición de datos: Se centra en los datos requeridos por las tareas y en cómo se puede descomponer en trozos distintos. El cálculo asociado con los fragmentos de datos solo será eficiente si los fragmentos de datos pueden ser operados con relativa independencia.

En la primera capa se procede a analizar y encontrar la concurrencia, para ello se enlistan las tareas y los datos con los que se cuentan.

Datos

- Instancia
- Soluciones

La Instancia cuenta con la información relativa al beneficio y costo de los proyectos, además de los presupuestos con los que cuenta cada área y región.

Las soluciones representan la cartera generada, teniendo un vector binario que representa los proyectos seleccionados por la cartera, además de su costo y beneficio total.

Análisis de Dependencias

Las tareas forman un solo grupo, ejecutándose una seguida de la otra como se muestra en la Figura 12.

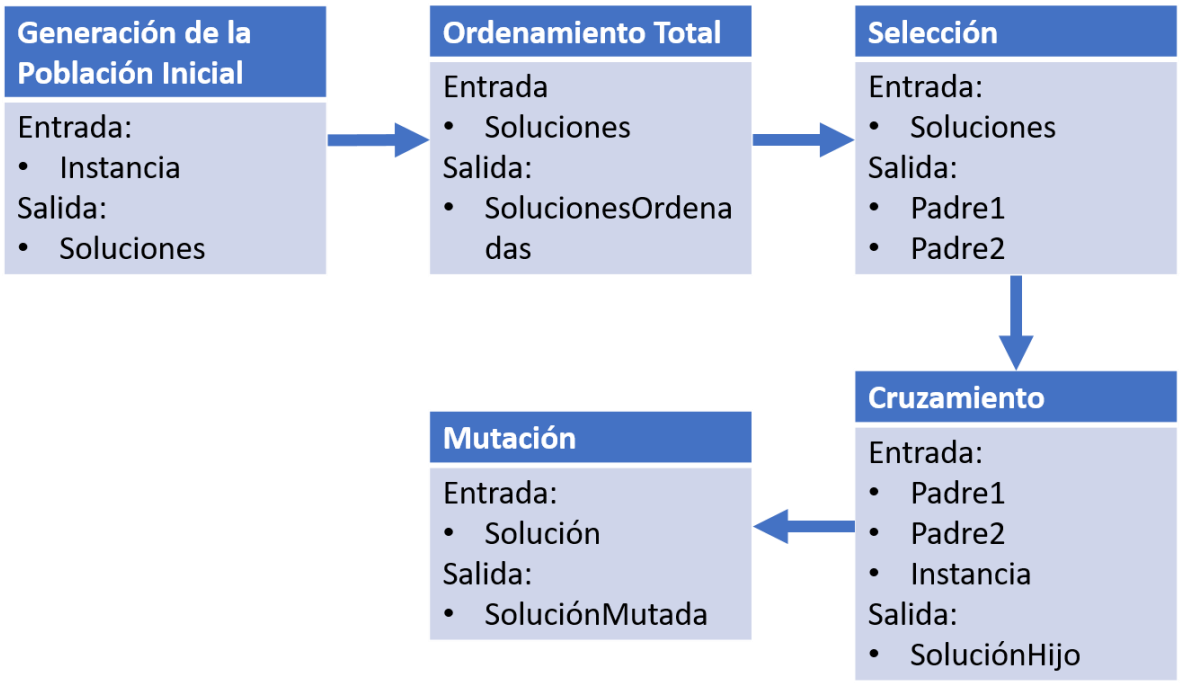


Figura 12. Grupo de Tareas y Secuencia Lógica.

Generación de la población inicial: Recibe la instancia del problema y devuelve un conjunto de soluciones.

Ordenamiento Total: Recibe las soluciones proporcionadas por el Generador de la Población Inicial y devuelve las soluciones ordenadas acorde a su función objetivo de manera descendente.

Selección: Recibe el conjunto de soluciones y devuelve dos soluciones que serán utilizadas en el cruzamiento.

Cruzamiento: Recibe dos soluciones y la instancia del problema para generar una nueva solución.

Mutación: Recibe la solución generada por el Cruzamiento y la instancia del problema para hacer una perturbación en la solución, devuelve la solución modificada.

Como existe una estricta secuencia en las tareas, no se podrían realizar múltiples tareas en paralelo, siendo una alternativa que cada secuencia de tareas sea desarrollada por un hilo o nodo de procesamiento.

Evaluación del Diseño

Al finalizar las subetapas se observa que no se puede explotar la concurrencia debido a las relaciones inherentes entre las tareas que son parte de la estrategia de solución.

Etapa 2: Estructura del Algoritmo

El siguiente paso es encontrar una estructura de algoritmo que represente cómo esta simultaneidad se asigna a las unidades de ejecución. Usualmente hay un principio de organización importante implicado en este proceso. Esto generalmente cae en uno de los tres campos: organización por tareas, organización por descomposición de datos y organización por flujo de datos.

Para algunos problemas, en realidad solo hay un grupo de tareas activo a la vez, y la forma en que interactúan las tareas dentro de este grupo es la característica principal de la concurrencia. Para otros problemas, se destaca la forma en que los datos se descomponen y se comparten entre las tareas como la forma principal de organizar la concurrencia. Finalmente, para algunos problemas, la característica principal de la concurrencia es la presencia de grupos de tareas que interactúan bien definidos, y la cuestión clave es cómo fluyen los datos entre las tareas

Para este trabajo de investigación se aplicará la estrategia de organización por tareas, ya que como se observó desde el módulo anterior existe una fuerte dependencia de las tareas entre procesos y entre los procesos dentro de las tareas identificadas en la etapa 1.

La estrategia de organización por tareas agrupa todas las tareas presentadas en la Figura 13. Cabe señalar que la primera tarea generación de la población inicial es abordada por una cantidad N de hilos; y las demás tareas necesitan llevar un orden. De manera que estas serán ejecutadas por un solo hilo a la vez, y simultáneamente existen una cantidad N de hilos que resuelven el macrogrupo de tareas.

Figura 13. Organización de las tareas para la propuesta de solución.

Etapa 3: Estructura de Soporte

La estructura de soporte es un paso intermedio entre la estructura del algoritmo y los mecanismos de implementación. Dicha etapa se divide en la estructura del programa y la estructura de datos.

En esta etapa se fundamenta cómo será la estructura del programa y la estructura de los datos, existen cuatro patrones básicos para la estructura del programa que son el SPMD, maestro/esclavo, paralelismo de ciclo y división/unión. Además existen tres patrones en la estructura de los datos los cuales son: los datos compartidos, cola compartida y datos distribuidos. Estos patrones no son mutuamente excluyentes entre sí por lo que se puede hacer una combinación entre ellos.

En este trabajo algoritmo genético está formado por diversos módulos y cada módulo hace uso de ciclos para crear llevar a cabo las tareas, por tal motivo se ha utilizado el patrón de paralelización de ciclos. Por ejemplo, en el módulo de la generación de la población inicial en el que de manera paralela se generan los individuos de la población, es necesario ordenar las soluciones para elegir las soluciones padre que se usarán en la siguiente generación, por lo que hay que unir los resultados obtenidos por las distintas unidades de procesamiento para elegir las soluciones, adaptándose al patrón de división y unión. Como se observa, se trabaja con una sola población la cual se almacena en una memoria común por lo que se hará uso del patrón de memoria compartida. Cabe señalar que en los otros módulos el comportamiento para el cálculo de cada uno de los procesos sigue el mismo patrón de comportamiento por lo tanto, los demás módulos también aplican los mismos patrones.

Módulo de la generación de la población inicial versión secuencial	Módulo de la generación de la población inicial versión paralela.
Inicio: for j=0 -> |Hijos| poblacionInicial() Fin.	Inicio: **par-for** j=0 -> |Hijos| poblacionInicial() Fin.
Módulo de operadores genéticos versión secuencial. Inicio: for i=0 -> Generaciones for j=0 -> |Hijos| seleccion() cruzamiento() mutación() ordenamiento() Fin.	Módulo de operadores genéticos versión paralela. Inicio: for i=0 -> Generaciones **par-for** j=0 -> |Hijos| seleccion() cruzamiento() mutación() ordenamiento() Fin.

Etapa 4: Mecanismo de implementación

Son conjuntos de instrucciones del procesador, comúnmente accedidas desde un lenguaje de programación de alto nivel. Estos mecanismos se dividen en el manejo de unidades de procesamiento, sincronización y comunicación.

Manejo de unidades de ejecución.

Una unidad de ejecución es una abstracción de la entidad que lleva a cabo cálculos y es manejada por el programador por medio del sistema operativo. Existen dos tipos: procesos e hilos.

Un proceso es un objeto que incluye su propia memoria, registros, archivos abiertos y todo lo necesario para definir su contexto en el sistema operativo. La creación de los procesos es computacionalmente costosa al igual que la comunicación entre ellos debido a que son conjuntos de tareas independientes entre sí.

Un proceso puede ser descompuesto en múltiples hilos que permitan realizar tareas en paralelo compartiendo los recursos del proceso común. La creación y destrucción de hilos es

menos costosa que crear un proceso, al igual que la comunicación entre hilos dentro de un mismo proceso.

Los hilos requieren relativamente pocos ciclos del procesador para ser creados, por lo que pueden ser creados y destruidos dentro de un programa según sean necesarios. Existen APIs que permiten el manejo de la creación y destrucción de hilos como lo es OpenMP, la cual se usará como mecanismo de implementación del algoritmo creando los hilos de manera implícita al usar la directiva de compilador **#pragma omp parallel**.

Sincronización

Impone un orden en los eventos que ocurren en las distintas unidades de ejecución.

Una tarea fundamental de la sincronización es la correcta lectura y escritura sobre estructuras de memoria compartidas, proporcionando mecanismos que permita hacer uso de la información del programa a múltiples hilos.

La sincronización no solo se presenta a nivel de datos, también puede ser realizada a nivel de instrucción por medio de barreras en la que los procesos o hilos esperan a que el resto llegue a un punto determinado, por ejemplo, a que todos terminen de realizar sus tareas dentro de un ciclo antes de continuar.

La sincronización es realizada por medio de OpenMP que permite el manejo implícito y explícito del manejo de memoria y de las instrucciones.

Comunicación

Los procesos tienen su propia memoria que solo puede ser manipulada por ellos mismos, para intercambiar información entre procesos se debe realizar de manera explícita por medio del pase de mensajes.

En la ejecución del algoritmo realiza por medio del manejo de hilos, por lo que se lleva a cabo comunicación explicita.

Capítulo 5: Experimentación y Resultados

En este capítulo se describen los experimentos realizados para validar el objetivo general que es resolver el problema de selección de cartera de proyectos mediante estrategias de asignación de recursos, hibridación de algoritmos, estrategias de paralelización y estrategias basadas en patrones de descomposición para algoritmos paralelos.

Aquí se diseñaron experimentos que validen la calidad de solución encontrada y el tiempo de ejecución necesario para dar un conjunto de soluciones por parte del algoritmo solucionador, cabe señalar que la calidad de la solución fue comparada contra un algoritmo exacto.

El capítulo se divide en cinco secciones principales. En la Sección 5.1 se muestran las condiciones experimentales para la ejecución del algoritmo. En la sección 5.2 se describe una instancia de ejemplo a utilizar, en la sección 5.3 se describen las medidas de calidad para la evaluación del algoritmo, en la sección 5.4 se describe la configuración inicial del algoritmo básico secuencial y en la 5.6 el conjunto de experimentos comenzando con el 5.6.1 que evalúa el algoritmo MOGA, en el 5.6.2 se calibra el algoritmo MOGA, en el 5.6.3 se compara el algoritmo MOGA secuencia contra el MOGA paralelo (MOGAP) y en la sección 5.6.4 se compara el MOGAP contra la Hibridación para la asignación de portafolios en la generación de carteras H-MOGAP.

5.1 Condiciones Experimentales

El algoritmo H-MOGAP fue programado en lenguaje C, usando el compilador GCC 4.4.7 con OpenMP 3.0 y Java con OpenJDK de 64-Bit versión 1.7.0_91 con entorno de desarrollo NetBeans 8.2, distribución Linux CentOS 6.7 de 64 Bits. Los algoritmos fueron ejecutados en una computadora con un procesador Intel® Xeon® de 10 Cores E5-2650 V3 (30MB de Cache, 2.30 GHZ y 64 GB de memoria RAM. Cada uno de los algoritmos fue ejecutado 30 veces sobre cada una de las instancias utilizadas en los distintos experimentos.

5.2 Descripción de la Instancia

Las instancias con las que se trabajaron fueron creadas por medio de un generador del grupo de trabajo y se describe el formato a continuación.

El nombre de la instancia indica las dimensiones, siguiendo el formato:

o*w***p***x***a***y***r***z*_*id*

Donde *w* es la cantidad de objetivos contemplados, *x* es la cantidad de proyectos, *y* es la cantidad de áreas, *z* la cantidad de regiones, e *id* es un consecutivo. Por ejemplo, la instancia "**o1p1000a3r2_0**" es la instancia número **0** del grupo de instancias con **1** objetivo, **1000** proyectos, **3** áreas y **2** regiones.

La instancia cuenta con los siguientes datos

Periodos	1
Proyectos	1000
Objetivos	1
Áreas	3
Regiones	2
Presupuesto	3750000
Mínimo y máximos por área	694440 2250000
	694440 2250000
	694440 2250000
Mínimo y máximos por región	1102940 3187500
	1102940 3187500
Proyecto 1	6245 3 1 9505
Proyecto 2	8325 1 2 6225
Proyecto 3	6360 1 1 5810
	…

Figura 14. Descripción de una instancia del problema de cartera de proyectos.

La Figura 14 se muestra un ejemplo de una instancia del problema de cartera de proyectos en el cual se deben planificar los recursos para **1** periodo de tiempo entre **1000** proyectos, cada uno de ellos tiene **1** objetivo además de pertenecer a 1 de las 3 áreas y a 1 de las 2 regiones con las que cuenta la cartera. El presupuesto a repartir entre los proyectos es de **3,750,000** unidades monetarias, además, cada área debe tener un presupuesto entre **694,440** y **2,250,000** unidades, y cada región entre **1,102,940** y **3,187,500** unidades. Después de los máximos y mínimos por área y región se enlista el costo, el área, la región y el beneficio de cada uno de los 1000 proyectos, teniendo el primer proyecto un costo de **6245**, perteneciendo al área número **3**, a la región **2** y teniendo un beneficio de **9505** unidades monetarias.

Para este trabajo se tienen 30 instancias de las cuales 10 son de 500 proyectos, 10 de 1000 proyectos y 10 de 1500 proyectos, las 30 instancias cuentan con 3 áreas y 2 regiones. Las instancias fueron creadas por un generador de instancias de desarrollado por Gilberto Zarate.

5.3 Medidas de calidad para la evaluación de algoritmos

Para la validación de la eficiencia se aplicarán las medidas de calidad de beneficio y error, tanto en las versiones secuenciales como en la versión paralela. Además se proponen medidas relacionadas con el tiempo para la versión paralela. Enseguida se describe cada medida.

La calidad se mide a través del beneficio, el cual se obtiene:

$$beneficio = \sum_{i}^{n} x_i c_i$$

Dónde

x es el vector de solución

c es la función objetivo

El porcentaje de error se obtiene por medio de:

$$\%Error = \left(\frac{beneficio - solución\ exacta}{solución\ exacta}\right) x100$$

Calculo de la eficiencia de un algoritmo paralelo:

$$speedup = \frac{T(1)}{T(n_{procesadores})}$$

Se espera que el *speedup* sea mayor que 1, lo que indicaría que es factible la paralelización del algoritmo.

La *eficiencia* es la relación entre el *speedup* y el número de procesadores con los que se ejecutó:

$$Eficiencia = \frac{speedup}{n}$$

en donde n es el número de procesadores.

5.4 Configuración inicial del algoritmo

La experimentación con el algoritmo genético básico secuencial conocido como MOGA dará inicio con la siguiente configuración inicial. Dicha configuración será evaluada en calidad de las soluciones para cada una de las instancias y verificar si es necesario un ajuste de parámetros.

La población empleada en cada instancia será de la misma cantidad que los proyectos, existiendo la siguiente relación. El tamaño de la población a utilizar en cada instancia es igual al número de proyectos como se puede ver en la Tabla 3.

Tabla 3. Configuración inicial del algoritmo.		
Instancia	Número de proyectos	Población
o1p500a3r2_0	500	500
o1p500a3r2_1	500	500
o1p500a3r2_2	500	500
o1p500a3r2_3	500	500
o1p500a3r2_4	500	500
o1p500a3r2_5	500	500
o1p500a3r2_6	500	500
o1p500a3r2_7	500	500
o1p500a3r2_8	500	500
o1p500a3r2_9	500	500
o1p1000a3r2_0	1,000	1,000
o1p1000a3r2_1	1,000	1,000
o1p1000a3r2_2	1,000	1,000
o1p1000a3r2_3	1,000	1,000
o1p1000a3r2_4	1,000	1,000
o1p1000a3r2_5	1,000	1,000
o1p1000a3r2_6	1,000	1,000
o1p1000a3r2_7	1,000	1,000
o1p1000a3r2_8	1,000	1,000
o1p1000a3r2_9	1,000	1,000
o1p1500a3r2_0	1,500	1,500
o1p1500a3r2_1	1,500	1,500
o1p1500a3r2_2	1,500	1,500
o1p1500a3r2_3	1,500	1,500
o1p1500a3r2_4	1,500	1,500
o1p1500a3r2_5	1,500	1,500
o1p1500a3r2_6	1,500	1,500
o1p1500a3r2_7	1,500	1,500
o1p1500a3r2_8	1,500	1,500
o1p1500a3r2_9	1,500	1,500

5.5 Experimentaciones

En esta sección se muestra un conjunto de experimentos que validan los resultados obtenidos por el algoritmo genético propuesto llamado MOGA.

5.5.1 Experimentación 1: Evaluación del MOGA secuencial

Como primera actividad se desarrolló un algoritmo genético secuencial básico conocido como MOGA, dicho algoritmo soluciona el problema de cartera de proyectos. A través de este algoritmo se busca tener una solución base para posteriormente desarrollar la versión paralela de dicho algoritmo.

En esta sección se evaluará el porcentaje de error promedio de las soluciones generadas por el MOGA. Las instancias usadas en la evaluación son las descritas en la Tabla 3. Cada instancia se ejecuto 30 veces y su promedio es comparado con la solución exacta.

La configuración para el experimento fue de 500 generaciones y 10% de porcentaje de mutación, los resultados se expresan en porcentaje de error en la Tabla 4 para las instancias de 500 proyectos, en la Tabla 5 para las instancias de 1000 y en la Tabla 6 para las instancias de 1500 proyectos.

Tabla 4. Resultados de la Ejecución del MOGA para 500 proyectos.

Instancia	Solución promedio	Solución exacta	Error promedio
o1p500a3r2_0.txt	1,606,303.67	2,059,455	22.00%
o1p500a3r2_1.txt	1,638,133.17	2,081,040	21.28%
o1p500a3r2_2.txt	1,570,294.83	2,051,825	23.47%
o1p500a3r2_3.txt	1,643,028.83	2,078,045	20.93%
o1p500a3r2_4.txt	1,605,124.00	2,082,975	22.94%
o1p500a3r2_5.txt	1,599,646.50	2,054,885	22.15%
o1p500a3r2_6.txt	1,624,573.50	2,091,650	22.33%
o1p500a3r2_7.txt	1,603,178.00	2,079,160	22.89%
o1p500a3r2_8.txt	1,605,650.00	2,042,535	21.39%
o1p500a3r2_9.txt	1,571,082.50	2,021,795	22.29%
		Promedio:	22.17%

Tabla 5. Resultados de la Ejecución del MOGA para 1000 proyectos.

Instancia	Solución promedio	Solución exacta	Error promedio
o1p1000a3r2_0.txt	3,182,233.17 $	4,112,060 $	22.61%
o1p1000a3r2_1.txt	3,118,113.50 $	4,094,045 $	23.84%
o1p1000a3r2_2.txt	3,206,935.00 $	4,140,325 $	22.54%
o1p1000a3r2_3.txt	3,203,101.00 $	4,162,030 $	23.04%
o1p1000a3r2_4.txt	3,204,033.17 $	4,116,050 $	22.16%
o1p1000a3r2_5.txt	3,177,032.00 $	4,114,440 $	22.78%
o1p1000a3r2_6.txt	3,196,430.67 $	4,078,920 $	21.64%
o1p1000a3r2_7.txt	3,206,218.83 $	4,120,965 $	22.20%

o1p1000a3r2_8.txt	3,158,842.17 $	4,076,790 $	22.52%
o1p1000a3r2_9.txt	3,188,642.50 $	4,115,990 $	22.53%
		Promedio:	22.59%

Tabla 6. Resultados de la Ejecución del MOGA para 1500 proyectos.

Instancia	Solución promedio	Solución exacta	Error promedio
o1p1500a3r2_0.txt	4,799,361.00 $	6,191,650 $	22.49%
o1p1500a3r2_1.txt	4,751,094.33 $	6,118,415 $	22.35%
o1p1500a3r2_2.txt	4,853,351.00 $	6,266,060 $	22.55%
o1p1500a3r2_3.txt	4,856,571.50 $	6,221,460 $	21.94%
o1p1500a3r2_4.txt	4,779,386.67 $	6,197,960 $	22.89%
o1p1500a3r2_5.txt	4,786,349.67 $	6,148,315 $	22.15%
o1p1500a3r2_6.txt	4,750,921.33 $	6,192,275 $	23.28%
o1p1500a3r2_7.txt	4,741,107.33 $	6,173,890 $	23.21%
o1p1500a3r2_8.txt	4,833,791.00 $	6,197,220 $	22.00%
o1p1500a3r2_9.txt	4,801,740.50 $	6,142,305 $	21.83%
		Promedio:	22.47%

Como se observa en las Tablas 4, 5 y 6 los tres conjuntos de instancias después de haberlas ejecutado 30 veces tienen un error promedio de 22% comparado con su solución exacta. Después haber obtenido estos resultados se busca mejorar la calidad del algoritmo. En el siguiente experimento se calibraron algunos de los parámetros del algoritmo con la finalidad de reducir el porcentaje de error.

5.5.2 Experimentación 2: Calibración del MOGA secuencial por medio del porcentaje de mutación y número de generaciones

En este experimento se trabajá con el MOGA básico secuencial y se seleccionarán algunos factores a los cuales se les asignarán diversos niveles de valores con el objetivo de reducir el porcentaje de error promedio de las soluciones generadas por el MOGA secuencial atreves del aumento de los porcentajes de mutación y número de generaciones.

Después de las pruebas realizadas a la implementación del MOGA secuencial se observa que se puede mejorar la calidad de las soluciones mediante una calibración de parámetros, con el objetivo de mejorar la calidad de las soluciones, para esto se experimentará con diversos valores que evalúen el algoritmo con diversas configuraciones usando diferentes combinaciones de porcentaje de mutación y número de generaciones.

Al igual que en la Experimentación 1, las instancias utilizadas para la evaluación son las mostradas en la Tabla 3, siendo evaluada cada instancia 30 veces y su promedio es

comparado con la solución exacta. Las configuraciones para el experimento se muestran en la Tabla 7.

Tabla 7. Configuraciones utilizadas en el algoritmo básico secuencial.

Configuración	Generaciones	Porcentaje de mutación
1	100	100%
2	200	100%
3	300	100%
4	400	100%
5	500	100%
6	100	75%
7	200	75%
8	300	75%
9	400	75%
10	500	75%
11	100	50%
12	200	50%
13	300	50%
14	400	50%
15	500	50%
16	100	25%
17	200	25%
18	300	25%
19	400	25%
20	500	25%

Los resultados de la experimentación 2 obtenidos para las instancias de 500 proyectos se muestran en la Tabla 8, los resultados de las instancias de 1000 proyectos en la Tabla 9 y los resultados de las instancias de 1500 proyectos en la Tabla 10.

Tabla 8. Resultados del algoritmo básico secuencial en instancias de 500 proyectos.

Número de Configuración	Generaciones	Porcentaje de mutación	Población	Tiempo Promedio	Error Promedio
1	100	100%	500	462.73 ms	13.23%
2	200	100%	500	894.57 ms	7.23%
3	300	100%	500	1,320.47 ms	4.12%
4	400	100%	500	1,739.40 ms	2.46%
5	500	100%	500	2,153.50 ms	1.58%
6	100	75%	500	460.33 ms	14.15%

7	200	75%	500	891.70 ms	8.27%
8	300	75%	500	1,314.80 ms	5.20%
9	400	75%	500	1,730.60 ms	4.08%
10	500	75%	500	2,094.53 ms	3.98%
11	100	50%	500	430.67 ms	15.52%
12	200	50%	500	830.33 ms	11.30%
13	300	50%	500	1,181.40 ms	10.21%
14	400	50%	500	1,466.60 ms	10.19%
15	500	50%	500	1,813.43 ms	9.98%
16	100	25%	500	382.93 ms	19.77%
17	200	25%	500	724.70 ms	19.57%
18	300	25%	500	1,064.43 ms	19.73%
19	400	25%	500	1,404.43 ms	19.51%
20	500	25%	500	1,741.80 ms	19.98%

Tabla 9. Resultados del algoritmo básico secuencial en instancias de 1000 proyectos.

Número de configuración	Generaciones	Porcentaje de Mutación	Población	Tiempo Promedio	Error Promedio
1	100	100%	1000	1,909.04 ms	18.63%
2	200	100%	1000	3,159.90 ms	13.47%
3	300	100%	1000	4,599.97 ms	9.56%
4	400	100%	1000	6,076.03 ms	6.83%
5	500	100%	1000	7,543.80 ms	4.90%
6	100	75%	1000	1,600.23 ms	19.18%
7	200	75%	1000	3,106.37 ms	14.44%
8	300	75%	1000	4,593.70 ms	10.67%
9	400	75%	1000	6,076.37 ms	7.89%
10	500	75%	1000	7,547.07 ms	5.93%
11	100	50%	1000	1,602.13 ms	20.08%
12	200	50%	1000	3,106.30 ms	15.79%
13	300	50%	1000	4,607.00 ms	12.53%
14	400	50%	1000	6,089.17 ms	10.31%
15	500	50%	1000	7,520.87 ms	9.42%
16	100	25%	1000	1,595.43 ms	21.47%
17	200	25%	1000	3,034.67 ms	19.53%
18	300	25%	1000	4,418.80 ms	18.99%
19	400	25%	1000	5,782.37 ms	18.83%
20	500	25%	1000	7,149.43 ms	18.85%

Tabla 10. Resultados del algoritmo básico secuencial en instancias de 1500 proyectos.					
Número de configuración	Generaciones	Porcentaje de mutación	Población	Tiempo Promedio	Error Promedio
1	100	100%	1500	4,379.70 ms	20.93%
2	200	100%	1500	8,474.07 ms	16.81%
3	300	100%	1500	11,716.43 ms	13.35%
4	400	100%	1500	13,950.20 ms	10.54%
5	500	100%	1500	17,350.33 ms	8.33%
6	100	75%	1500	3,677.33 ms	21.37%
7	200	75%	1500	7,110.07 ms	17.52%
8	300	75%	1500	10,536.17 ms	14.33%
9	400	75%	1500	13,948.03 ms	11.56%
10	500	75%	1500	17,356.07 ms	9.43%
11	100	50%	1500	3,684.13 ms	21.91%
12	200	50%	1500	7,127.73 ms	18.57%
13	300	50%	1500	10,566.90 ms	15.67%
14	400	50%	1500	14,019.40 ms	13.23%
15	500	50%	1500	17,439.90 ms	11.30%
16	100	25%	1500	3,697.67 ms	22.73%
17	200	25%	1500	7,100.47 ms	20.40%
18	300	25%	1500	10,428.90 ms	19.01%
19	400	25%	1500	13,646.90 ms	18.32%
20	500	25%	1500	16,850.20 ms	17.88%

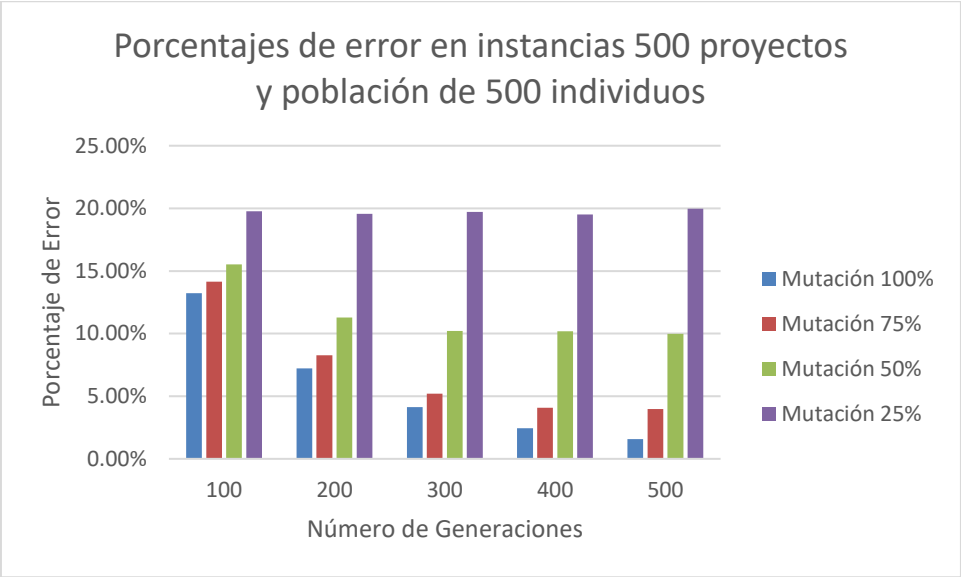
Se puede observar en la Tabla 8 los resultados de las instancias de 500 proyectos, para este conjunto de instancias la mejor calidad de las soluciones se alcanza cuando se configura el número de generaciones en 500 y el porcentaje de mutación en 100%, teniendo un error promedio de 1.58%.

Los resultados para las instancias de 1000 proyectos se observan en la Tabla 9, la mejor calidad de la solución del algoritmo se alcanza cuando se configura el número de generaciones en 500 y el porcentaje de mutación en 100%, con un error promedio del 4.90%.

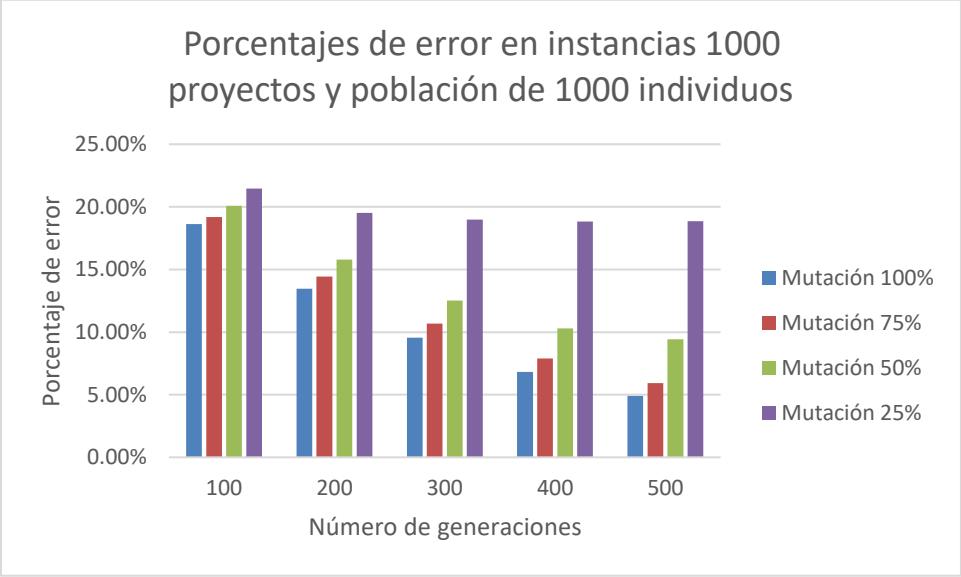
En la Tabla 10 se encuentran los resultados para las instancias de 1500 proyectos, se observa que la mejor calidad de la solución del algoritmo se alcanza cuando se configura el número de generaciones en 500 y el porcentaje de mutación en 100%, con un error promedio del 8.33%.

Concluyéndose que la mejor configuración probada para el MOGA secuencial es el número de generaciones en 500 y el porcentaje de mutación en 100%.

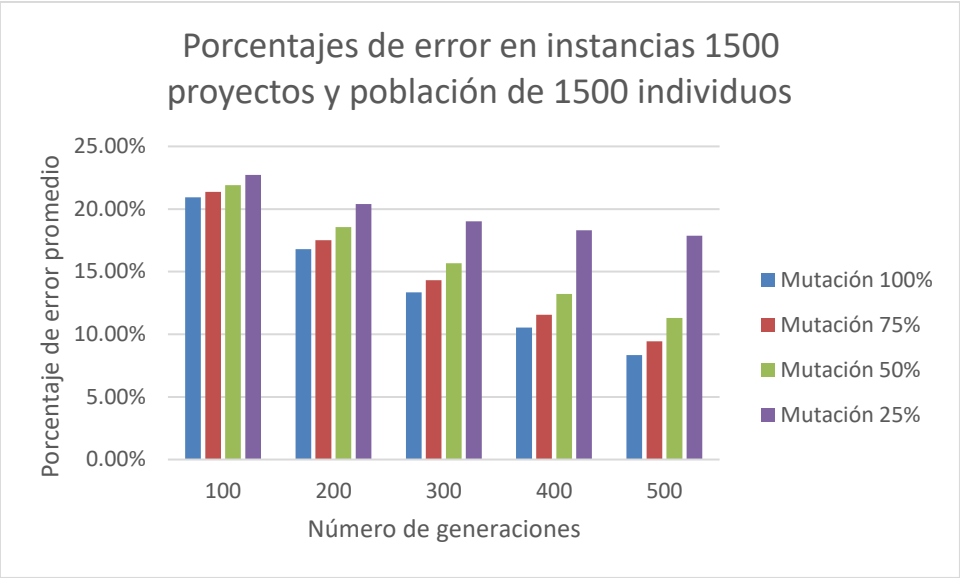
Enseguida se presentan los resultados en forma gráfica los cuales agrupan la información mostrada en las Tablas 8,9 y 10.



Gráfica 1. Porcentajes de error del algoritmo básico secuencial en instancias de 500 proyectos.



Gráfica 2. Porcentajes de error del algoritmo básico secuencial en instancias de 1000 proyectos.



Gráfica 3. Porcentajes de error del algoritmo básico secuencial en instancias de 1500 proyectos.

Como se comentó en las conclusiones de las Tablas 8, 9 y 10 en las Gráficas 1, 2 y 3 se observa claramente que el porcentaje de error aumenta con el aumento de la población y que la mejor configuración en todos los experimentos fue cuando se trabajó con 500 generaciones y 100% de mutación.

En la siguiente experimentación se hará la comparación entre el MOGA y su versión paralela llamada MOGAP.

5.5.3 Experimentación 3: Comparación del porcentaje de error y tiempos de ejecución entre MOGA y MOGAP, y el cálculo del aceleramiento y eficiencia en paralelo

Hasta este momento se cuenta con una versión secuencial del MOGA la cual fue calibrada para obtener la mejor calidad en la solución. En la siguiente etapa de la experimentación se trabajará con esta versión y se modificará para obtener una versión paralela la cual se denominará MOGAP.

Para validar la calidad del MOGAP se comparan los porcentajes de error y tiempos de ejecución entre MOGA y MOGAP, además de otras características necesarias que hay que evaluar en un algoritmo paralelo como son el aceleramiento y eficiencia.

Después de la calibración realizada a la implementación del MOGA se encontró una combinación de parámetros que mejoran la calidad de las soluciones, siendo de interés evaluar el comportamiento del algoritmo en paralelo con la finalidad de conocer si es posible reducir el porcentaje de error y el tiempo de ejecución, además del tiempo de ejecución se proponen otras métricas especiales para algoritmos paralelos como son el aceleramiento y la eficiencia del algoritmo mencionadas en la sección 5.1.

Para la evaluación del MOGAP se utilizaron las instancias de 500, 1000 y 1500 proyectos con 500 generaciones y 100% de probabilidad de mutación, ya que esta configuración fue la que dio el menor error promedio en la experimentación anterior, siendo evaluada la configuración con 1, 2, 4, 6, 8 y 10 hilos como se muestra en la Tabla 11.

Tabla 11. Configuraciones del MOGAP.

Número de configuración	Número hilos	Proyectos	Población
1	1	500	500
2	2	500	500
3	4	500	500
4	6	500	500
5	8	500	500
6	10	500	500
7	1	1000	1000
8	2	1000	1000
9	4	1000	1000
10	6	1000	1000
11	8	1000	1000
12	10	1000	1000
13	1	1500	1500
14	2	1500	1500
15	4	1500	1500
16	6	1500	1500
17	8	1500	1500
18	10	1500	1500

Tabla 12. Resultados del MOGAP.

Configuración	Proyectos	Número de hilos	Tiempo Promedio	Error Promedio	Aceleramiento	Eficiencia
1	500	1	2,153.50 ms	1.58%	100.00%	100.00%
2	500	2	1,214.07 ms	1.01%	177.38%	88.69%
3	500	4	683.30 ms	0.47%	315.16%	78.79%
4	500	6	546.73 ms	0.30%	393.89%	65.65%
5	500	8	452.00 ms	0.22%	476.44%	59.55%
6	500	10	440.57 ms	0.22%	488.80%	48.88%
7	1000	1	7,543.80 ms	4.90%	100.00%	100.00%
8	1000	2	5,010.03 ms	3.47%	150.57%	75.29%
9	1000	4	2,802.83 ms	1.67%	269.15%	67.29%
10	1000	6	2,237.20 ms	0.95%	337.20%	56.20%
11	1000	8	1,862.10 ms	0.62%	405.12%	50.64%

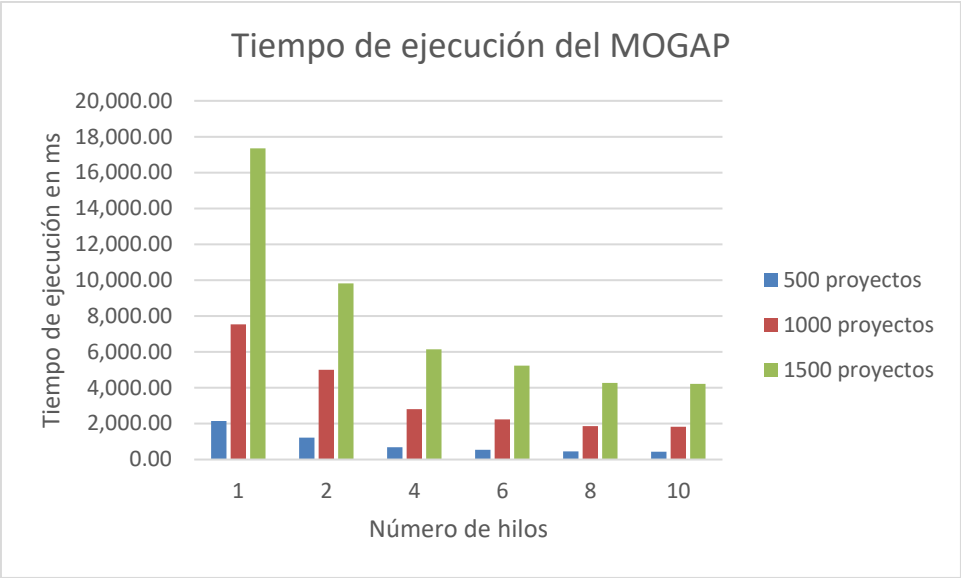
12	1000	10	1,828.53 ms	0.62%	412.56%	41.26%
13	1500	1	17,350.33 ms	8.33%	100.00%	100.00%
14	1500	2	9,814.43 ms	6.34%	176.78%	88.39%
15	1500	4	6,150.13 ms	3.42%	282.11%	70.53%
16	1500	6	5,241.83 ms	2.03%	331.00%	55.17%
17	1500	8	4,278.97 ms	1.31%	405.48%	50.68%
18	1500	10	4,225.70 ms	1.30%	410.59%	41.06%

En la Tabla 12 se presentan los resultados experimentales del MOGAP, aquí se observa que el MOGAP cumple con el objetivo al alcanzar una reducción en la calidad del algoritmo la cual es medida a través del error promedio; además se logra una reducción en los tiempos de ejecución del algoritmo.

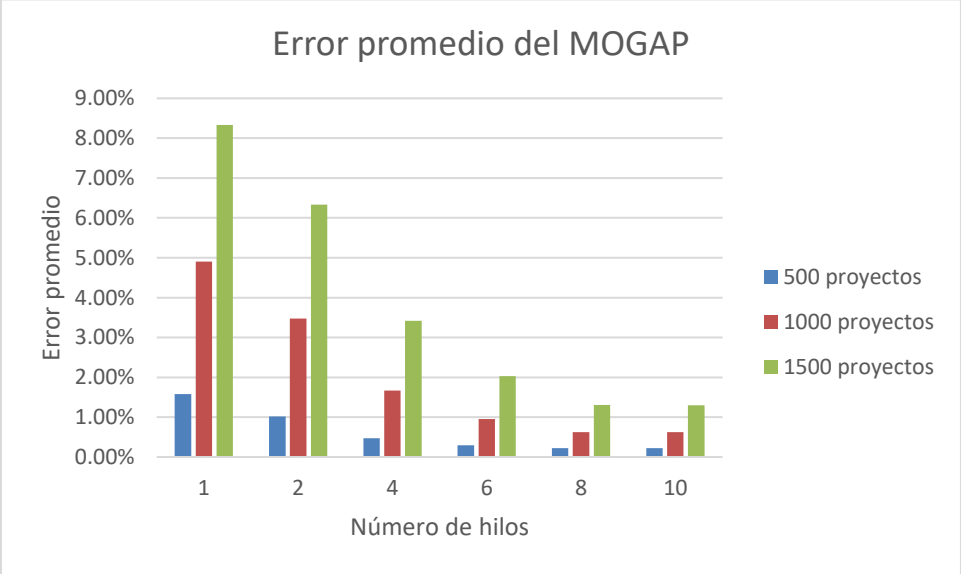
Como se puede observar en la quinta columna de la Tabla 12, el porcentaje de error se reduce desde 1.58% ejecutado con 1 hilo hasta 0.22% con 8 hilos para las instancias de 500 proyectos, también se ve la reducción en las instancias de 1000 proyectos al reducirse de 4.9% a 0.62% y en las instancias de 1500 la reducción va del 8.33% al 1.3%.

Respecto a los tiempos de ejecución, que se encuentran en la cuarta columna de la Tabla 12, se logra una reducción de hasta el 488.8% para las instancias de 500 proyectos, 412.56% para las instancias de 1000 proyectos y 410.59% para las instancias de 1500 proyectos siendo al menos 4 veces más rápida la obtención de resultados en paralelo comparado con su versión secuencial. La eficiencia decrece con respecto al aumento del número de hilos, alcanzando al menos el 50% de eficiencia con 8 hilos y decayendo por debajo de dicho porcentaje con el uso de 10 o más hilos.

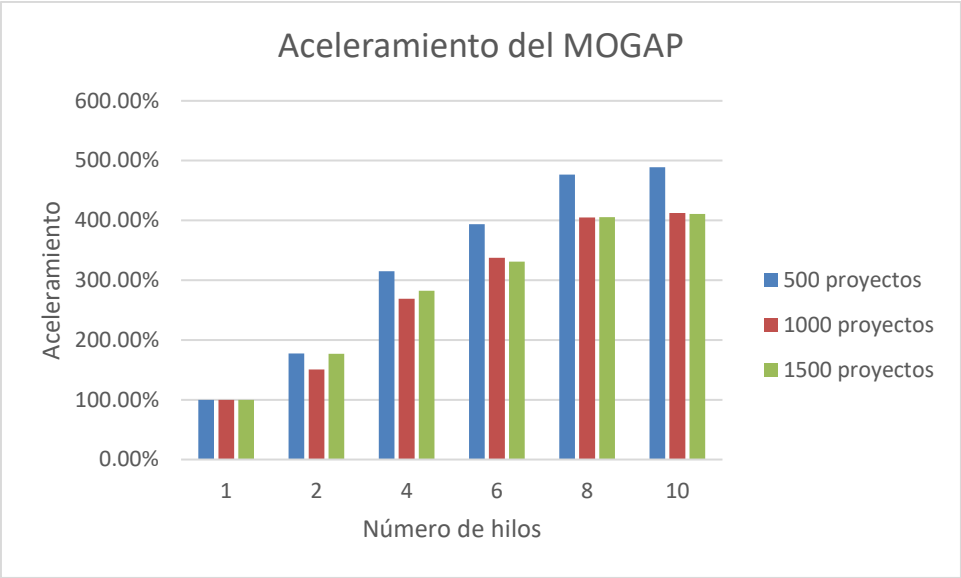
Además de las tablas se presentan los resultados en forma Gráfica donde se puede observar en la Gráfica 4 se representan los tiempos de ejecución del algoritmo, y en la Gráfica 5 el porcentaje de error, en la Gráfica 6 el aceleramiento y en la Gráfica 7 la eficiencia del algoritmo.



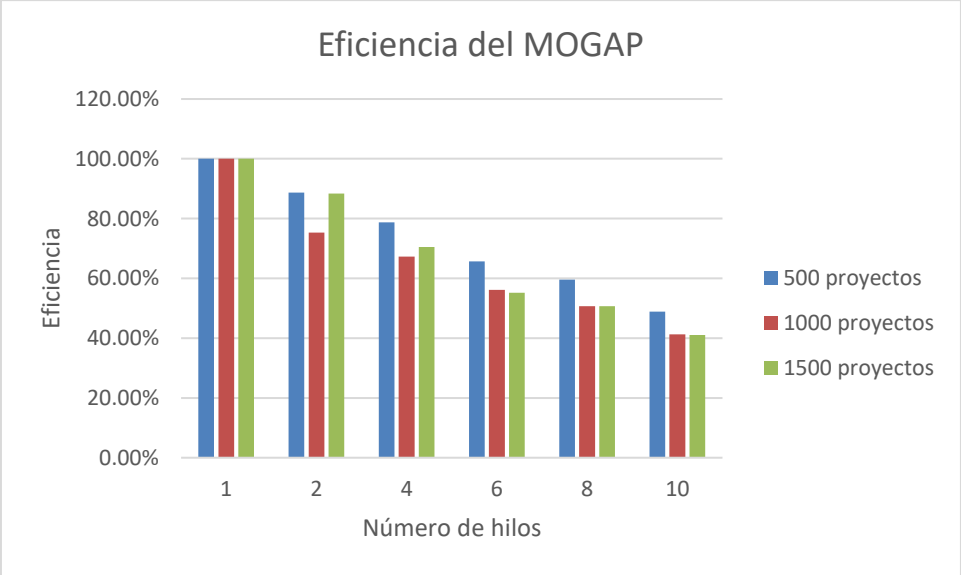
Gráfica 4. Tiempo de ejecución del MOGAP.



Gráfica 5. Error promedio del MOGAP.



Gráfica 6. Aceleramiento del MOGAP.



Gráfica 7. Eficiencia del MOGAP.

El porcentaje de error y tiempo de ejecución del algoritmo ejecutado con 8 y 10 hilos es muy similar acorde a lo reportado en la Tabla 12, sin embargo, la eficiencia del algoritmo con 10 hilos decrece alrededor de 10% en comparación con la eficiencia usando 8 hilos por lo que se realizarán futuras experimentación con 8 hilos.

En cuanto al porcentaje de error de las instancias, como se mencionó anteriormente, se logró obtener un porcentaje de error en las instancias de 500 proyectos de 0.22%, en instancias de 1000 proyectos de 0.62% y en las de 1500 proyectos se obtuvo 1.3% de error, habiendo un particular interés en poder reducir el porcentaje de error de las instancias de 1500 proyectos a un porcentaje de error menor al 1%, resultado que se espera alcanzar con

la hibridación del algoritmo con un método de programación lineal entera denominado H-MOGAP.

5.5.4 Experimentación 4: Asignación de carteras al portafolio, mediante un método de programación lineal entera para hibridar el H-MOGAP

Finalmente se trabajó una última experimentación donde se hibridizó el algoritmo MOGAP con el objetivo de mejorar la calidad de los resultados. En el experimento se evaluó el porcentaje de error promedio de las soluciones generadas por el H-MOGAP y se comparó el porcentaje de error y tiempo de ejecución contra MOGAP.

Con la finalidad de mejorar la calidad de las soluciones que genera el algoritmo MOGAP se diseñó una hibridación denominada H-MOGAP que consiste en la creación de una población elite mediante un método de programación lineal entera. Para la construcción de una solución de una instancia de *N* proyectos, se hace uso de una estrategia de covering array de *N* columnas fuerza 2 y alfabeto 2, el número de filas que tendrá el covering array depende del número de columnas, al ser el alfabeto 2 el covering array tendrá 0's y 1's, esos 0's y 1's se utilizarán para subdividir la cartera en dos subproblemas que sean más fáciles de resolver para el método de programación lineal entera.

Por ejemplo, se tiene el siguiente covering array.

Covering Array:	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**
Índice:	0	1	2	3	4	5	6	7	8	9

Con la metodología propuesta se harán dos carteras, la cartera 0 con los proyectos {0, 1, 3, 4, 5, 9} y la cartera 1 con los proyectos {2, 6, 7, 8}, el presupuesto original de la cartera es dividido sobre dos al igual que el presupuesto de cada una de las áreas y de las regiones.

Cada subcartera es solucionada de manera independiente entre sí y si ambas subcarteras generan una solución factible entonces es añadida la solución a un subconjunto de soluciones elite, el número de soluciones elite es del 10% de la población y el resto son soluciones generadas aleatoriamente que serán tratadas por el algoritmo MOGAP que en conjunto con la metodología propuesta definen el algoritmo híbrido H-MOGAP.

En la experimentación 3 se identificó el número de hilos apropiados para reducir el tiempo de ejecución y porcentaje de error de las soluciones, definiéndose el uso de 8 hilos para la evaluación del H-MOGAP y posterior comparación con el MOGAP. Además de la definición del número de hilos se extiende la evaluación ampliando el número de generaciones probadas para la comparación de los algoritmos como se muestra en la Tabla 13.

Tabla 13. Configuraciones MOGAP vs H-MOGAP con método de programación lineal entera.

Configuración	Número de generaciones	Número de hilos	Porcentaje de mutación
1	100	8	100%

2	200	8	100%
3	300	8	100%
4	400	8	100%
5	500	8	100%
6	600	8	100%
7	700	8	100%
8	800	8	100%
9	900	8	100%
10	1000	8	100%

Los resultados de esta experimentación se pueden observar en la Tabla 14.

Tabla 14. Comparación entre H-MOGAP y H-MOGAP.

Número de hilos	Generaciones	Probabilidad de mutación	Población	MOGAP		H-MOGAP	
				Tiempo Promedio	Error Promedio	Tiempo Promedio	Error Promedio
8	100	100%	500	186.53 ms	4.74%	2,429.80 ms	4.61%
8	200	100%	500	255.73 ms	1.63%	2,466.47 ms	1.59%
8	300	100%	500	327.30 ms	0.71%	2,532.67 ms	0.68%
8	400	100%	500	391.60 ms	0.38%	2,612.37 ms	0.37%
8	500	100%	500	452.00 ms	0.22%	2,595.27 ms	0.22%
8	600	100%	500	511.93 ms	0.15%	2,607.43 ms	0.14%
8	700	100%	500	574.77 ms	0.10%	2,692.57 ms	0.10%
8	800	100%	500	634.43 ms	0.07%	2,753.13 ms	0.08%
8	900	100%	500	689.77 ms	0.06%	2,798.17 ms	0.06%
8	1000	100%	500	756.27 ms	0.04%	2,872.00 ms	0.04%
8	100	100%	1000	719.60 ms	8.20%	4,790.97 ms	7.65%
8	200	100%	1000	1,043.47 ms	3.89%	5,095.00 ms	3.66%
8	300	100%	1000	1,332.67 ms	1.93%	5,394.87 ms	1.83%
8	400	100%	1000	1,609.33 ms	1.05%	5,616.63 ms	1.00%
8	500	100%	1000	1,862.10 ms	0.62%	5,867.77 ms	0.60%
8	600	100%	1000	2,084.93 ms	0.40%	6,134.17 ms	0.39%
8	700	100%	1000	2,308.00 ms	0.27%	6,371.33 ms	0.27%
8	800	100%	1000	2,564.00 ms	0.19%	6,620.80 ms	0.19%
8	900	100%	1000	2,809.47 ms	0.14%	6,860.00 ms	0.14%
8	1000	100%	1000	3,031.97 ms	0.12%	7,034.03 ms	0.11%
8	100	100%	1500	1,651.17 ms	10.47%	8,322.13 ms	9.30%
8	200	100%	1500	2,431.40 ms	6.07%	9,003.40 ms	5.72%
8	300	100%	1500	3,047.03 ms	3.51%	9,542.30 ms	3.28%
8	400	100%	1500	3,687.53 ms	2.09%	9,975.83 ms	1.96%
8	500	100%	1500	4,278.97 ms	1.31%	10,508.10 ms	1.23%
8	600	100%	1500	4,879.57 ms	0.85%	11,098.10 ms	0.81%
8	700	100%	1500	5,364.20 ms	0.58%	11,667.43 ms	0.56%
8	800	100%	1500	5,907.23 ms	0.41%	12,198.63 ms	0.40%

8	900	100%	1500	6,428.23 ms	0.30%	12,703.24 ms	0.29%
8	1000	100%	1500	6,936.00 ms	0.23%	13,231.40 ms	0.23%

En la Tabla 14 se puede observar los resultados obtenidos de la experimentación, que en general favorece al H-MOGAP con respecto al porcentaje de error, habiendo una diferencia máxima favorable de hasta 0.12% en las instancias de 500 proyectos, 0.55% para las de 1000 y 1.17% en las de 1500 en comparación con el MOGAP, sin embargo, el H-MOGAP requiere más de 2 segundos en promedio para las instancias de 500 proyectos, 4 segundos para las de 1000 y 6 segundos para las de 1500 con respecto al MOGAP.

La diferencia en tiempo de ejecución puede deberse a que el método de programación lineal entera en la población inicial es secuencial, además de ser llamado desde su API en Java, lenguaje que es interpretado a diferencia de C que es compilado impactando en el tiempo de ejecución.

Capítulo 6: Conclusiones y trabajo futuro

En este capítulo se presentan, las contribuciones científicas derivadas del trabajo de investigación presentado en este documento. De igual manera, se especifican algunas áreas de oportunidad identificadas durante el desarrollo de la investigación, las cuales podrían dar continuidad al presente proyecto.

6.1 Conclusiones

En esta tesis se cumplió el objetivo principal, en el cual se propuso resolver el problema de selección de cartera de proyectos mediante estrategias de asignación de recursos, hibridación de algoritmos, estrategias de paralelización y estrategias basadas en patrones de descomposición para algoritmos paralelos.

Para esto, se diseñó un algoritmo metaheurístico, llamado H-MOGAP el cual cumple con el objetivo logrando una mejoría en la calidad de las soluciones al reducir el error promedio al ser comparado con una estrategia secuencial.

Aportaciones principales de la investigación

1. Un algoritmo metaheurístico secuencial MOGA calibrado que resuelve el problema de selección de cartera de proyectos.

2. Un algoritmo metaheurístico paralelo MOGAP calibrado que resuelve el problema de selección de cartera de proyectos.

3. Un proceso metodológico que hace un análisis de la versión secuencial del algoritmo para llevar acabo la versión paralela del algoritmo.

4. Un conjunto de instancias creadas con un generador del grupo que sirvieron para validar los algoritmos propuestos.

5. Una estrategia de hibridación que permitió mejorar la calidad de los resultados generados por el algoritmo paralelo.

7.2 Trabajo futuro

Durante el desarrollo de este trabajo se visualizaron algunas líneas de investigación para dar continuidad al proyecto, las cuales se enumeran a continuación:

a) Probar el algoritmo desarrollado para problemas multiobjetivo, que resuelvan el problema de cartera de proyectos.
b) Buscar e incluir otras estrategias de hibridación, para mejorar la calidad de los resultados.
c) Probar con otras estrategias algorítmicas poblacionales que den solución al problema de estudio.
d) Realizar pruebas estadísticas para validar la calidad de los resultados obtenidos.

Bibliografía

[1] M. Nikkhahnasab y A. A. Najafi, «Project Portfolio Selection with the Maximization of Net Present Value,» *Journal of Optimization in Industrial Engineering,* nº 12, pp. 85-92, 2013.

[2] R. M. Cunquero, «Algoritmos Heurísticos en Optimización Combinatoria,» Universidad de Valencia, Facultad de Ciencias Matemáticas, Valencia, 2003.

[3] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Hoboken, New Jersey, USA.: John Wiley & Sons, Inc., 2005.

[4] J. A. Cerecedo Cordoba, Algoritmo de Procesamiento Celular para Solución del Problema de Cartera de Proyectos.

[5] [En línea]. Available: https://iupsm.files.wordpress.com/2010/04/optimizacion.pdf. [Último acceso: 19 5 2016].

[6] «Wikipedia,» [En línea]. Available: Optimización combinatoria. [Último acceso: 19 5 2016].

[7] M. Nikkhahnasab y A. A. Najafi , «Project Portfolio Selection with the Maximization of Net Present Value,» *Journal of Optimization in Industrial Engineering,* pp. 85-92, 2013.

[8] A. Darmann, U. Pferschy y J. Schauer , «Resource Allocation with Time Intervals».

[9] A. Schrijver, Theory of linear and integer programming, John Wiley & Sons., 1998.

[10] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, John Wiley & Sons, 2001.

[11] L. Chi-Ming y G. Mitsuo, «Multiobjective resource allocation problem by multistage decision - based hybridgenetic algorithm,» *Applied Mathematics and Computation,* 2007.

[12] Y. C. Hou y Y. H. Chang, «A new efficient encoding mode of genetic algorithms for the generalized plant allocation problem,» *Journal of Information Science and Engineering,* vol. 20, p. 1019–1034., 2004.

[13] H. Luss y S. K. Gupta, «Allocation of effort resource among competing activities,» *Operations Research,* vol. 23, nº 2, p. 360–366, 1975.

[14] Y. S. Dai, M. Xie, K. L. Poh y B. Yang, «Optimal testing-resource allocation with genetic algorithm for modular software systems,» *Journal of Systems and Software*, vol. 66, nº 1, p. 47–55, 2003.

[15] M. Johannesson y M. C. Weinstein, «On the decision rules of cost-effectiveness analysis,» *Journal of Health Economics*, vol. 12, nº 4, p. 459–467, 1993.

[16] A. Ernst, H. Jiang y M. Krishnamoorthy, «Mathematical programming approaches for solving task allocation problems,» de *Proceedings of the 16th National Conference of Australian Society of Operations Research*, McLaren Vale, 2001.

[17] G. Gamrath, Generic Branch-Cut-and-Price, Technische Universität Berlin, 2010.

[18] G. Dantzig, Linear Programming Under Uncertainty, Management Science, 1955.

[19] G. Polya, How to solve it?, Priceton University Press, 1957.

[20] H. Zanakis, J. Stelios y A. Evans, Heuristic optimization: Why, when and how to use it..

[21] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers and Operations Research, 1986.

[22] A. Duarte Muñoz, J. J. Pantrigo Fernández y M. Gallego Carrillo, Metaheurísticas, Madrid: Universidad Rey Juan Carlos.

[23] J. P. Kelly y I. H. Osman, «Meta-Heuristics: An Overview,» de *Meta-Heuristics: Theory & Applications*, Norwell, Massachusetts, USA, Kluwer Academic Publishers, 1996, pp. 1-21.

[24] Goldberg y D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Boston, USA.: Addison-Wesley Longman Publishing Co., Inc., 1989.

[25] E.-G. Talbi, Metaheuristics: From Design to Implementation, New Jersey: Wiley, 2009.

[26] H. Alfonso, C. Salto, G. Minetti, N. Stark, C. Bermúdez, A. Orellana y P. Graglia, «Algoritmos Metaheurísticos para Optimización y Aplicación a Problemas NP Completos,» La Pampa.

[27] T. G. Crainic y G. Laporte, Fleet management and logidtics, pag. 205-251., Springer, 1998.

[28] J. A. Guerra Sánchez, «Concepto de optimización de recursos,» Gestiopolis, 24 6 2015. [En línea]. Available: http://www.gestiopolis.com/concepto-de-optimizacion-de-recursos/. [Último acceso: 19 5 2016].

[29] C. I. Fábián y Z. Szoke, «Solving Two-Stage Stochastic Programming Problems with Level Decomposition».

[30] R. Contreras Vásquez, Modelo de Optimización Estocástico Multietapa del Sistema Hidrotérmico de el Salvador, Universidad de el Salvador, 2012.

[31] A. Ramos y S. Cerisola, OPTIMIZACIÓN ESTOCÁSTICA, Madrid: Universidad Pontificia Comillas, 2016.

[32] A. De la Torre De la Fuente, Algoritmos geneticos paralelos, 2005.

[33] C. A. Coello Coello, Introduccion a la Computación Evolutiva, México, D.F., 2015.

[34] F. Gruau, Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm., Lyon, France: PhD thesis, Ecole Normale Supérieure de Lyon, 1994.

[35] J. Aguilar y E. Leiss, Introducción a la Computación Paralela, Merida, Venezuela, 2004.

[36] OpenMP, «OpenMP 4.0 API C/C++ Syntax Quick Reference Car,» 2013.

[37] D. Guerrero Martínez y S. Rodríguez Lumley, «Programación Distribuida y Paralela,» Universidad de Granada, 2010. [En línea]. Available: http://lsi.ugr.es/jmantas/pdp/ayuda/mpi_ayuda.php. [Último acceso: 09 06 2016].

[38] C. G. Gómez Santillán, *Afinación Estática Global de Redes Complejas y Control Dinámico Local de la Función Tiempo de Vida en el Problema de Direccionamiento de Consultas Semánticas,* Altamira, México: Instituo Politécnico Nacional: Centro de Investigación en Ciencia Aplicada y Tecnología Aplicada, Unidad Altamira, 2009.

[39] M. B. The Problem of Tuning Metaheuristics as seen from a machine, Bruxelles: Universidad libre de Bruxelles, 2004.

[40] Á. E. E. R. H. y Z. M. , «Parameter Control in Evolutionary Algorithms,» *IEEE Transactions on Evolutionary,* vol. 3, pp. 124-141, 1999.

[41] Z. . M. y D. F. , How to Solve It: Modern Heuristics, Springer, 2nd ed., 554 p, 2004.

[42] I. Foster, Designing and Building Parallel Programs, Addison-Wesley, 2003.

[43] N. M. Arratia Martínez y F. Lopez-Irarragorri, «MILP for selecting portfolio of R&D projects in public organizations with partial and full resource allocation policies.,» *In Fourth International Workshop on Knowledge Discovery, Knowledge Management and Decision Support. Atlantis Press.*, 2013.

[44] E. Fernández-González, I. Vega-López y J. Navarro-Castillo, «Public Portfolio Selection Combining Genetic Algorithms and Mathematical Decision Analysis,» de *Bio-Inspired Computational Algorithms and Their Applications*, INTECH, 2012, pp. 139-160.

[45] N. M. Arratia Martínez, F. López Irarragorri, S. E. Schaeffer y L. Cruz-Reyes, «Static R&D project portfolio selection in public organizations,» *Decision Support Systems*, vol. 84, pp. 53-63, 2016.

[46] G. Rivera Zárate, Solución A Gran Escala Del Problema De Cartera De Proyectos Caracterizados Con Múltiples Criterios, Tijuana, Baja California, México.: Instituto Tecnológico de Tijuana. División de estudios de posgrado e investigación., 2015.

[47] H. Rastegar y M. Rasti-Barzoki, «Multi-criteria approach to project portfolio selection considering structural hardness and correlations between projects,» *Journal of Industrial and Systems Engineering*, vol. 10, nº 4, pp. 141-157, 2017.

[48] A. Shaheen y A. Sleit, «Comparing between different approaches to solve the 0/1 Knapsack problem,» *International Journal of Computer Science and Network Security*, vol. 16, nº 7, pp. 1-10, 2016.

[49] B. de Almeida Dantas y E. N. Cáceres, «Sequential and Parallel Implementation of GRASP for the 0-1 Multidimensional Knapsack Problem,» *Procedia Computer Science*, vol. 51, nº 2739-2743, pp. 2739-2743, 2015.

[50] P. C. Chu y J. E. Beasley, «A genetic algorithm for the multidimensional knapsack problem,» *Journal of Heuristics*, vol. 4, nº 1, p. 63–86, 1998.

[51] J. Deane y A. Agarwal, «Neural, genetic, and neurogenetic approaches for solving the 0-1 multidimensional knapsack problem,» *International Journal of Management & Information Systems*, vol. 17, nº 1, pp. 43-54, 2013.

[52] H. Fingler, E. N. Cáceres, H. Mongelli y S. W. Song, «A CUDA based solution to the multidimensional knapsack problem using the ant colony optimization,» *International Conference on Computational Science*, vol. 29, p. 84 – 94, 2014.

[53] B. De Almeida Dantas y E. N. Cáceres, «A Parallelization of a Simulated Annealing Approach for 0-1 Multidimensional Knapsack Problem Using GPGPU,» *28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, vol. 28, pp. 134-140, 2016.

[54] D. Soto, W. Soto y Y. Pinzón, «A parallel nash genetic algorithm for the 3d orthogonal knapsack problem.,» *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 4, nº 3, pp. 2-10, 2013.

[55] V. P. Mercado y A. D. Villagra, «Hibridación de Metaheurísticas aplicadas al Problema de Ruteo de Vehículos,» Caleta Olivia, 2013.

[56] R. Bianchini y C. Brown, Parallel genetic algorithms on distributedmemory architectures, Transputer Research and Applications.

[57] R. Benayoun, J. De Montgolfier, J. Tergny y O. Laritchev, «Linear programming with multiple objective functions: Step method (STEM),» *Mathematical programming*, pp. 366-375, 1971.

[58] C. A. C. Coello, D. A. Van Veldhuizen y G. B. Lamont, Evolutionary algorithms for solving multi-objective problems., New York: Kluwer Academic, 2002.

[59] R. S. Garfinkel y G. L. Nemhauser, Integer programming, NewYork: Wiley, 1972.

[60] R. M. Karp, «Reducibility Among Combinatorial Problems,» *Complexity of Computer Computations*, pp. 85-103, 1972.

[61] S. Cerisola, A. Ramos y A. Baíllo, «Modelado de algoritmos de descomposición con GAMS,» Escuela Técnica Superior de Ingeniería , 2004.

[62] Decide, «http://www.decidesoluciones.es/,» 29 5 2014. [En línea]. Available: http://www.decidesoluciones.es/tecnicas-de-descomposicion-i-introduccion/. [Último acceso: 19 5 2016].

[63] P. A. Jensen, «Resource Allocation Problem,» http://www.me.utexas.edu/, 2004. [En línea]. Available: http://www.me.utexas.edu/~jensen/ORMM/models/unit/linear/subunits/resource_allocation/. [Último acceso: 19 5 2015].

[64] F. Pérez García, J. Molina Luque, R. Caballero Fernández, C. A. Coello Coello y A. G. Hernández-Díaz, «Hibridación de métodos exactos y heurísticos para el problema multiobjetivo,» *Journal Economic Literature: XV Jornadas de ASEPUMA y III Encuentro Internacional.*, 2007.

[65] V. A. Serrano Hernández, Métodos para reducir evaluaciones en algoritmos evolutivos multi-objetivo, basados en aproximación de funciones., Centro de Investigación y de Estudios avanzados del Instituto Politécnico Nacional, Departamento de Ingeniería Eléctrica sección de Computación., 2007.

[66] L. Santana, Un Algoritmo Basado en Evolución Diferencial para Resolver Problemas Multiobjetivo, Centro de Investigación y de Estudios avanzados del

Instituto Politécnico Nacional, Departamento de Ingeniería Eléctrica sección de Computación, 2004.

[67] F. Y. Edgeworth, Mathematical Psychics, Londres, Inglaterra: P. Keagan, 1881.

[68] V. Pareto, Cours d'economie politique, volume 1 & 2, Lausanne, 1896.

[69] Y. Huang y Q. P. Zheng, «Benders Decomposition,» Department of Industrial and Management Systems Engineering West Virginia University, Virginia, EUA.

[70] Y. Li, Decision Making Under Uncertainty in Power System Using Benders Decomposition, ProQuest, 2008.