Algoritmo Metaheurístico basado en el enfoque MOS para resolver el problema 0-1 de la mochila cuadrática en instancias de alta dimensionalidad

Orlando Paz Duarte
Daniel Felipe Cepeda

Director: PhD. Carlos Alberto Cobos Lozada
Codirectora: PhD. Martha Eliana Mendoza Becerra

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Grupo de I+D en Tecnologías de la Información
Línea Investigación: Sistemas Inteligentes
Popayán, marzo de 2018

TABLA DE CONTENIDO

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE ECUACIONES

LISTA DE TABLAS

1 INTRODUCCIÓN

1.1 Planteamiento del Problema

El problema 0-1 de la mochila cuadrática (Quadratic Knapsack Problem, QKP) es un problema de optimización combinatoria que consiste en maximizar el beneficio de ingresar un conjunto de objetos en una mochila, restricto a la capacidad de ésta, es decir, el peso total que es capaz de llevar, donde cada uno de los elementos a incluir en la mochila cuenta con un peso y un beneficio dado. Además, cada objeto tiene asociado un beneficio respecto a cada uno de los otros objetos que se incluyen en la mochila, independiente del beneficio propio del objeto, el cual también se suma al beneficio total. Entre las aplicaciones del 0-1 QKP está el tráfico global entre estaciones satelitales, la localización de aeropuertos, terminales o estaciones de ferrocarril, el diseño de compiladores, la gestión de redes VLCI, el problema del clique, entre otros [1].

0-1 QKP cuenta con una definición matemática sencilla, pero el planteamiento de un problema 0-1 QKP puede ser sumamente complejo de resolver (Problema NP-Hard). Debido a la variedad de aplicaciones que puede tener en el mundo real, en las últimas décadas se han intensificado las investigaciones sobre el mismo, dado que, para instancias de tamaños considerablemente grandes, el espacio de soluciones crece de forma exponencial y encontrar soluciones óptimas se hace muy costoso computacionalmente y en muchos casos las técnicas no pueden encontrar dichas soluciones. En la literatura se categorizan tres tipos de enfoques o técnicas utilizadas para resolver este problema, a saber:

- Algoritmos Exactos: Ramificación y poda (branch and bound) [1], Relajación de Lagrange [2, 3], Descomposición de Lagrange [4, 5], linealización [6], Reformulación [3] y Reducción Agresiva [7], entre los principales.
- Algoritmos de Aproximación: series de aristas paralelas [8].
- Algoritmo Heurísticos y MetaHeurísticos: heurísticas voraces (greedy) [9, 10], algoritmos genéticos híbridos [10, 11], algoritmos basados en enjambres de partículas [12, 13], GRASP+Tabú [14, 15], algoritmos cuánticos [16, 17] y esquemas de búsqueda de vecindad variable (VNS) [18].

Los mejores resultados se han logrado mediante la implementación de heurísticas y metaheurísticas, donde se registran instancias resueltas eficientemente con tamaño de hasta 2000 elementos, como también se alcanza el máximo valor conocido y tiempos de ejecución superiores en las instancias de referencia (benchmark instances) presentadas en [5], con tamaños de 20, 50, 100 y 200

elementos con diferentes densidades (porcentaje de valores no ceros en la matriz de beneficio) de 25%, 50%, 75% y 100%.

Entre los primeros trabajos se encuentran: en 2005, Julstrom [10], propone un híbrido entre un algoritmo genético y una heurística voraz (greedy), siendo este inicialmente el estado del arte metaheurístico, superado en 2007 por Xie y Liu [12], quienes presentaron el mini-swarm donde se mejoran significativamente las métricas de tiempo de ejecución y tasa de éxito. Después en 2009, Pulikanti y Singh [19], presentan un algoritmo de colonia de abejas artificial (ABC) mejora el algoritmo greedy [10] respecto a tiempos de ejecución y tasa de éxito. En el mismo año, Azad y otros[13], plantean un algoritmo de enjambre de peces artificial simplificado superan al mini-swarm en estas mismas métricas. Más adelante en 2013, Yang y Chu [14], presentan un algoritmo GRASP embebido con búsqueda Tabú (GRASP+Tabú) se posiciona como el estado del arte proponiendo soluciones de instancias hasta de 1000 y 2000 elementos, y solucionando las mismas instancias trabajadas en la literatura anteriormente con un menor tiempo y mayor tasa de éxito. En 2015, Patvardhan y otros [17], proponen un algoritmo evolutivo cuántico (QIEA-PSA) mejora notoriamente tiempos de ejecución y el número de evaluaciones de la función objetivo requeridas para encontrar soluciones óptimas en instancias de (100 y 200 elementos). Ese mismo año, Jarboui y otros [20], desarrollan un algoritmo SGVNS que implementó una variación del algoritmo de vecindad variable (VNS) para instancias de 1000 y 2000 elementos, mejorando algunas instancias presentadas por GRASR+Tabu. El siguiente año, en 2016, Patvardhan y otros [16], realizan una mejora del algoritmo [17] que usa computación en paralelo, mejora sus tiempos de ejecución para la solución de instancias de 100 y 200 elementos y resuelve las instancias de 1000 y 2000 elementos mejorando el tiempo de ejecución respecto al estado del arte pero con resultados desfavorables en tasa de éxito. Finalmenteen 2017, Chen y Hao [15], presentan un algoritmo de exploración de hiperplanos IHEA se posiciona como la mejor solución conocida para el 0-1 QKP ya que reporta los mejores resultados para los grupos de instancias de 100, 200, 1000 y 2000 elementos, superando considerablemente los tiempos de todos los algoritmos mencionados previamente, con tasas de éxito muy cercanas al 100% en todos los grupos de instancias.

Por su parte, MOS (Multiple Offspring Sampling) [21] es una metaheurística que permite realizar un trabajo sinérgico de diversas técnicas metaheurísticas subordinadas (sub-técnicas) buscando obtener los mejores beneficios de cada una de ellas. Para lograr esto, MOS gestiona dinámicamente la participación de cada una de sus sub-técnicas en un problema dado. En [21] se presenta la estructuración formal para el diseño de algoritmos basados en MOS, generalizando el tipo de técnica que se use, lo cual deja abierta la cantidad de posibles técnicas que se

pueden combinar, para mejorar el desempeño en la solución de diversos problemas de optimización. MOS fue testeado en 20 funciones continuas de alta dimensionalidad, establecidas como marco de referencia para probar el rendimiento de diversos algoritmos de optimización y se obtuvieron resultados muy competitivos y superiores en general al estado del arte.

Teniendo en cuenta las ventajas que presenta MOS y los buenos resultados obtenidos con los algoritmos del estado del arte en la solución del problema específico de 0-1 QKP, en la formulación de este proyecto se consideró útil analizar detalladamente las ventajas que otorgan estos métodos a la hora de converger a soluciones óptimas, ya que cada uno maneja un enfoque diferente de solución. Logrando así la definición de un nuevo algoritmo metaheurístico basado en MOS que usa sub-técnicas del estado del arte para resolver problemas 0-1 QKP.

Teniendo en cuenta que hasta el inicio del proyecto se contaba en el estado del arte con tan solo 2 metaheurísticas GRASP+tabu [14] y SGVNS [20] que resuelven instancias de alta dimensionalidad (1000 y 2000 objetos) de manera secuencial, en este proyecto se consideró apropiado usarlas como técnicas subordinadas del MOS [21] para resolver el problema 0-1 de la mochila cuadrática.

Con lo anteriormente mencionado, en este proyecto se planteó la siguiente pregunta de investigación: ¿Cómo se puede mejorar el desempeño (mayor tasa de éxito y menor tiempo) de dos técnicas trabajando sinérgicamente bajo un enfoque MOS para resolver problemas 0-1 QKP en instancias reconocidas por la comunidad académica y científica del área (disponibles en [14])?.

1.2 Aportes

Desde la perspectiva de investigación, las contribuciones de este proyecto se centraron en generar nuevo conocimiento, dirigido a la definición de un nuevo algoritmo para resolver el problema binario de la mochila cuadrática (0-1 Quadratic Knapsack Problem) usando un enfoque MOS (Multiple Offspring Sampling) de forma secuencial con técnicas que reportan los mejores resultados en la literatura, así como su evaluación y comparación con los métodos del estado del arte. Es de resaltar que MOS originalmente está planteado para usar técnicas subordinadas poblacionales, y las técnicas usadas en este trabajo son de estado simple, por esto, en el trabajo además se presenta una forma de diseñar algoritmos MOS con técnicas subordinadas de estado simple.

Con el desarrollo del proyecto se logró obtener un nuevo algoritmo metaheurístico basado en el enfoque MOS [21] que usa adaptaciones de los algoritmos GRASPr

[14] y SGVNS [20] como técnicas subordinadas junto con la comparación de sus resultados frente al estado del arte. Es preciso resaltar que en la literatura no se han reportado técnicas que trabajen cooperativamente bajo algún framework o gestor de técnicas como MOS, para resolver el problema del 0-1 QKP. Lo anteriormente dicho se hace teniendo en cuenta que hasta el momento no se ha encontrado en las bases de datos IEEE, ScienceDirect, Springer Link y ACM ninguna investigación que use el enfoque MOS para resolver el problema QKP secuencial en instancias de alta dimensionalidad.

Este nuevo conocimiento es de tipo exploratorio y descriptivo ya que se hizo una clasificación de las técnicas que se comportan mejor bajo este enfoque y los factores que incidieron para que los resultados obtenidos sean de buena o mala calidad con relación al tiempo de ejecución.

En cuanto a la innovación, en este proyecto se implementó un algoritmo basado en el enfoque MOS con las técnicas que mejores resultados presentaron a través de las diferentes etapas de la metodología seleccionada, que permitió resolver instancias 0-1 QKP. El algoritmo resultante está diseñado para resolver problemas reales y al estar implementado en MatLab puede ser usado por todos los usuarios de esta herramienta académica, industrial y científica.

1.3 Objetivos

A continuación, se presentan los objetivos aprobados en el anteproyecto de este trabajo de grado, por parte del Consejo de Facultad de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca.

1.3.1 Objetivo General

Proponer un algoritmo utilizando el enfoque MOS (Multiple Offspring Sampling) con dos (2) metaheurísticas subordinadas para el problema 0-1 de la mochila cuadrática en instancias de alta dimensionalidad.

1.3.2 Objetivos Específicos

- Establecer la línea base de la investigación con el modelado e implementación de un marco de prueba que incluya cuatro (4) algoritmos que resuelven 0-1 QKP e instancias de problemas reconocidos por la comunidad académica y científica.
- Modelar e implementar un algoritmo basado en MOS con dos (2) técnicas, producto de la combinación de las cuatro (4) técnicas subordinadas derivadas del estado del arte para resolver el 0-1 QKP, seleccionando la combinación que reporte el mejor rendimiento.
- Realizar un estudio comparativo entre los resultados obtenidos del algoritmo MOS propuesto, con los del estado del arte, en función de la tasa de éxito y el

tiempo de ejecución necesario para alcanzar el máximo (óptimo) conocido, respecto a la dimensión y densidad de las instancias del problema.

1.4 Resultados Obtenidos

- Monografía, el presente documento que describe la motivación del proyecto y el estado del arte del mismo, incluyendo cada una de las etapas, aportes significativos, problemas encontrados, alternativas a dichos problemas y soluciones seleccionadas. También se presenta un informe detallado del algoritmo final junto con los resultados de su evaluación y comparación. Finalmente, las conclusiones del trabajo y trabajos futuros de I+D en el área.
- Código fuente del algoritmo implementado en matlab, que utiliza las instancias de prueba generadas por [22].
- Un artículo en el que se resumen los resultados de la investigación que se espera pueda ser publicado en evento o revista internacional.

1.5 Estructura de la Monografía

A continuación, se describe de manera general el contendido y organización de la presente monografía:

CAPITULO 1: INTRODUCCIÓN

Hace referencia al presente capitulo que introduce el tema de investigación, presenta la pregunta de investigación que origino el trabajo, los aportes al problema, también los objetivos (general y específicos) definidos en el anteproyecto, un breve resumen de los resultados obtenidos y finalmente la organización de la monografía.

CAPITULO 2: CONTEXTO TEÓRICO Y ESTADO DEL ARTE

En el segundo capítulo se realiza una descripción teórica (contexto teórico) del tema de investigación (0-1 QKP), luego se describe brevemente el enfoque MOS y finalmente se presentan las investigaciones más relevantes realizadas para resolver 0-QKP (estado del arte), elaborando una categorización según el enfoque de solución y enfatizando en las técnicas metaheurísticas como mecanismo de solución.

CAPITULO 3: 3 PROCESO DE INVESTIGACIÓN REALIZADO

Este capítulo presenta una descripción del proceso de investigación realizado para desarrollar el algoritmo propuesto en este trabajo, donde se describen las etapas

más relevantes del desarrollo del mismo, las contribuciones más notables, los problemas encontrados, las alternativas planteadas y las soluciones seleccionadas

CAPITULO 4: ALGORITMO PROPUESTO

Este capítulo describe el algoritmo propuesto en este trabajo con las adaptaciones realizadas al enfoque MOS y a las técnicas subordinadas GRASPr y SGVNS, detallando cuales fueron las funcionalidades agregadas, modificadas y eliminadas de cada trabajo.

CAPITULO 6: EXPERIMENTACIÓN

En este capítulo se detalla el proceso realizado para la evaluación del algoritmo propuesto y su comparación con los algoritmos del estado del arte GRASP+tabu y IHEA. Y finalmente, se realiza un análisis de los resultados obtenidos por las técnicas en función de la tasa de éxito y el tiempo, desde diferentes perspectivas.

CAPITULO 5: CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se presentan las conclusiones obtenidas al finalizar el trabajo de grado y las recomendaciones que puedan ser tenidas en cuenta para realizar un trabajo futuro.

CAPITULO 6: BIBLIOGRAFIA

Finalmente, este capítulo contiene las referencias a los artículos, libros, páginas web y otro tipo de documentos consultados para la realización del proyecto.

2 CONTEXTO TEÓRICO Y ESTADO DEL ARTE

2.1 El problema de la mochila cuadrática

El problema 0-1 de la mochila cuadrática (0-1 QKP) es una generalización del problema de la mochila (Knapsack Problem, KP) y fue introducido por Gallo en 1980 [1]. Formalmente se puede definir de la siguiente manera: Sea N el número de artículos (objetos o elementos) a introducir en una mochila (knapsack), j un entero positivo para identificar cada uno de los artículos, w_j (entero positivo) el peso de cada artículo j y la matriz P = (p_{ij}) de valores enteros, no negativos, con dimensión de n x n, representando en p_{jj} el beneficio (valor) obtenido si el articulo j es seleccionado y $p_{ij} + p_{ji}$ el beneficio obtenido si los artículos i y j son seleccionados al tiempo. El 0-1 QKP busca crear un subconjunto de elementos cuyo peso total no exceda la capacidad de la mochila expresada por un número c entero positivo a fin de maximizar el beneficio global. En la formulación matemática se cuenta con una variable binaria x_j (o x_i) que es igual a uno (1) si el artículo j pertenece a la solución o cero (0) si no está presente en la solución. A continuación, en la Ecuación 2.1 se presenta la formulación del 0-1 QKP.

$$\text{Maximizar} \quad \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j$$

$$\text{sujeto a} \quad \sum_{j \in N} w_j x_j \leq c,$$

$$x_j \in \{0,1\} \text{ y } j \in N$$

Ecuación 2.1 *Problema 0-1 de la mochila cuadrática*

Teniendo en cuenta que $1 \leq j \leq N$, $w_j \leq c < \sum_{j \in N} w_j$ y que la matriz de beneficios es simétrica, es decir, $p_{ij} = p_{ji}$. Nota: si $w_j < 0$, se puede dar la vuelta a la variable x_j para $1 - x_j$. Si $w_j > c$ ese artículo no se tiene en cuenta en el proceso de optimización.

Ejemplo: Se cuenta con tres (N=3) elementos disponibles a ingresar en una mochila, que tiene una capacidad máxima de dos (c=2). Los pesos de cada uno de los 3 elementos a ingresas en la mochila es igual a uno (w1=w2=w3=1) y la matriz de beneficios es la siguiente:

$$P = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 1 & 10 \\ 0 & 10 & 1 \end{pmatrix}$$

Los elementos disponibles para ingresar en la mochila están representados por el conjunto S= {1, 2, 3} y el valor que cada uno de estos elementos posee como beneficio propio está representado en la diagonal de la matriz de beneficios P de esta manera entonces para el elemento 1 el beneficio es de 10, para el elemento 2 el beneficio es de 1 y para el elemento 3 el beneficio es de 1.

Si se decide ingresar a la mochila el elemento 1 del conjunto S el beneficio será de **10** pero queda un espacio de 1 en la mochila debido a que $w_1 = 1$ y c=2, ahora bien si se decide evaluar la combinación de ingresar el elemento 1 y 2 del conjunto S, el valor de beneficio será de **11** debido a que el primer elemento tiene un beneficio propio de 10, el elemento 2 un beneficio propio de 1 y la relación de estos dos elementos tiene un valor de 0 (ver posición P_{12} o P_{21} en la matriz de beneficios P), además de esto no se rompe la restricción de capacidad de la mochila ya que $w_1 + w_2 = 2$. Al evaluar la combinación de los elementos 1 y 3 se obtiene el mismo beneficio total de la combinación 1 y 2, es decir de **11**.

Al evaluar la combinación de los elementos 2 y 3 en la mochila, el beneficio total será de **12** debido a que el beneficio del segundo elemento es 1, el beneficio del tercer elemento es 1 y la suma de las relaciones entre los elementos 2 y 3 según la matriz de beneficios es 10 (ver posición P_{23} o P_{32} en P). También es una solución viable debido a que cumple la restricción de capacidad $w_2 + w_3 = 2$. Por lo tanto, la combinación de elementos 2 y 3 es la que mejor beneficio aporta en este problema con un beneficio de **12**.

Como se indicó anteriormente, el problema 0-1 QKP es una generalización del problema de la mochila (KP) con un mayor grado de dificultad en términos de la complejidad de los cálculos, pero con las mismas dificultades al momento de hallar una solución como lo es su discontinuidad, la tendencia a quedarse atrapado en óptimos locales y la poca efectividad al momento de solucionar problemas con gran número de elementos.

2.2 Enfoque MOS

MOS [21] integra diferentes técnicas metaheurísticas subordinadas y gestiona la participación de estas durante la búsqueda de soluciones para lograr obtener los mejores beneficios de cada una de ellas. MOS provee la formalización funcional necesaria para el diseño de este tipo de algoritmos y las herramientas para identificar y seleccionar la mejor configuración para el problema que se está resolviendo. La hibridación de diversos algoritmos implica los siguientes comportamientos:

- Sinergia mediante la colaboración de los diferentes algoritmos que trabajan en conjunto.
- Competencia entre las técnicas, variando su participación de las diferentes fases de la búsqueda de solución.

MOS inicialmente reparte las evaluaciones totales de la función objetivo (FES) en igual medida para todos sus algoritmos subordinados con el fin de dar igualdad de condiciones en la competencia de FES. Este número total de FES a repartir se maneja estáticamente o dinámicamente dependiendo del contexto. Durante la ejecución del algoritmo la participación de cada técnica puede variar constantemente a valores altos y bajos, pero garantizando un nivel mínimo de participación, por ejemplo si se tienen 3 algoritmos subordinados y el primero de estos tiene un nivel mínimo de participación del 20%, sin importar el poco o nulo aporte que haga nunca podrá tener un valor menor a este ver Figura 2.1, y la suma total de la participación de todos los algoritmos será igual al total de FES.

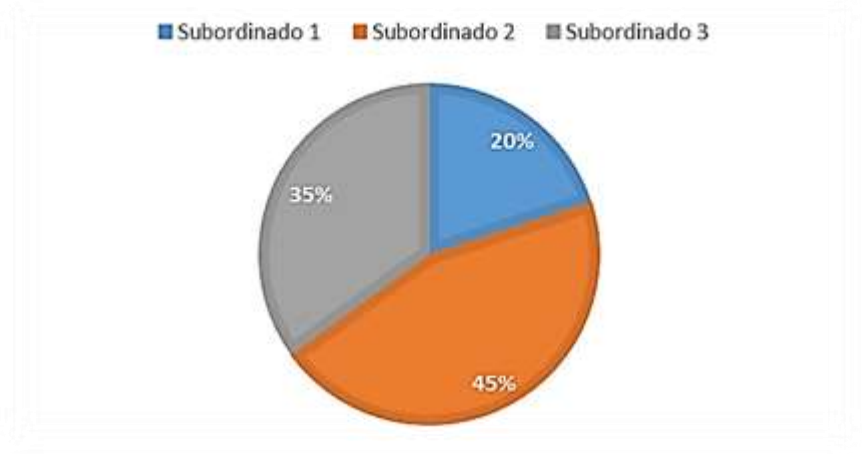

Figura 2.1 *Ejemplo de porcentaje de participación de 3 técnicas bajo MOS*

Esta participación es calculada luego de que todas las técnicas terminan su ejecución, de esta forma, al inicio de una nueva iteración MOS, cada algoritmo subordinado es penalizado o recompensado en función de su desempeño previo por medio de un ponderado, manteniendo una competición limpia y equilibrada. Este ponderado se calcula a partir del promedio de la calidad de cada individuo o individuos (dependiendo si el algoritmo es de estado simple o poblacional) generado(s) en cada técnica, y todos estos valores son usados conjuntamente en el recalculo de la participación, para que los resultados estén altamente relacionados con el desempeño.

Respecto al manejo de la descendencia, MOS necesita proporcionar al inicio de cada iteración una entrada para cada uno de sus algoritmos subordinados, entradas que convenientemente deben ser los individuos que hicieron parte de la salida de cada algoritmo, aunque en MOS no se especifica el mecanismo de selección de

soluciones ver **Figura 2.2**, se debe incorporar dicha funcionalidad para poder filtrar los mejores individuos y evolucionar en cada iteración a mejores soluciones.

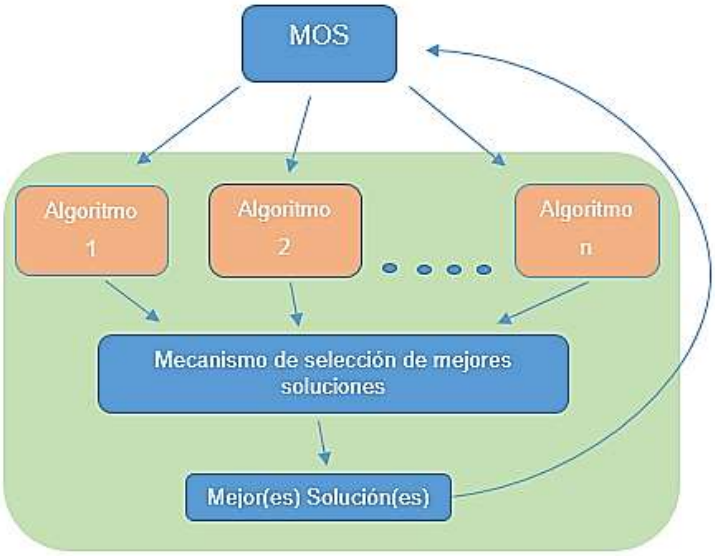

Figura 2.2 *Proceso de actualización*

2.3 Estado del Arte en 0-1 QKP

A continuación, se presentan los métodos reportados en el estado del arte para la solución de 0-1 QKP, organizados en tres grupos: métodos exactos, algoritmos de aproximación y algoritmos heurísticos y metaheurísticos.

2.3.1 Métodos Exactos

Las primeras soluciones al problema 0-1 QKP estuvieron enfocadas en la búsqueda de Upper Bounds, que son cotas superiores para la función a maximizar. El 0-1 QKP fue introducido por primera vez por Gallo, Hammer, and Simeone [5], los cuales propusieron inicialmente en 1986 la derivación de Upper Plannes (reemplazar la función objetivo por una función lineal g(x) que la domina en todos los puntos factibles), para resolverlo mediante un algoritmo de Ramificación y Poda (Branch and Bound). Luego se usaron métodos como la relajación de Lagrange [2] en 1989, la linearizacion [6] en 1996, la reformulación [3] en 1999, la descomposición de Lagrange [4, 5] en 1999 y 2004, y la programación semi-definida [23] en 2000. En 2007 [7], proponen un nuevo algoritmo exacto basado en la relajación y la descomposición de Lagrange y técnicas de reducción masiva, resolviendo instancias hasta de 1500 elementos con un desempeño considerablemente óptimos respecto a los anteriores trabajos con este tipo de métodos. Entre los mejores resultados en estos métodos se resalta el trabajo presentado en 2012 [24], que propone un método para acelerar el proceso de relajación y descomposición de Lagrange, alcanzando a resolver instancias hasta de 600 elementos en un tiempo

optimo relativo al número de elementos del problema mejorando trabajos previos con este mismo enfoque.

2.3.2 Algoritmos de Aproximación

Los algoritmos de aproximación a diferencia de los heurísticos o metaheurísticos, usualmente encuentran soluciones buenas en tiempos razonablemente rápidos, se enfocan en buscar soluciones que tengan una complejidad previamente demostrada y que tienda a ser lo más exacta posible, es decir cuyos tiempos de ejecución están acotados por cotas conocidas.

En 2002 [8], se presenta un esquema de aproximación de tiempo polinomial (FPTAS - fully polynomial time approximation scheme) para el caso en que los valores de la matriz de beneficios son todos mayores o iguales a cero y donde el grafo subyacente G=(V,E) es llamado arista serial paralela. El resultado se basa en el hecho de que es bastante fácil de desarrollar un algoritmo de programación dinámica para el caso presentado. Si el grafo subyacente es el vértice serial paralelo, el 0-1 QKP es estrictamente NP-Completo. Se obtiene un tiempo pseudo polinomial de O (N*c), donde N es el número de elementos y c la capacidad de la mochila.

2.3.3 Algoritmos Heurísticos y Metaheurísticos

Debido a las limitaciones que pueden tener los métodos anteriormente nombrados, donde el tiempo para encontrar una solución óptima puede ser muy alto, por el gran tamaño del espacio de soluciones que puede generarse para un tamaño considerable de artículos (objetos, elementos), entonces se aplican técnicas alternativas para alcanzar soluciones de calidad en un tiempo mucho menor, dando paso a varios estudios realizados con diferentes heurísticas y metaheurísticas.

2.3.4 Algoritmos basados en Población

Inicialmente en 2005 [10] se resuelven instancias del 0-1 QKP generadas aleatoriamente usando algoritmos voraces (greedy), con los cuales se logran resolver pero sin alcanzar la mayoría de los casos (tasa de éxito menor al 100%), luego se propone un algoritmo genético con una parametrización variada para resolver problemas 0-1 QKP con diferentes valores de densidad, capacidad de la mochila y número de elementos. Luego se realizó un estudio comparativo de dos heurísticos greedy, un algoritmo genético y un algoritmo genético greedy, este último obtiene los mejores resultados debido a las heurísticas utilizadas para generar soluciones iniciales, y también para luego refinar descendientes a medida que se generan estos por medio de mutación o cruce. Posteriormente, el mismo autor propone un operador de selección con permutación para dos algoritmos genéticos, que permuta números enteros para representar el orden en que los

objetos serán agregados a la solución, obteniendo mayor probabilidad de encontrar el valor óptimo con el algoritmo genético heurístico.

Un algoritmo de enjambres inteligentes llamado mini-swarm se propuso en 2007 [12], donde se tiene una población de agentes que cooperan en un entorno de intercambio social, por medio de una implementación de memoria a corto plazo que se comparte con los demás agentes con el fin común de encontrar óptimos para el 0-1 QKP. En este trabajo se logra superar considerablemente los resultados obtenidos en [5] (método exacto con ramificación y poda) y en [10] (genético greedy), resolviendo las instancias de soufit [14] de 100 y 200 elementos, en mucho menor tiempo y con mayor tasa de éxito para alcanzar el mejor óptimo conocido, desde los resultados reportados en [5] con un algoritmo de Branch and Bound.

Después, se propuso un algoritmo híbrido de colonia de abejas (ABC) en 2009 [19], donde se agrega una heurística voraz (greedy) para "reparar" las soluciones generadas por el algoritmo de colonia de abejas y posteriormente ser mejoradas por un algoritmo de búsqueda local que intercambia objetos dentro de la mochila con los que no hacen parte de la solución, hasta que se consiga un aumento en el beneficio alcanzado. Las soluciones iniciales se generan de igual forma que en [10], utilizando la densidad de cada objeto para analizar su beneficio individual respecto a una solución parcial. Los resultados se mejoran ligeramente en comparación con los presentados en [12] (mini-swarm), con una probabilidad de encontrar máximos globales del 98.7% y mejores tiempos para encontrar soluciones en cada instancia de diversas densidades.

En 2014 [13], se desarrolla una nueva versión simplificada del Algoritmo Enjambre de Peces (AFSA) llamada algoritmo binario de enjambre artificial de peces simplificado (S-bAFSA). S-bAFSA simplifica los procedimientos de selección del algoritmo para reducir requisitos de cálculo en términos de iteraciones y tiempo de ejecución para así poder llegar en menos tiempo a una solución óptima. Este algoritmo fue comparado con la versión no simplificada y se llegó a la conclusión de que para problemas con instancias más grandes el algoritmo S-bAFSA se comporta mejor. También se comparó con un algoritmo genético greedy y se llegó a la conclusión de que para instancias de 200 objetos el algoritmo S-bAFSA muestra un mejor comportamiento, mientras que para instancias de 100 objetos es menos eficiente. Respecto al mini-swarm [12], se hace una comparación muy resumida, pero se puede apreciar que no logra superar el desempeño de éste. El algoritmo reporta probabilidad de encontrar óptimos globales en el 50% de los casos con 12500 evaluaciones de la función objetivo (FES) en promedio.

El QIEA propuesto en 2015 [17] es un algoritmo de optimización global perteneciente a la familia de los algoritmos evolutivos. Se basa en la teoría de la computación cuántica, tomando principalmente conceptos como la rotación, superposición y observación de Qbits, que son la unidad mínima de procesamiento en esta área, que en este caso permiten representar soluciones de un problema de optimización específico. Haciendo una modificación a este algoritmo se obtuvo QIEA-PSA (QIEA-Patvardhan–Sulabh–Anand) que permite resolver problemas 0-1 QKP, añadiendo técnicas para la inicialización de Qbits con una mejor calidad, en el sentido de otorgar más posibilidad de agregar a la solución a aquellos objetos que sumen más beneficio respecto a una solución parcial. También se agregaron funciones para obtener una mejor solución inicial, para reparar y refinar la población generada en cada iteración al observar el conjunto de Qbits. El algoritmo se comparó con el S-bAFSA (2014) [13] y el ABC (2009) [19] mejorando drásticamente métricas para 30 ejecuciones independientes, como el tiempo promedio de ejecución, número de veces que se alcanza el máximo global (tasa de éxito) en la mayoría de las instancias de 100 y 200 elementos. También reporta la resolución de instancias de 300 elementos con densidad de hasta el 50% con valores sobresalientes en las métricas nombradas anteriormente.

Aprovechando las ventajas de la computación paralela, en 2015/2016 [16] se presenta IQIEA-P [17] un algoritmo que añade paralelismo a diversas tareas del algoritmo como la inicialización de Qbits, la reparación de soluciones después de observar el estado de los Qbits, la exploración por búsqueda local o refinamiento, entre otras fases del algoritmo que permiten ser paralelizadas. Con esta propuesta se resuelven instancias de 1000 y 2000 elementos, mejorando drásticamente los resultados en tiempo de ejecución y encontrando nuevos máximos globales para 2 instancias. El QIEA-PSA es el estado del arte para algoritmos paralelos, de los cuales es el único en su categoría resolviendo el 0-1 QKP.

2.3.5 Algoritmos de Estado Simple

En [25] se describe el Hill Climbing, un algoritmo que busca encontrar una mejor solución iterativamente cambiando uno o varios elementos de la solución. Si dicho cambio produce una mejor solución que la actual, el algoritmo procede a hacer uno o varios cambios en la nueva solución. Este proceso se repite hasta que el algoritmo no pueda encontrar mejoras. Dado que este algoritmo se usa con mucha frecuencia para encontrar óptimos locales en una estructura de vecindad y que no siempre garantiza encontrar la mejor solución posible, a este se le añaden otras características para potenciar su rendimiento en propuestas como: búsqueda local iterada, búsqueda tabú o modificaciones estocásticas. El algoritmo hill climbing [25] es seleccionado en muchos casos como la primera opción al momento de resolver problemas de optimización combinatoria debido a su relativa simplicidad.

El GRASPr introducido en [14], propone un algoritmo GRASP para resolver 0-1 QKP con un componente de memoria y reinicio. El componente de memoria permite llevar un conteo de frecuencias para identificar los elementos que han aportado más beneficio al encontrar óptimos locales, para que estos tengan mayor probabilidad de ser elegidos. El componente de reinicio consiste en reiniciar la solución con los elementos que se han encontrado en las mejores soluciones hasta el momento del inicio de cada iteración. En este trabajo se propone otro algoritmo llamado GRASP+tabu, que utiliza un algoritmo Tabú convencional, para mejorar la exploración, evitar estancamientos y evitar visitar soluciones repetidamente. El GRASPr obtiene mejores resultados que el mini-swarm [12] y el GRASP+tabu para las instancias de 100 y 200 elementos en todas las densidades, con un promedio de tiempo de 0.162 segundos para 100 iteraciones, aproximadamente la sexta parte del tiempo del mini-swarm. El GRASP+tabu es utilizado para resolver instancias de 1000 y 2000 elementos y cuatro valores de densidad, logrando resolver todas las instancias con un tiempo promedio de 299 segundos (menos de 5 minutos) y una tasa de éxito de 91%, lo cual es un gran avance por la alta dimensionalidad de las instancias.

Posteriormente, se propuso un algoritmo llamado búsqueda de vecindad variable sesgada (SGVNS) [18], que está basado en el algoritmo de búsqueda de vecindad variable (VNS). El SGVNS propone dos estructuras de vecindad, la primera se basa en el intercambio de objetos y la segunda en adición o extracción de objetos en la mochila. En este trabajo se centran en resolver instancias de 1000 y 2000 elementos presentadas en [14], y se comparan los resultados con el GRASP+tabu, superando los resultados en la mayoría de las instancias respecto a tiempo de ejecución promedio y tasa de éxito para encontrar el máximo en un determinado número de ejecuciones.

Finalmente en 2017, se propone el algoritmo denominado IHEA [15] (Iterated Hyperplane Exploration) un algoritmo de búsqueda local para resolver problemas 0-1 QKP, que busca un conjunto de hiperplanos definidos por una restricción de cardinalidad para delimitar la búsqueda en áreas prometedoras del espacio de solución y así no tener que explorar todo el espacio de solución. Para buscar soluciones de alta calidad dentro de cada sub-problema reducido, IHEA emplea un procedimiento de búsqueda tabú que permite salir de regiones no factibles para facilitar la transición entre diferentes soluciones estructuralmente viables. Otra estrategia aplicada es la perturbación que permite escapar de óptimos locales cuando no hay mejora. IHEA resuelve las instancias de alta dimensionalidad (1000 y 2000 elementos) del estado del arte con una tasa de éxito del 100% en 77 de las 80 instancias.

3 PROCESO DE INVESTIGACIÓN REALIZADO

En la siguiente sección se presenta un recuento del desarrollo del proyecto que contiene: las etapas más relevantes del desarrollo del mismo, las contribuciones más notables, los problemas encontrados, las alternativas planteadas y las soluciones seleccionadas. El proyecto se desarrolló en cuatro iteraciones que en forma incremental buscaban incrementar el conocimiento en el problema y las estrategias de solución al mismo, al punto de permitir realizar la propuesta final del algoritmo basado en MOS que fue posible obtener en el tiempo definido para el trabajo de grado.

3.1 Primera Iteración

En esta etapa del proyecto se procedió a recolectar y organizar las instancias de prueba para el problema 0-1 QKP e implementar algunos de los algoritmos que reportan los mejores resultados en el estado del arte. También se implementaron otros algoritmos que reportaban resultados prometedores para problemas binarios los cuales fueron adaptados al problema.

3.1.1 Instancias

El primer paso en el desarrollo del proyecto fue hacer una revisión en el estado de arte con el fin de conseguir las instancias de prueba. De esta revisión se obtuvo como resultado las instancias generadas por [5], disponibles en [26] para (100,200 y 300 elementos) y un algoritmo generador de instancias para el problema 0-1 QKP, construido en C++ y disponible en [22].

3.1.2 Algoritmos BBA, BDA, GREEDY, SBASFA y QIEA-PSA

El siguiente paso en esta etapa fue la implementación de dos algoritmos binarios que reportaban resultados competitivos en diversos problemas. Los algoritmos se adaptaron al problema 0-1 QKP. Por un lado, está el Binary bat algorithm (BBA) [25] que es un algoritmo poblacional basado en el comportamiento natural de los murciélagos cuando vuelan por su entorno y cazan sus presas y por otro lado el Dragonfly algorithm (BDA) [26], que también es poblacional y está basado en el comportamiento de las libélulas, cuando están en época de reproducción.

Debido a que estos dos algoritmos fueron adaptados al problema 0-1 QKP los resultados no fueron los mejores y sólo se logró obtener resultados aceptables en un 5% de las 80 instancias de prueba por lo que se optó por no continuar explorando estas dos alternativas.

Después se implementó el algoritmo genético greedy [10], este algoritmo es poblacional y aplica técnicas codiciosas para la inclusión de elementos a la mochila.

Empieza incluyendo el primer elemento a la mochila basado en el elemento de mayor valor de densidad absoluta y continúa incluyendo a la solución los elementos con respecto a los valores de densidad relativa que estos aportan, haciendo uso de un torneo binario. En este algoritmo se presentó la dificultad que en el artículo no se especifica exactamente los valores que deben tomar algunos parámetros de entrada, por lo tanto, no fue posible repetir de manera fiel los resultados de los experimentos presentados. Debido a esto se evaluaron varios valores para dichos parámetros lo cual no fue de mucha ayuda dado que los resultados tampoco fueron los mejores.

Posteriormente se implementó el algoritmo de enjambre de peces (SBASFA) [13], que simula el comportamiento de un enjambre de peces cuando realizan movimientos en busca de comida. Obteniendo resultados aceptables en instancias pequeñas (de 100 elementos), pero debido a su comportamiento altamente exploratorio y una explotación pobre no hay convergencia hacia óptimos de buena calidad en instancias de 200 y 300 elementos debido a su vasto espacio de búsqueda. Por lo que se descartó su estudio para posteriores etapas de este proyecto porque no presenta los atributos necesarios para ser considerado en un trabajo colaborativo con otras técnicas.

Luego en esta etapa se implementó el algoritmo cuántico QIEA-PSA [17], un algoritmo basado en la teoría de la computación cuántica, poblacional, con el cual se probaron inicialmente instancias de 100 y 200 elementos para corroborar los resultados del artículo, y se obtuvo un desfase considerable en los tiempos de ejecución, encontrando los máximos para la mayoría de las instancias probadas pero en tiempos superiores, atribuyendo esto a partes del código donde no se especifica bien como se hacen distintos procedimientos, posiblemente por no dar a conocer la totalidad de su funcionamiento debido a temas de propiedad intelectual o secreto empresarial. Los autores no realizan pruebas con instancias de alta dimensionalidad con este algoritmo.

En esta iteración finalmente se concluyó que para instancias de 200 y 300 elementos los algoritmos anteriormente presentados se apartan del objetivo de la propuesta, ya que no obtuvieron un buen desempeño respecto a tasa de éxito y tiempos de ejecución debido a su naturaleza poblacional, lo que implica que se hace muy costoso para ellos explorar el vasto espacio de búsqueda y necesitan de mayores tiempos de ejecución para alcanzar resultados aceptables.

3.2 Segunda Iteración

Con la experiencia de la etapa anterior, obtenida del trabajo con algoritmos poblacionales que resuelven el problema 0-1 QKP en instancias de 100 y 200 elementos, en esta iteración, se hizo mayor énfasis en algoritmos de búsqueda local para estudiar su comportamiento y viabilidad. Se propuso estudiar para esta etapa un algoritmo hill climbing altamente modular que permite combinar diferentes técnicas de mutación y reparación bajo el modelo de estado simple para poder hacer un seguimiento más detallado de la evolución de una solución en particular [25]. En esta etapa también se procedió a implementar el algoritmo MOS que luego usa diferentes combinaciones del hill climbing como técnicas subordinadas. Algoritmo Hill Climbing y MOS

Se planteó un algoritmo utilizando MOS, en nivel superior, como gestor de participación de las técnicas hill climbing, dado a la gran variedad de formas en que se pueden realizar modificaciones o mutaciones en un cromosoma o solución, y también las formas en que se puede realizar una reparación a una solución no factible. Por consiguiente, se consideró conveniente hacer una investigación del comportamiento que pueden tener todas estas combinaciones y la sinergia que puede generarse. En el estado del arte no se reporta un estudio que haya realizado combinaciones de múltiples métodos heurísticos de reparación y modificación al mismo tiempo, para el 0-1 QKP lo que aportó motivación para realizar este estudio.

Para el propósito de estas pruebas, el algoritmo MOS sirvió para calcular el comportamiento, evolución y desempeño de cada combinación de técnicas durante diferentes etapas en la ejecución del algoritmo, a partir de la calidad de soluciones generadas por cada técnica se obtiene determinada participación o puntaje para medir el número de ejecuciones que le corresponde a cada una de estas.

Para iniciar el algoritmo en un buen punto y otorgar un impulso hacia una zona con buena calidad (fitness), se usa un algoritmo que crea una solución inicial tal como en GREEDY [10], y basado en la densidad relativa se construye una solución determinista, para cada instancia, delegando la exploración y explotación a las posteriores fases del algoritmo.

En la primera versión realizada de este algoritmo se plantean como criterios para construir las técnicas, los conceptos de densidad relativa y densidad absoluta propuestos inicialmente en [10], resultando así dos métodos para realizar modificaciones y dos técnicas de reparación. En la parte de modificación el criterio de selección para extraer un elemento de la mochila era sacar el de menor densidad relativa, según la fase de ejecución del algoritmo se sacaba más de 1 elemento de la mochila. También se hicieron pruebas no solo sacando elementos en la parte de

la modificación, sino realizando inserciones y eliminando elementos que pertenecen a la solución. En la sección de la reparación por densidades también se tuvo como criterio a la hora de reparar, cuando la mochila estaba sobrecargada, sacar el menos denso (absoluto o relativo) y luego llenando cuando la solución fuera factible, llenando el espacio vacío agregando el más denso a la solución.

El principal problema de esta primera versión fue el constante estancamiento en óptimos locales, los cuales impidieron encontrar los máximos en la mayoría de los casos. De la revisión del progreso que iba obteniendo el mejor fitness o valor de función objetivo global al fin de cada iteración MOS, se pudo observar que el diseño de las técnicas de modificación y reparación tenían un comportamiento determinístico muy alto, por lo que siempre se llegaba a los mismos puntos en el escalado hacia óptimos locales, aunque se contaba con una sección de perturbación aleatoria en el algoritmo para agregar exploración, no fue suficiente para contrarrestar la tendencia hacia estancamientos producida por la falta de exploración inducida por las técnicas de densidad, por lo que se previó agregar una mayor exploración para equilibrarla con la explotación, y permitir de esta forma a las técnicas propuestas, navegar por otras zonas del espacio de búsqueda.

3.3 Tercera Iteración

Después de esto, se agregó al abanico de criterios de modificación o reparación, los conceptos de aleatoriedad, peso y valor (fitness), con el fin de lograr un mayor equilibrio entre la exploración y la explotación del espacio de soluciones. Se redefinió la forma en que funciona cada sub-algoritmo híbrido. En la Figura 3.1 se puede observar el proceso de ejecución de las fases propuestas.

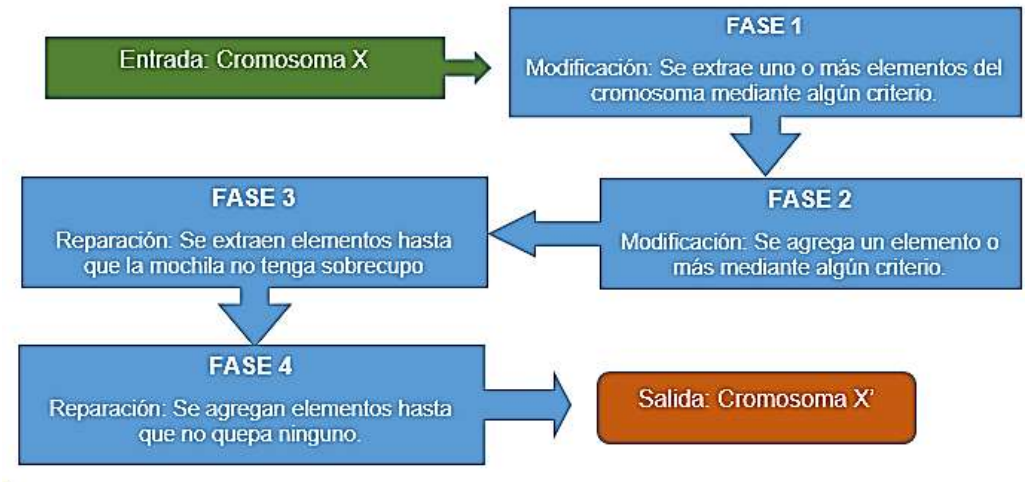

Figura 3.1 *Fases de ejecución de algoritmos Hill Climging*

Se puede observar el proceso de ejecución de cada algoritmo hibrido hill climbing bajo la gestión de MOS, donde recibe inicialmente una solución X y al final obtiene la solución X', producto de la acción de las diferentes fases que se describen a continuación:

1. Modificación (Extraer): En esta fase se hace una mutación del cromosoma, sacando un elemento de forma aleatoria, el que tenga mayor peso o el que tenga menor densidad relativa, por ejemplo.
2. Modificación (Agregar): En esta fase se hace una adición al cromosoma, agregando un elemento de forma: aleatoria, el que tenga mayor valor o el que tenga mayor densidad relativa. Sin revisar si el peso es excedido y la solución queda con sobrepeso o no factible, para permitir que elementos con estrecha relación o poco aporte con la solución actual puedan hacer parte de la solución luego de una reparación, debido a que algunas veces para saltar de un óptimo local a otro, es necesario pasar por soluciones no factibles, pero necesarias para hacer el escalamiento hacia la mejor solución.
3. Reparación (Extraer): Se prosigue a realizar una extracción de elementos de la solución actual, si la mochila se encuentra con sobrepeso, para dejar el cromosoma factible. Se hace bajo los conceptos de: Aleatorio, el que tenga menor densidad relativa o el que tenga menor densidad absoluta.
4. Reparación (Agregar): Ahora se debe llenar el espacio vacío que hay en la mochila luego de ejecutar la fase anterior, por lo que se debe agregar elementos hasta que ni uno solo de los que no pertenecen a la solución pueda ser incluido dentro de la mochila. Este procedimiento se hace bajo los conceptos de: Aleatorio, el que tenga mayor densidad absoluta o el que tenga mayor densidad relativa.

El algoritmo MOS-Hill climbing construido consta de 3 sub-algoritmos híbridos creados por la combinación de las técnicas mencionadas anteriormente, teniendo en cuenta que cada una de ellas cuente con un equilibrio entre la exploración y explotación. Por ejemplo, un algoritmo que utiliza en sus fases 1, 2, 3 y 4 los criterios de peso, valor, aleatorio y densidad relativa respectivamente, cuenta con un muy buen equilibrado de exploración y explotación en las 2 primeras fases, debido a que el peso tiene muy poca información acerca del aporte que pueda agregar un elemento a la solución pero igualmente se trata de agregar elementos con poco peso lo que indica una explotación a bajo nivel, luego en la fase 2 se agrega por valor, que posee un nivel de explotación más alto que el de peso, debido a que el fitness proporciona información más directa sobre la calidad de un elemento, después se sacan elementos en la reparación de forma aleatoria aportando un alto nivel de exploración en la tercera fase, y finalmente se termina de llenar la mochila agregando elementos con alta densidad relativa agregando el nivel de explotación

más alto en esta fase. Así mismo se evaluaron otras combinaciones como Aleatorio-aleatorio-densidad absoluta-densidad relativa y densidad relativa-densidad relativa, aleatorio-aleatorio.

Después de realizar modificaciones, combinaciones y pruebas al algoritmo MOS, se observó que se estaba gastando mucho tiempo en las evaluaciones de la función objetivo, debido a que el algoritmo estaba visitando los mismos puntos muchas veces y calculando su fitness repetidamente. La solución planteada fue poner los puntos visitados en una tabla hash y así evitar que el algoritmo volviera a pasar por esos puntos (memoria tabú). Esta modificación mejoró el desempeño, permitiendo que se alcanzaran algunos máximos, pero aumentó el tiempo en que se alcanzaban esos máximos.

En MOS, no se especifica el criterio con el que se elige la nueva entrada luego de que cada sub-algoritmo ha terminado su ejecución y debe iniciar una nueva iteración MOS con una nueva entrada para cada uno de sus subordinados, delegando esta fase para que el lector introduzca esta función. Hasta el momento se estaba eligiendo la mejor solución global encontrada por todos los subordinados, y esta era la nueva entrada en la nueva iteración. Se modificó este comportamiento y se delegó en MOS la selección de las mejores soluciones alcanzadas hasta el momento por cada uno de los subordinados, por ejemplo “*Mejores Xn*” en la Figura 3.2, son las mejores soluciones (representada por X) encontradas por la técnica n en la iteración actual. Luego al finalizar se reciben esas n soluciones (“Mejores n X”) y se distribuyen mediante un sorteo a los algoritmos subordinados en la nueva iteración, siendo n igual al número de algoritmos subordinados. Esto se introdujo debido a que cada algoritmo subordinado tiene diferentes habilidades de diversificación y explotación, permitiendo entre ellos escapar de óptimos locales, dado el caso que el comportamiento de una o más técnicas solo lleven a determinados puntos del espacio de búsqueda, bloqueando el progreso del algoritmo malgastando las evaluaciones de la función objetivo (FES) en zonas poco prometedoras.

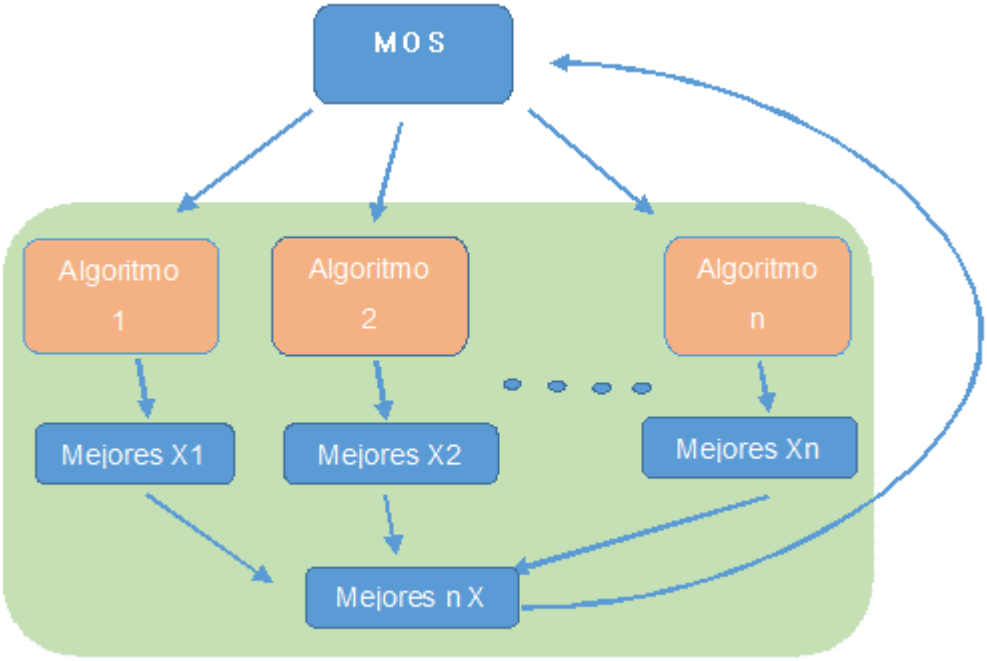

Figura 3.2 *Actualización de la nueva entrada para las técnicas subordinada*

Luego de esta serie de cambios, se mejoraron considerablemente los resultados para al menos el 50% de las instancias, principalmente por la memoria Tabú con los puntos visitados, que permitió escapar de óptimos locales difíciles de evadir. Se identificó que las técnicas que mejores resultados obtenían eran las que tenían densidad relativa en su reparación y también que en la fase de modificación funciona mejor la técnica aleatoria, por encima incluso de densidad relativa o absoluta. Se observó que la primera fase de modificación (Agregar) no tuvo mayor impacto en el desempeño del algoritmo sea cual sea el criterio de adición, e incluso en muchos casos se obtuvieron mejores resultados omitiendo esta fase, por lo que al menos 1 algoritmo de los 3 utilizados en el algoritmo MOS, no implementa ninguna acción en esta fase. Técnicas con reparación aleatoria muestran un desempeño muy pobre, quedando al final del algoritmo con un nivel de participación muy bajo, y aunque su intención es principalmente exploratoria, no cumple con el aporte necesario para que otras técnicas con mejor rendimiento, encuentren mejores soluciones.

Se logró alcanzar el máximo reportado en la literatura para el 80% de las instancias de 100 y 200 elementos en un promedio de 3 y 7 segundos respectivamente, con una tasa de éxito promedio del 70%. Los tiempos son relativamente altos respecto a los reportados en el estado del arte, debido principalmente a las restricciones de no visitar puntos repetidos, comportamiento que mejoró bastante la búsqueda de

óptimos globales, pero repercute negativamente en el tiempo de ejecución. Posteriormente se hicieron unas adecuaciones adicionales al algoritmo para probar las instancias de alta dimensionalidad y observar el comportamiento del algoritmo, para mirar su viabilidad, para seguir haciendo modificaciones o pasar a estudiar otros algoritmos.

Poco después se logró obtener las instancias de 1000 y 2000 que fueron enviadas por el autor del artículo [17], en consecuencia se utilizaron para poder realizar los estudios comparativos con los artículos que usaban estas instancias y proyectar la línea base de trabajo respecto a los algoritmos que iban a ser optimizados y adaptados para futuras etapas del proyecto. En este momento es preciso mencionar que, hasta ese punto de la investigación, fue la única persona que contesto los correos electrónicos enviados a los autores de los diferentes artículos que trabajaron las instancias de alta dimensionalidad, desde mucho antes de iniciar el proyecto, a los cuales se trató de contactar con el fin de pedirles el código fuente y así poder replicar de manera correcta las implementaciones y posteriores pruebas.

Finalmente, en cuanto al hill climbing se utilizaron covering arrays para generar una correlación entre las técnicas y evitar realizar todas las combinaciones posibles de adición y remoción lo cual generó 30 combinaciones posibles, pero desafortunadamente los resultados no fueron los mejores dado que el número de máximos alcanzados no superó el 10% de las 80 instancias de 1000 y 2000 elementos y en muchos casos el algoritmo no pudo terminar su ejecución debido a estancamientos masivos.

Por limitaciones de tiempo y alcance del proyecto, se dio paso a la implementación de otros algoritmos dejando de lado el algoritmo hill climbing MOS que, aunque tuvo un comportamiento aceptable en las instancias de 100 y 200 elementos, no cumplió con los niveles de competitividad para hacer frente a los reportados en el estado del arte para instancias de alta dimensionalidad.

3.4 Cuarta Iteración

3.4.1 Algoritmo SGVNS y GRASPr

En esta etapa se procedió a implementar los algoritmos SGVNS Y GRASPr que hasta la fecha eran las dos únicas metaheuristicas de estado simple que resuelven el problema QKP para instancias de alta dimensionalidad (1000 y 2000 elementos) de manera secuencial, y que reportaban los mejores resultados para dichas instancias, debido a esto se optó por estudiar el comportamiento de estos algoritmos para incorporarlos como técnicas subordinadas en el MOS que se construyó.

El algoritmo SGVNS, es uno de los algoritmos de estado simple que resuelven las instancias 0-1 QKP de alta dimensionalidad de 1000 y 2000 elementos, donde se identificaron algunos problemas en la descripción que se hace del algoritmo en el artículo, tales como información insuficiente y ambigua, que deja a discreción del lector a la hora de implementar muchas funcionalidades que no se describen como es debido y por lo tanto no se logró programar el algoritmo exacto que los autores dicen haber utilizado. El algoritmo replicado del artículo, no logro converger a los óptimos reportados en la literatura en la gran mayoría de las instancias, presentado estancamientos constantes y con tiempos que exceden lo estipulado en el artículo. Por lo anterior, se hicieron adecuaciones propias para mejorar el rendimiento de este algoritmo como en las estructuras de vecindad y en la forma que se eligen los elementos para cada una de estas estructuras, mejorando los resultados llegando a tener una taza de éxito de 60% en las 80 instancias de prueba. Luego de exhaustivas pruebas experimentales con el algoritmo SGVNS modificado, se incorporan variables de control y fases nuevas al algoritmo que permitieron mejorar significativamente la búsqueda de soluciones óptimas hasta en un 80% de las instancias 0-1 QKP, con tiempos mucho mejores que las anteriores versiones trabajadas. A partir de este momento se consideró viable realizar la integración de MOS con el algoritmo SGVNS para aspirar mejorar los resultados obtenidos individualmente.

Luego se implementó el algoritmo GRASPr, con mucho mejor resultado que el SGVNS debido a que se tiene más detalle de los procedimientos del algoritmo descritos en el artículo, aunque también hay una sección donde no se especifica puntualmente como se hace la búsqueda local, por lo que también se hicieron modificaciones propias para mejorar esta parte y reducir el vasto espacio de soluciones que explora el algoritmo original. En el mismo trabajo se agrega al algoritmo GRASPr un componente Tabú denominado GRASP+Tabu, el cual se implementó, pero no se lograron mejores resultados que con la versión GRASPr, por lo tanto, se consideró tomar como base esta versión. Se modificó considerablemente el algoritmo, en los siguientes aspectos: 1) La forma en que se calculan algunas variables como Minlen, Maxlen y sigma. 2) También se agregaron nuevas variables de control como por ejemplo actQ y el vector GV que permiten acotar el rango de exploración en la búsqueda local. 3) Se decidió eliminar las variables gamma, beta y m que venían del algoritmo original debido a que estas no se usan en la versión modificada de GRASP dado que el cálculo de la RCL se realiza de manera diferente. Como resultado se obtuvo un algoritmo más eficiente, encontrando el máximo en un 95% de las instancias.

Finalmente se utilizaron estos dos algoritmos como técnicas subordinadas del MOS con lo cual se llegó a la propuesta final debido a que fue la combinación que mejor

sinergia presentó, tanto en tiempos de ejecución como en tasa de éxito para las instancias de alta dimensionalidad (1000 y 2000 elementos) y también debido al tiempo máximo de desarrollo del proyecto (alcance del mismo).

La comparación de resultados se realizó con los algoritmos GRASPr y IEHA, este último es el estado del arte metaheuristico para el 0-1 QKP publicado en febrero del 2017. Se descartó realizar la comparación con SGVNS [20] debido a que falta de una descripción detallada de los procedimientos mencionados allí, que impidieron replicar el experimento y por ende alcanzarlos resultados reportados en ese trabajo, por ejemplo, en sus resultados solo alcanzan el máximo 1 vez en 100 ejecuciones de prueba para cada instancia, en 15 instancias imposibilitando replicar el mismo experimento para alcanzar los resultados reportados. Se considera que por las mismas circunstancias los autores de IEHA [15] descartaron hacer comparaciones con este trabajo.

La descripción a profundidad del algoritmo propuesto se aborda en el siguiente capítulo.

4 ALGORITMO PROPUESTO

En esta sección se presenta la propuesta desarrollada en este trabajo de grado para resolver el problema 0-1 QKP en instancias de alta dimensionalidad. El algoritmo metaheurístico basado en el enfoque MOS (Multiple Offspring Sampling) que sirve como coordinador de las sub-técnicas metaheurísticas GRASPr (Greedy Randomized Adaptive Search Procedures) y SGVNS (Skewed general variable neighborhood search).

4.1 MOS (Multiple Offspring Sampling)

El algoritmo MOS [21] implementado presenta una estrategia de hibridación de relé de bajo nivel (Low-level relay hybrid, HRH) que permite que dos o más metaheurísticas subordinadas se ejecuten en secuencia, es decir una seguida de la otra. En [21] se deja a consideración del lector distintas secciones del algoritmo que se deben personalizar según el contexto del problema o según los algoritmos subordinados utilizados, una de estas partes es la forma en que se actualiza la solución que ingresa a cada técnica después de cada iteración MOS. El algoritmo MOS implementado otorga la misma entrada o vector solución a todas las técnicas al inicio de la ejecución de MOS, y luego actualiza esta entrada para cada algoritmo al finalizar la ejecución.

En el diagrama de flujo de la Figura 4.1 se puede apreciar el funcionamiento de MOS y en color verde las adaptaciones realizadas al algoritmo MOS que serán descritas con más detalle en la descripción del pseudocódigo.

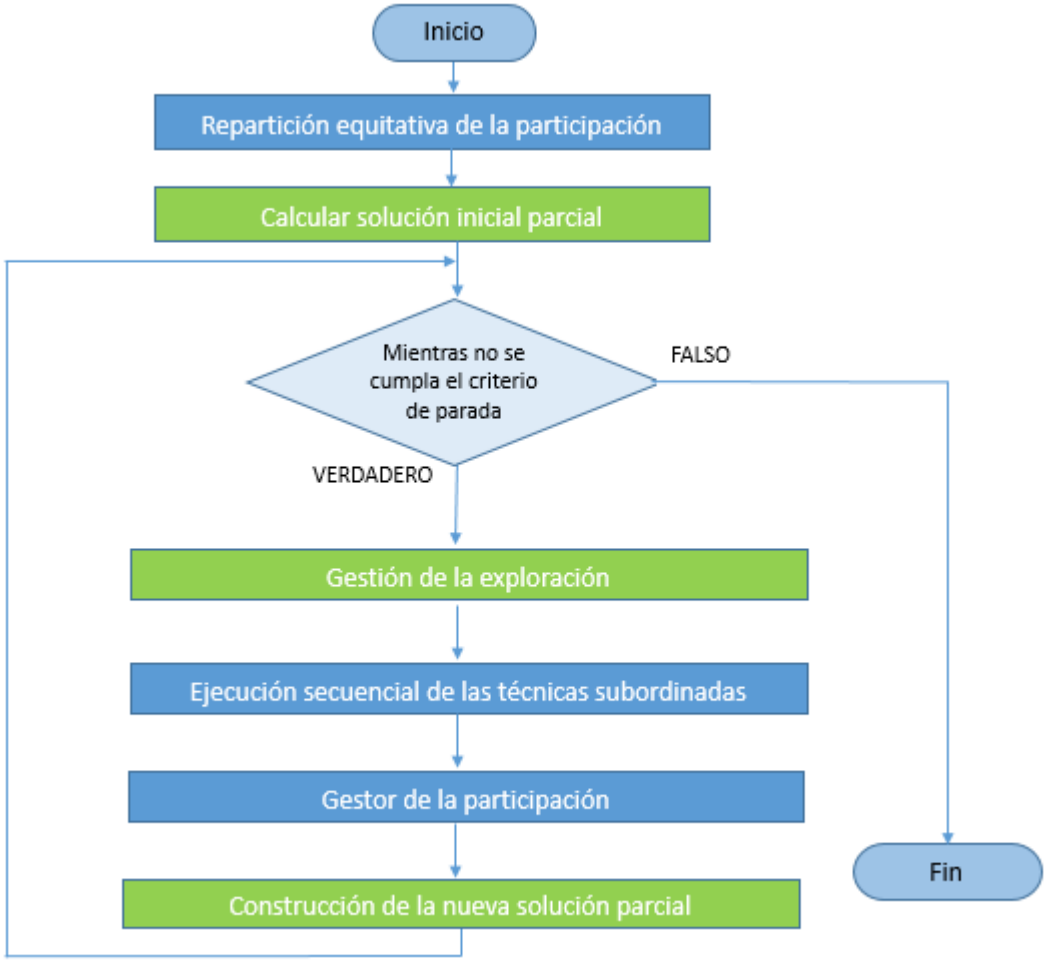

Figura 4.1 *Diagrama de Flujo MOS Adaptado*

Se implementó el algoritmo general con un bajo acoplamiento para permitir su modularidad a la hora de evaluar diferentes combinaciones de técnicas, y proporcionar a cada una de estas solamente la información necesaria para que puedan funcionar bajo las condiciones de MOS y así delegar el funcionamiento interno de las técnicas a ellas mismas, sin que interfieran en el proceso superior. MOS gestiona el número de llamadas o invocaciones a la función objetivo para cada subordinado, función especificada formalmente en la Algoritmo 4.1.

El algoritmo MOS adaptado e implementado en el trabajo de grado el cual se explica detalladamente a continuación.

Algoritmo MOS (stepFES, it, ξ, instancia)

```
1.  leer instancia y obtener variables c, p, w, n, MaxV
2.  Inicializar vector de frecuencias Qf
3.  [X ,GV] = Solución Inicial (c, p, w, n)
4.  indiceS1=0
5.  indiceS2=0
6.  PARA i=1 HASTA Numero de Técnicas HACER
7.      P(i) = 1/Numero de Técnicas
8.      FES(i) = StepFES * P(i)
9.  FIN PARA
10. ei1 = 0 , ei2 = 0
11. fa1 = 0 , fa2 = 0
12. bestGV = GV
13. MIENTRAS No se exceda el número de iteraciones it HACER
14.     SI (P(1)>P(2) && (fa2>fa1 || (fa1==0 && fa2==0))) ENTONCES
15.         SI (indiceS1 ~= indiceS2) ENTONCES
16.             fa2 = fa2 + 0.06;
17.             fa1 = fa1 - 0.0025;
18.         SINO
19.             fa2 = fa2 + 0.008*it;
20.             fa1 = fa1 - 0.0025;
21.         FIN
22.         SI (fa1<0) ENTONCES
23.             fa1 = 0;
24.         FIN
25.         SI (fa2>0.08)    ENTONCES
26.             fa2 = 0.08;
27.         FIN
28.         SINO (P(2)>P(1) && (fa1>fa2 || (fa1==0 && fa2==0)))
                ENTONCES
29.         SI (indiceS1 ~= indiceS2) ENTONCES
30.             fa1 = fa1 + 0.06;
31.             fa2 = fa2 - 0.0025;
32.         SINO
33.             fa1 = fa1 + 0.0075*it;
34.             fa2 = fa2 - 0.0025;
35.         FIN
36.         SI (fa2<0) ENTONCES
37.             fa2 = 0;
38.         FIN
```

```
39.             SI (fa1>0.075) ENTONCES
40.                 fa1 = 0.075;
41.             FIN
42.         FIN
43.         [Xt1, Qf, e1, bestGV, Fprom1] = SGVNS (X, c, p, w, Qf, FES(1),
                                                   Maxv, e1, GV, fa1, bestGV)
44.         SI fitness(Xt1) >= Maxv
45.             Xbest = Xt1
46.             ROMPER CICLO LOCAL
47.         FIN SI
48.         [Xt2, Qf, e2, bestGV, Fprom2] = GRASPr(X, c, p, w, Qf, FES(2),
                                                   Maxv, e2, GV, fa2, bestGV)
49.         SI fitness(Xt2) >= MaxV ENTONCES
50.             Xbest = Xt2
51.             ROMPER CICLO LOCAL
52.         FIN SI
53.         indiceS1 = número de elementos en Xt1
54.         indiceS2 = número de elementos en Xt2
55.         actualizar vector de participación P(Fprom1,Fprom2)
56.         actualizar vector de Evaluaciones de Función (FES(i), ξ)
57.         inds = Obtener_elementos_con_frecuencia_entre ((1…n), Qf, 1, ∞)
58.         inds = intersección entre (bestGV y inds);
59.         nind = número de elementos en inds
60.         X = (0, 0, 0, ..., 0) // solución con n elementos no seleccionados
61.         GV  = [ ]
62.         PARA i = 1 HASTA nind HACER
63.             SI((Peso de X + w(inds(i)) < (c*0.8)) ENTONCES
64.                 X(inds(i))=1
65.                 Agregar el elemento inds(i) a GV
66.             FIN SI
67.         FIN PARA
68.     FIN MIENTRAS
69.     RETORNAR Xbest
```

Algoritmo 4.1. MOS adaptado al QKP

4.1.1 Parámetros de entrada MOS

MOS recibe 4 parámetros los cuales son: el número de evaluaciones de función objetivos (stepFES), el número total de iteraciones (it), el factor de reducción (ξ), y el nombre de la instancia (o el archivo de disco donde la misma se almacena) que se va a ejecutar con el algoritmo. A continuación, se describe cada uno de estos parámetros:

- **Número de FES por iteración (StepFES)**

En [21] se especifica que el número de FES a ser repartido entre las técnicas para cada iteración MOS puede ser estático o dinámico, para el trabajo actual se decidió manejar un número estático de evaluaciones de función en cada iteración puesto que no se encontró un beneficio particular en manejar un valor dinámico y de hacerlo así se dificultaba más el estudio del progreso de las técnicas en el transcurso de la ejecución del algoritmo, El número de FES se utiliza en las líneas (6-18) para el cálculo inicial de participación y el re-cálculo del mismo en cada iteración.

- **Número de Iteraciones MOS (it)**

Este parámetro especifica cuantas iteraciones MOS se otorgan para la ejecución del ciclo principal que inicia desde la línea 11 hasta la línea 40. El número total de evaluaciones de función globales será igual al StepFES multiplicado por el número de iteraciones.

- **Factor de Reducción (ξ)**

El núcleo de MOS se encarga de la distribución controlada del número de evaluaciones de la función objetivo en cada una de las técnicas subordinadas, lo cual es regulado por la variable épsilon (ξ), que indica el factor de reducción o tasa de transferencia de evaluaciones de la función objetivo de una técnica a otra, esto se hace después de que cada técnica se ha ejecutado y se tienen los resultados de su desempeño. El valor de ξ se utiliza en el cálculo de $\Delta_i^{(j)}$ como se muestra en la **Ecuación 4.3**.

- **Nombre de la instancia**

Recibe como parámetro el nombre de la instancia que se trabajará durante la ejecución del algoritmo, extrayendo de un archivo plano con este mismo nombre, la información necesaria para que los dos algoritmos subordinados puedan resolver el problema de la mochila cuadrática binaria, tal como el número de elementos (n), la matriz de beneficios (p), el vector de pesos (w), la capacidad de la mochila (c) y la mejor solución reportada por el estado del arte (Maxv). El algoritmo MOS se encarga de compartir esta y otra información necesaria para el funcionamiento de las técnicas subordinadas.

A continuación, se inicia con la descripción del *Algoritmo 4.1* MOS.

En la línea 1 del algoritmo MOS se hace la lectura del archivo de la instancia 0-1 QKP que aporta los datos necesarios para resolver el problema. Luego se inicializa el vector de frecuencias Qf en la línea 2 donde se almacena el número de veces

que todos los elementos han pertenecido a un óptimo local, y este vector es utilizado por todas las técnicas en las respectivas fases de búsqueda local.

4.1.2 Solución inicial

Se crea una solución inicial en la línea 3 para no empezar desde una solución vacía ya que para resolver problemas 0-1 QKP es de suma importancia iniciar con una buena solución, es decir proporcionar una solución inicial que oriente la búsqueda a zonas de buena calidad, ya que resulta bastante costoso e innecesario aplicar un algoritmo de búsqueda local sobre una solución vacía, con pocos elementos o peor aún si estos han sido agregados aleatoriamente, debido a que para llegar a una solución cercana a la óptima, a partir de operaciones de búsqueda local sobre un cromosoma construido sin ningún criterio de calidad, puede conllevar a consumo de tiempo de ejecución elevado el cual crece exponencialmente cuando se trabaja con instancias de alta dimensionalidad. Incluso se encontró que partir de una mala solución dirige la búsqueda local hacia zonas de baja calidad y el algoritmo queda atrapado en óptimos locales.

Algoritmo Solución Inicial (c, p, w, n)

```
1.  aux = (), wx = 0, GV=()
2.  X = (0, 0, 0, ..., 0) // solución con n elementos no seleccionados
3.  PARA i=1 HASTA n HACER
4.      aux (i) = densidad absoluta elemento i
5.  FIN PARA
6.  p = elemento con mayor valor en aux
7.  X(p) = 1 //indica que la posición p se selecciona en la solución X
8.  wx = w (p)
9.  ind0 = elementos que no pertenecen a la solución y caben en la mochila
10. n0 = número de elementos en (ind0)
11. MIENTRAS (n0 > 0 && wx < c) HACER
12.     aux = ()
13.     PARA i=1 HASTA n0 HACER
14.         aux (i) = densidad relativa del elemento i en ind0
15.     FIN PARA
16.     ordenar de mayor a menor valor el vector aux
17.     in = 0
18.     re = wx/c;
19.     maxin = redondear(14*re^2 - 30*re + 17);
20.     PARA j = 1 HASTA número de elementos en aux HACER
21.         p = índice de elemento aux (j)
22.         SI wx + w(p) <= (c*0.9) ENTONCES
23.             X(p)=1
```

```
24.                         wx = wx + w(p)
25.                         in = in + 1
26.                         Agregar elemento p a GV
27.                         SI in == Maxin ENTONCES
28.                             romper ciclo local
29.                         FIN SI
30.                     FIN SI
31.             FIN PARA
32.             ind0 = elementos que no pertenecen y caben en la solución.
33.             n0 = número de elementos en ind0
34.     FIN MIENTRAS
35.     RETORNAR X, GV
```

Algoritmo 4.2 *Solución inicial*

El algoritmo utilizado para generar la solución inicial está basado en el que fue usado en [20], donde primero en la línea 1 se inicializa la variable aux, la cual es una variable temporal en la que se guardara el densidad absoluta de cada elemento, wx que es el peso de la solución que se va a construir y el vector GV en el que se guardan los índices de los elementos en el mismo orden que se van agregando a la solución X, para luego utilizar esta información en las técnicas subordinadas, más específicamente en sus fases de búsqueda local con el objetivo de reducir el espacio de búsqueda e intensificar el esfuerzo en analizar intercambios de ciertos elementos que tienen más posibilidad de alcanzar mejores soluciones. En la línea 2 se inicializa el vector solución que se va a construir denominado por la variable X.

Luego ingresa el elemento con mayor densidad absoluta a la mochila (líneas 3-8), y después se termina de llenar agregando elementos basado en su densidad relativa (líneas 9-31), ordenando este valor de mayor a menor (línea 16) cada vez que es calculado, y luego se agregan los elementos en este orden (líneas 19-27). Como se puede apreciar, este procedimiento es totalmente determinista (siempre y cuando todos los parámetros sean los mismos), es decir, que siempre se genera como solución inicial la misma en todas las ejecuciones.

Para obtener los elementos que no pertenecen a la solución como sucede en la línea 9 del Algoritmo 4.2, se utiliza la rutina que se muestra en el Algoritmo 4.3, donde ingresa como parámetro el vector solución binario, y se obtienen los índices que no pertenecen a la solución en el vector ítems.

```
Algoritmo Obtener_elementos_NO_pertenecen_solución ( X )
1.      items = ()
2.      PARA i=1 HASTA número de elementos en X HACER
```

```
3.          SI X(i)==0 ENTONCES
4.              Agregar i al vector items
5.          FIN SI
6.      FIN PARA
7.      RETORNAR ítems
```

Algoritmo 4.3 *Seleccionar los elementos que NO Pertenecen a una Solución*

También se implementa una rutina para obtener los elementos que pertenecen a la solución, la cual se muestra en el Algoritmo 4.4.

```
Algoritmo Obtener_elementos_pertenecen_solución ( X )
1.      items = ()
2.      PARA i=1 HASTA número de elementos en X HACER
3.          SI X(i)==1 ENTONCES
4.              Agregar i al vector items
5.          FIN SI
6.      FIN PARA
7.      RETORNAR items
```

Algoritmo 4.4 *Seleccionar los elementos que Pertenecen a una Solución*

Al algoritmo de solución inicial propuesto en [20] se le realizó una modificación para aumentar la velocidad de ejecución de este procedimiento. El algoritmo original calcula la densidad relativa de todos los elementos luego de agregar un elemento a la mochila, lo que para instancias de alta dimensionalidad tiene un alto costo en tiempo de ejecución, incluso llegando a durar más que toda la ejecución del resto del algoritmo en algunas instancias. Para abordar este problema, se agregó una variable llamada Maxin, que determina cuántos elementos se pueden ingresar a la mochila después de realizar un cálculo de densidad relativa de todos los elementos que se encuentran fuera de la solución y después de esto volver a realizar este cálculo, por ejemplo, si este valor es 15, entonces se agregaran 15 elementos después de hacer el cálculo de densidad, y luego se calcula otra vez la densidad, y así sucesivamente hasta que no haya espacio.

El cálculo de Maxin se hace en las líneas (18-19) del Algoritmo 4.2, donde primero se hace el cálculo de la variable temporal *re*, que representa un índice de crecimiento del peso parcial *wx* respecto al peso total *c*, que crece desde 0 hasta a 1 a medida que se van ingresando elementos a la solución actual, y permite obtener valores de Maxin desde 14 al inicio del procedimiento hasta valores cercanos a 1. La tabulación de valores de *re* y el valor de Maxin se muestra en la Tabla 4.1.

***re* = wx/c**	**Valor Maxin**
0,1	14
0,25	10
0,5	6
0,75	1,5
1	1

Tabla 4.1 *Valores para Maxin*

El valor de *re* se calcula con la siguiente ecuación:

$$re = wx / c$$

Ecuación 4.1 *Porcentaje de peso parcial*

Donde wx es el peso de la solución actual y c es la capacidad total de la mochila.

En la Tabla 4.1 especifica que valores aproximados debería tener Maxin en determinados puntos porcentuales, en función de cuantos elementos han sido agregados a la mochila, con lo que se busca hacer una regresión lineal para obtener una ecuación que facilite este cálculo durante el proceso. La ecuación resultante se muestra en la Ecuación 4.2.

$$maxin = 13.98*re^2 - 30*re + 17$$

Ecuación 4.2 *Maxin*

La Ecuación 4.2 tiene un comportamiento polinomial de grado 2, que tiende progresivamente hacia el eje x representando una explotación progresiva. La grafica de la ecuación Maxin se muestra en la Figura 4.2.

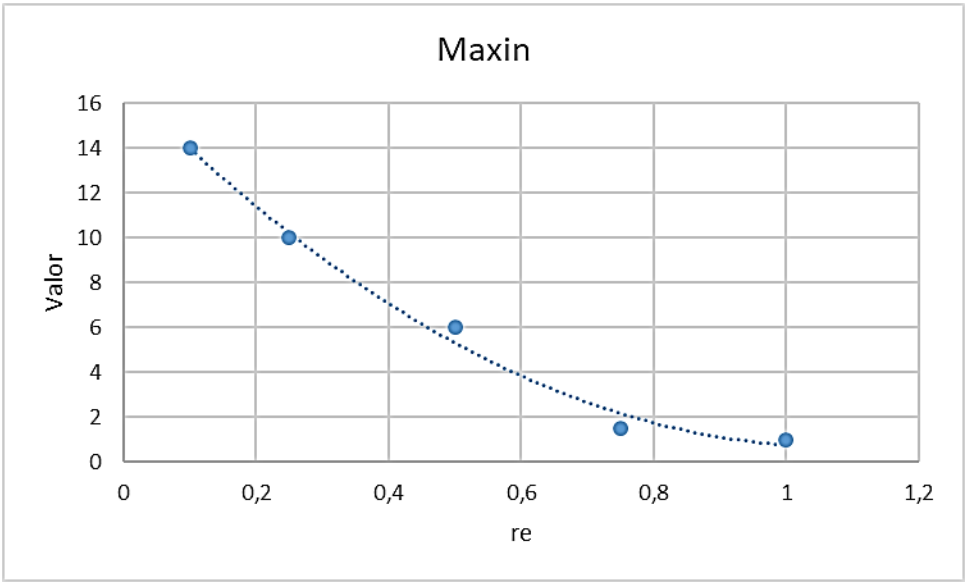

Figura 4.2 *Grafica de la ecuación Maxin*

Se confirmó experimentalmente que para la mayoría de las instancias los resultados eran los mismos ya sea realizando o no la implementación de Maxin, y en otros casos la solución tenía un poco menor fitness, pero no una diferencia considerable lo que justifica utilizar el control de densidad con Maxin, evitando así realizar costosas e innecesarias llamadas a la función objetivo o de fitness. En cualquier caso, se optó por delegar el ejercicio de encontrar el óptimo global a los procedimientos de búsqueda local de la versión del SGVNS implementada en este trabajo.

4.1.3 Inicialización de la participación

Continuando con la descripción del algoritmo MOS, entre las líneas (6-9) se hace la repartición equitativa de participación para todas las técnicas (en este caso 2). En la línea 10 se realiza la declaración de las variables de estado ei1 y ei2, las cuales guardan la iteración en la que quedó la técnica 1 y 2 respectivamente al finalizar la ejecución de una de estas cuando se haya alcanzado las (FES) asignadas por MOS en dicha iteración.

4.1.4 Gestión de la exploración

En la línea 11 se incorporan 2 variables de control denominadas fa1 y fa2, para la primera y segunda técnica respectivamente, que tienen la función de llevar el control del valor que MOS otorga (líneas 14-42) en cada nueva iteración para permitir que se regulen las zonas o planos donde se explora una solución, y estas variables son ajustadas y enviadas a las técnicas subordinadas para que estas las utilicen en la fase de pre construcción por peso que fue incorporada en las dos técnicas subordinadas con el fin de que haya una correspondencia o hilo que las guie hacia

donde se estén encontrando mejores soluciones, tomando como punto de referencia el mismo criterio con el que se reparten las evaluaciones de función.

4.1.5 Ciclo interno MOS

En la línea 12 se inicializa el vector bestGV, que guarda el mejor vector greedy obtenido en toda la ejecución y se usa en la fase de actualización de la solución (líneas 57-67) con la información que proporciona este vector. En el ciclo entre las líneas (13-68) se realiza la ejecución de MOS hasta que se agoten todas las iteraciones especificadas por el parámetro *it* anteriormente. En las líneas (43-52) se ejecutan en forma secuencial las dos técnicas subordinadas (para efectos prácticos, si estas dos líneas se ejecutan en paralelo y se sincroniza la ejecución en la línea 27 los resultados son similares a la ejecución secuencial), cada técnica retorna los siguientes parámetros: la mejor solución encontrada (Xt1 o Xt2), el vector de frecuencias Qf actualizado con la información obtenida en la ejecución de las técnicas, el valor de la iteración principal de cada técnica ei1 y ei2 para guardar el progreso al agotarse las evaluaciones de función y simular el curso natural en la ejecución, también se entrega el mejor vector greedy bestGV y el promedio de fitness Fprom1 o Fprom2 de las técnicas para reajustar su participación.

4.1.6 Gestión de la participación

En las líneas (55-56) del algoritmo MOS se actualiza la participación de cada una de las técnicas asignando las FES que ejecutarán cada una de las técnicas en la siguiente iteración. La gestión de la participación de las técnicas consiste en repartir al inicio de cada iteración el número de FES que cada técnica usará con base en la información de los resultados obtenidos por cada una de estas en la iteración anterior.

Se utiliza la **Ecuación 4.3** y la **Ecuación 4.4** para el ajuste de la participación de cada técnica durante el proceso de búsqueda. Primero se calcula el valor de $\Delta_i^{(j)}$ para cada técnica j en la iteración i, siendo un factor que define la diferencia relativa entre la técnica con mejor desempeño Q_i^{max} y las demás técnicas, y se calcula con la Ecuación 4.3.

$$\Delta_i^{(j)} = \xi \bullet \frac{Q_i^{max} - Q_i^{(j)}}{Q_i^{max}} \bullet \Pi_{i-1}^{(j)} \; \forall j \in [1, n] \Big/ j \notin \Omega$$

Ecuación 4.3 *Factor de diferencia relativa entre técnicas [21]*

Donde $\Pi_{i-1}^{(j)}$ representa el porcentaje de participación para cada técnica j en la iteración i-1 (iteración anterior) y $Q_i^{(j)}$ es la calidad o desempeño de la técnica j durante la iteración i, que para el algoritmo propuesto se trabajó como la sumatoria del fitness de cada individuo producido al final de cada iteración de los algoritmos subordinados (en la sección posterior donde se presentan los algoritmos

subordinados se describe más detalladamente este proceso). El valor de Q_i^{max} se calcula con la Ecuación 4.4. El valor de ξ se utiliza en esta ecuación para calcular el incremento o decremento de participación a una técnica determinada.

$$\Pi_i^{(j)} = \begin{cases} \Pi_{i-1}^{(j)} + \eta & if\ j \in \Omega, \\ \Pi_{i-1}^{(j)} - \Delta_i^{(j)} & otherwise \end{cases}$$

$$\eta = \frac{\sum_{k \notin \Omega} \Delta_i^{(k)}}{|\Omega|}$$

$$\Omega = \left\{ l \;\middle/\; Q_i^{(l)} \geq Q_i^{(m)}\ \forall l, m \in [1, nt] \right\}$$

$$Q_i^{max} = \max\{Q_i^{(j)}\ \forall j \in [1, nt]\}$$

Ecuación 4.4 *Ajuste de la participación [21]*

En la Ecuación 4.4 se determina el conjunto Ω luego de encontrar los valores $\Delta_i^{(j)}$, que es el conjunto de técnicas que tienen el mayor desempeño $Q_i^{(j)}$, y luego se encuentra η, que es el incremento de participación en $\Pi_{i-1}^{(j)}$ que obtendrá la técnica j en caso de pertenecer al conjunto Ω, y se hace el descuento de participación correspondiente $\Delta_i^{(j)}$ a cada técnica que no pertenece a este conjunto Ω.

Ejemplo: Si se tiene dos técnicas subordinadas de MOS y se encuentra en la iteración (i=2), en este paso el porcentaje de participación $\Pi_{i=2}^{(j)}$ obtenido es **0.4** para la técnica j=1 y **0.6** para la técnica j=2, la calidad de desempeño $Q_i^{(j)}$ de las dos técnicas en dicha iteración es **50** para la técnica j=1 y **60** para la técnica j=2. Con lo anterior se obtiene que Q_i^{max} = **60** que es la técnica con la mejor calidad de desempeño en esta iteración, esto se obtiene del conjunto de técnicas con mejor calidad de desempeño Ω; el factor de reducción ξ se fija arbitrariamente para este ejemplo en 0.2. Posteriormente por medio de **Ecuación 4.3** se calcula el factor de diferencia relativa para la técnica 1 que es la única que no pertenece al conjunto Ω, de esta forma: $\Delta_2^{(1)} = 0.2 \bullet \frac{60-50}{60} \bullet 0.4$ lo cual da como resultado $\Delta_i^{(j)}$ = 0.0133. Este último valor se utiliza en **Ecuación 4.4** para calcular el ajuste de la participación para las dos técnicas en la siguiente iteración (i=3); Se debe calcular también el valor de η que van a incrementar las técnicas del grupo Ω, asi: $\eta = \frac{0.0133}{1}$. Finalmente el ajuste de la participación para la siguiente iteración $\Pi_3^{(j)}$= { $\Pi_{i=2}^{(1)}$ (0.4 – 0.0133), $\Pi_{i=2}^{(2)}$ (0.6 + 0.0133)} en consecuencia el ajuste de la participación en la iteración i=3 tendrá $\Pi_3^{(j)}$ = {0.3867, 0.6133}. A continuación, en la Figura 4.3 se puede observar

gráficamente cómo funciona el ajuste de la participación de acuerdo con el ejemplo anterior.

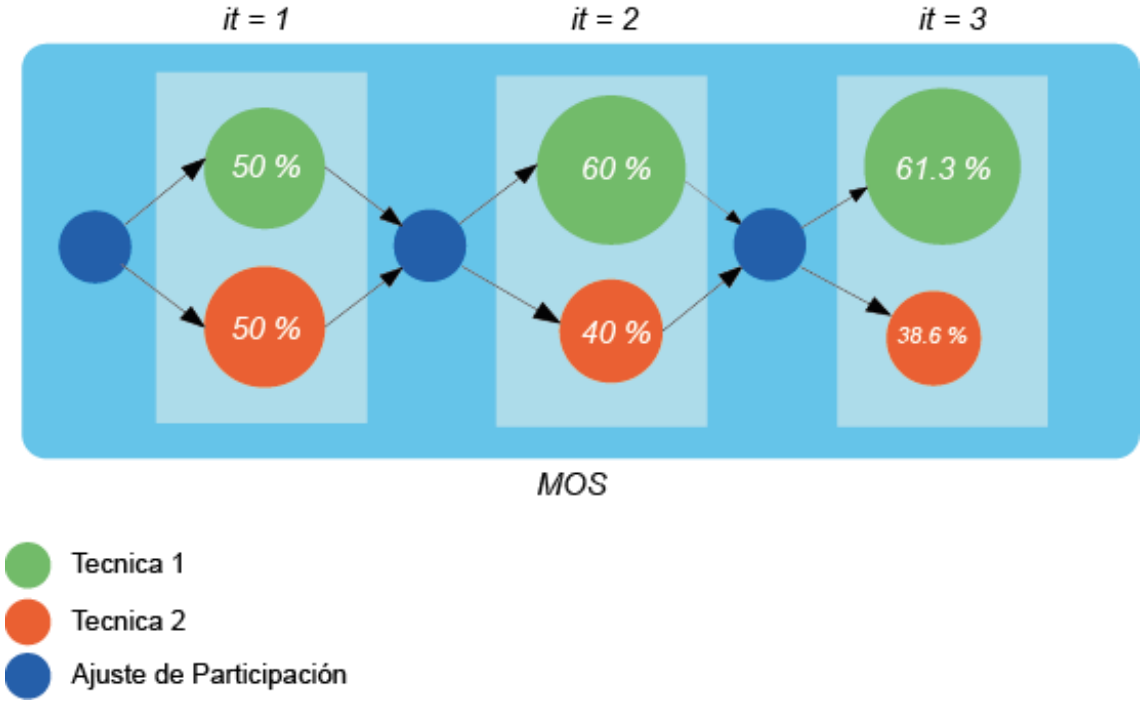

Figura 4.3 *Ejemplo MOS*

4.1.7 Actualización de la solución actual

El algoritmo principal continua el flujo con las líneas (57-67) donde se realiza la actualización de la solución de la siguiente iteración de modo similar al que se hace en [14]. En esencia un algoritmo que cuenta con fase de construcción tipo GRASP como las técnicas subordinadas de MOS, siempre debe disponer de una solución vacía o parcial para funcionar, nunca debe estar llena porque el proceso principal de un algoritmo tipo GRASP es la fase de construcción, donde se implementa una rutina de adición de elementos que atribuye al algoritmo la característica de inicio múltiple, que es la generación de soluciones distintas al inicio de cada iteración (al finalizar cada iteración del ciclo principal), pero la semilla de estas soluciones debe ser cambiada periódicamente para evitar estancamientos y evitar malgastar tiempo en zonas de baja calidad.

Para definir la entrada o solución actual de cada técnica luego de cada iteración MOS, se crea a partir del vector de frecuencias considerando solamente esos elementos que tienen una frecuencia mayor a cero en este conjunto, teniendo en cuenta así aquellos objetos que tienen una tendencia a quedar en las diferentes soluciones obtenidas en las técnicas subordinadas. Este proceso se hace solamente hasta que no haya más elementos con el valor máximo, o hasta que se llegue a un 80% de la capacidad de la mochila, debido a que las fases de

construcción de los algoritmos subordinados lo demandan para hacer una exploración inicial solamente en este 20% sin llenar, y en este sentido otorgar a las técnicas una aproximación con calidad considerable que funcione como semilla para la construcción de soluciones.

4.2 ALGORITMOS SUBORDINADOS

El algoritmo MOS desarrollado en este trabajo se compone de dos algoritmos subordinados, el SGVNS y el GRASPr, ambos de búsqueda local y de estado simple. A continuación, se hace la descripción de la adaptación realizada de estos algoritmos que fue necesaria para incrementar la efectividad de los mismos en la solución de 0-1 QKP.

4.2.1 Técnica subordinada SGVNS

La adaptación llevada a cabo del SGVNS [20] no logró obtener los resultados que este mismo reporta en su publicación, por lo que se hicieron considerables adecuaciones al algoritmo original para mejorar los resultados, con modificaciones en las estructuras de vecindad, en la actualización de la solución, permitir el paso de información entre técnicas a través de MOS, y añadir las fases de pre construcción por peso y construcción por densidad, para generar una solución personalizada al inicio de cada iteración. El SGVNS tiene su núcleo en la fase de búsqueda local, y en esta se implementa un algoritmo de vecindad variable en descenso (VND) que es una variante del VNS convencional propuesto en [27], que se caracteriza principalmente porque el cambio de sus estructuras de vecindad se efectúa de forma determinística, y en el mismo sentido en un procedimiento de búsqueda local que por medio de determinadas vecindades busca secuencialmente óptimos locales, saltando a nuevos puntos del espacio de búsqueda y permitiendo analizar con diferentes estrategias los vecinos de la solución actual, ayudando a escapar de estancamientos locales cuando una estructura de vecindad no puede encontrar una mejora.

A continuación se muestra el diagrama de flujo de la adaptación del algoritmo SGVNS donde se resalta en color verde las modificaciones hechas al algoritmo original Figura 4.4.

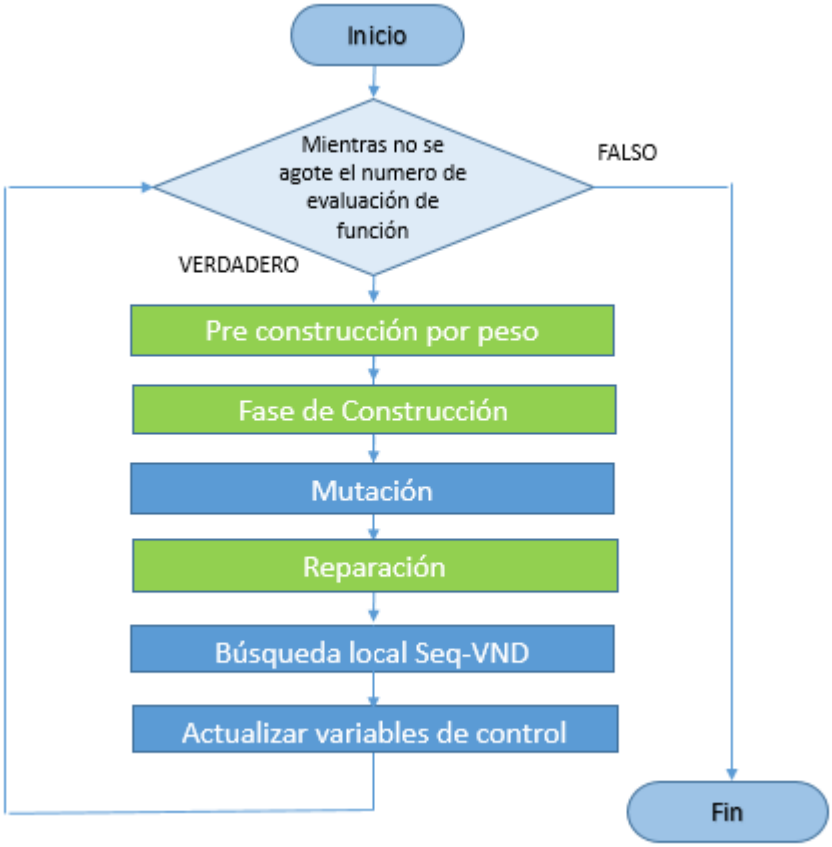

Figura 4.4 *Diagrama flujo adaptación SGVNS*

En el siguiente Algoritmo 4.5 se ilustra el funcionamiento y las adaptaciones que se realizaron para la técnica subordinada SGVNS.

Algoritmo SGVNS (X, Xbest, c, p, w, Qf, FES, Maxv, ei, GV, fa, bestGV)

1. h_{max} = 10
2. Ind0 = obtener elementos que no pertenecen a solución parcial X
3. Ind0 = ordenar elementos en ind0, menor a mayor en función del peso
4. Fprom = 0
5. **REPETIR**
6. h ← 1
7. **MIENTRAS** h+ei ≤ h_{max} **HACER**
8. X' = X
9. GV' = GV
10. reg = número aleatorio entre 0 y 0.03
11. **MIENTRAS** hallan elementos en ind0 **HACER**
12. i = primer elemento en ind0
13. **SI** (peso de X'+ w(i)) > (c * 1 - reg - fa) **ENTONCES**

```
14.                     Romper ciclo local
15.                 FIN SI
16.                 agregar elemento i a X'
17.                 agregar elemento i a GV'
18.                 eliminar elemento i de ind0
19.             FIN MIENTAS
20.             MinLen = redondear(1,5*h - 0,7)
21.             MaxLen = redondear(2.4315*h + 4.7561)
22.             actQ = (k)/ hmax
23.             [X',GV] = CP(Minlen,Maxlen,k,Qf,X',kc,c,v,actQ,GV')
24.             X'← shaking (X', w, c, h)
25.             X' ← reparar (X', c, p, w)
26.             X' ← Seq-VND (X', c, p, w, Qf, Maxv,GV')
27.             Actualizar Qf con X'
28.             Actualizar valor de Fprom con X'
29.             SI f(X') > f(X_best) ENTONCES
30.                 X_best ← X'
31.                 bestGV = GV'
32.                 SI f(X') >= maxv ENTONCES
33.                     RETORNAR X_best,Qf,ei,bestGV,Fprom
34.                 FIN SI
35.                 h ← 1
36.             SI NO
37.                 h ← h + 1
38.             FIN SI
39.         FIN MIENTRAS
40.         ei=0
41.     HASTA que el número de FES se agote.
42.     Asignar a ei el valor h en el cual quedó
43.     RETORNAR X_best,Qf,ei,bestGV,Fprom
```

Algoritmo 4.5 *SGVNS*

- **Parámetros de Entrada de SGVNS**

Entre los parámetros recibidos por el SGVNS está el vector solución base X, la mejor solución encontrada Xbest, los proporcionados por el archivo de la instancia 0-1 QKP, el vector de frecuencias Qf, el número de evaluaciones de la función objetivo asignadas, el máximo valor reportado para la instancia, el valor de iteración anterior ei2 el vector greedy GV, el factor de ajuste GA y el mejor vector greedy global bestGV. A continuación, se describe cada uno de ellos:

Vector Solución Base X. Para esta implementación del SGVNS no se trabaja con una solución base que está totalmente al tope de su capacidad, sino que se tiene una solución parcial que se encuentra a un 80% de su capacidad, esto debido a la adición de características propias del algoritmo GRASP que en su fase de construcción utiliza el 20% restante para crear múltiples inicios en cada iteración y evitar la presencia de estancamientos que presenta el algoritmo original SGVNS.

Vector Xbest. Se debe recibir del algoritmo principal MOS la mejor solución encontrada globalmente hasta el momento para controlar el progreso gradual que se obtiene en esta y todas las técnicas subordinadas.

Parámetros de la instancia. Son los parámetros proporcionados por el archivo plano de la instancia a resolver, como lo son la capacidad de la mochila (c), el número de elementos (n), el vector de pesos (w) y la matriz de beneficios (p).

Vector de Frecuencias (Qf). Este vector permite llevar un conteo del número de veces que determinado elemento ha sido parte de los óptimos locales encontrados en las diferentes fases del algoritmo. El vector (Qf) es usado en la fase de búsqueda local principalmente para reducir el espacio de búsqueda al momento de hacer intercambios, debido a que almacena un historial de cuantas veces los elementos han sido parte de la solución, lo cual dirige la búsqueda a elementos que han tenido más probabilidad de estar en las soluciones a lo largo de las anteriores fases del algoritmo. También se utiliza la información de este vector para reconstruir la solución parcial al final de cada iteración, que sirve como entrada para las técnicas subordinadas, otorgando esta misma entrada a las dos técnicas para que de esta forma haya igualdad de condiciones para medir el desempeño de las mismas.

Número de evaluaciones de función (FES). Parámetro calculado y proporcionado por MOS, para controlar el número de veces que se podrá llamar la función objetivo. Valor calculado a partir del desempeño que cada técnica logre al final de cada iteración.

Valor Máximo reportado para cada instancia (MaxV). Es el valor máximo reportado en [14] para cada instancia, utilizado como criterio de parada tanto en MOS como en las técnicas subordinadas, cuando el valor máximo alcanzado en determinado punto de la ejecución sea igual o superior a este valor, permitiendo dado el caso, encontrar valores superiores a los reportados en el estado del arte.

Estado de iteración (ei). En esta variable se recibe el valor de la iteración en la cual SGVNS termino su ejecución en la iteración MOS anterior, dado a que en las evaluaciones de función asignadas no es posible llevar a cabo toda la ejecución

natural del algoritmo, y por ello es necesario guardar el valor del iterador h en que se agotaron estas evaluaciones de función, para simular el transcurso normal del algoritmo.

Greedy Vector (GV). En este parámetro se recibe el vector greddy base desde el algoritmo principal, el cual es relativo a la solución inicial creada al inicio de la ejecución, y debe ser completado por cada técnica subordinada en las fases de pre construcción por peso y construcción por densidad. Es utilizado tanto por SGNVS y GRASPr en la fase de búsqueda local, específicamente en la estructura de vecindad SWAP2, donde con ayuda de este vector se hace una reducción de los elementos candidatos para este proceso.

Factor de ajuste (fa). Este parámetro es calculado y entregado desde el algoritmo principal MOS, el cual permite controlar la fase de pre construcción por peso que es indispensable para que las técnicas converjan hacia planos de soluciones que muestren tendencia a soluciones con mejor calidad.

Mejor Vector Greedy (bestGV). Durante cada iteración MOS se lleva un control de cuál fue el vector greedy con el que se obtuvo la mejor solución en las técnicas subordinadas, por lo que se envía como parámetro este valor para actualizarlo en caso de que se logre mejorar la solución global hasta el momento, para luego utilizar este vector greedy en la fase de actualización del algoritmo principal MOS con el fin de proveer la mejor semilla a este proceso, y conjuntamente aportar a tener una buena solución base al inicio de cada iteración MOS.

El algoritmo SGVNS inicia la exploración con una solución X que se recibe como parámetro desde MOS, el cual es una solución parcial que no está totalmente al tope de la capacidad, para que esta sea completada en la fase de construcción incorporada en este algoritmo, luego se inicializa el número de iteraciones internas (h) que ha sido fijado a 10 para lograr alcanzar una cobertura amplia de los valores Minlen y Maxlen que están en función de la variable h y son usados en la fase de construcción (CP), este valor de h también especifica el número de cambios que se realizan en el procedimiento de shaking o sacudida. Luego se obtiene el vector ind0, que contiene los índices de los elementos que no pertenecen a la solución actual pero ordenados en función de su peso de menor a mayor. Después de esto se repiten las líneas (5-41) mientras que el criterio de parada no se cumpla, en este caso mientras el número de evaluaciones de función asignadas no se agoten o si se encuentra el máximo fitness reportado en la literatura para la instancia. El ciclo de las líneas (7- 39) se repite mientras h sea menor o igual que 10, y se reinicia el valor de h a 1 cuando se rompa el ciclo y vuelve a empezar, siempre y cuando haya FES disponibles para utilizar.

- **Fase de pre construcción por peso**

Entre las líneas (8-19) se efectúa la fase de pre construcción por previa a la fase de construcción basada en GRASP, que busca hacer un ajuste regulado que permita a las técnicas orientar su búsqueda hacia zonas prometedoras basado en que ambas técnicas exploran espacios distintos, y esto se logra indicando a la otra técnica que explore o se salga del espacio en el que esta, haciendo ajustes graduales que permitan ir recorriendo y visitando otros planos de soluciones. El llenado que se realiza en esta fase se hace hasta un porcentaje por debajo de la capacidad total de forma que sea posible ingresar más elementos en la fase de construcción.

El paso inicial consiste en realizar una copia del vector X y GV que se tienen como base durante la ejecución de SGVNS en una iteración MOS, debido a que estos se encuentran a un 85% de la capacidad desde la segunda iteración (90% en la primera) y luego se completan en las fases siguientes, y se necesita esta misma base para la siguiente iteración SGVNS. Esta fase cuenta con dos variables de regulación que son las que determinan el porcentaje de llenado que tendrá la solución base antes de entrar la fase de construcción siguiente. Por un lado, se incorpora la variable reg que permite añadir un comportamiento aleatorio al llenado de esta fase, el cual se requirió para mantener un avance dinámico, que haga saltos pequeños por las zonas por donde se va avanzando la exploración. También se regula la capacidad de esta fase con la variable fa (Algoritmo 4.5) que se recibe desde el algoritmo principal, donde se actualiza y calcula para inducir más o menos valor, al porcentaje de llenado.

En la primera técnica la estrategia en la fase de pre construcción por peso es iniciar en un llenado alto hasta el 100% de la capacidad con un índice de variación hacia abajo de 3%(97% hasta 100%) representado por la variable reg, logrando con esto alcanzar planos altos o de más número de elementos en las soluciones, entre mayor sea la capacidad de llenado y planos bajos o de menor número de elementos en las soluciones cuanto menor sea el tope de llenado en esta fase. En caso de que las soluciones de la otra técnica reporten mejor calidad o proyección entonces el factor de ajuste fa interviene para mover la zona de exploración de esta técnica controladamente hacia espacios cercanos a la otra a medida que avanzan las iteraciones MOS.

- **Calculo de índices Minlen y Maxlen**

Para la fase de construcción que se agregó al SGVNS, se utiliza un tamaño dinámico de la lista reducida de candidatos (RCL) que permite hacer una

exploración controlada y objetiva del espacio de soluciones. Para este fin se optó por construir dos funciones que permitieran obtener los valores de Minlen y Maxlen (líneas 20 y 21), siendo Minlen el tamaño mínimo que pudiese tener esta lista reducida de candidatos y Maxlen el tamaño máximo, obteniendo un intervalo de tamaños variado según la etapa en que se encuentre el algoritmo, y enfocando el trabajo en las zonas deseadas.

Para obtener la función de Minlen, se realiza primero una tabla de iteraciones versus valor, de forma que se obtenga el valor deseado que experimentalmente fue ajustado como se muestra en la Tabla 4.2, donde se tiene un total de 5 iteraciones para cubrir un rango de valores que inician en 1 y crecen hasta llegar a 7 a medida que las iteraciones crecen desde 1 hasta 5. Este valor de 10 iteraciones se obtiene luego de analizar cuáles eran los valores que podría alcanzar el algoritmo en una iteración MOS, o sea cuantas iteraciones promedio serían suficientes para que con el número de evaluaciones de función asignadas por MOS logre cubrir un amplio rango de valores para Minlen, incluso si esta técnica pierde participación.

Iteración	Valor MINLEN
1	1
2	2
3	4
4	5
5	7

Tabla 4.2 *Valores para Minlen SGVNS*

La ecuación obtenida para esta proyección es la siguiente:

$$Minlen = 1{,}5h - 0{,}7$$

Ecuación 4.5 *Minlen SGVNS*

Donde h es la iteración actual. La proyección de iteración con valor se muestra en la Figura 4.5, donde se puede observar el comportamiento lineal que posee la trayectoria.

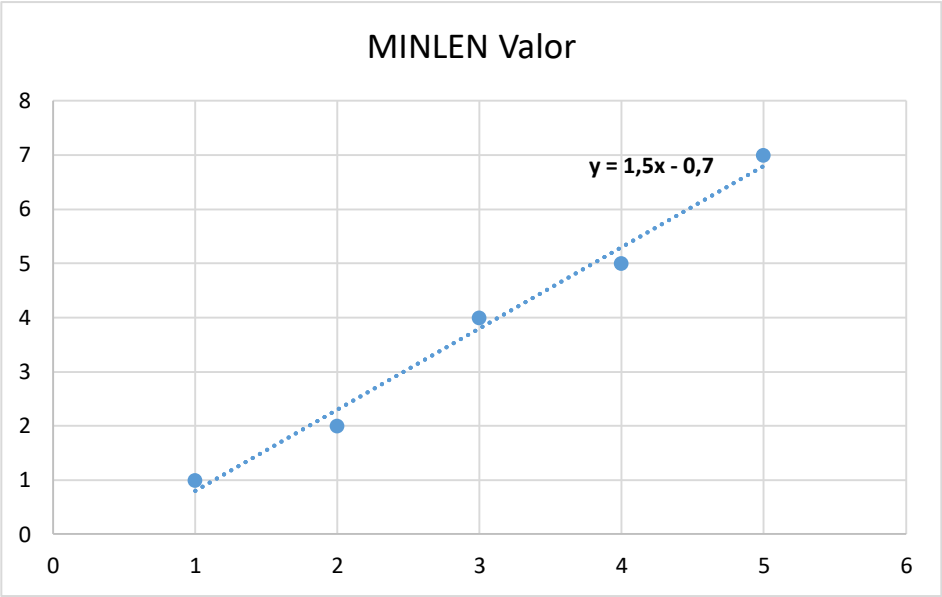

Figura 4.5 *Curva ecuación Minlen SGVNS*

De igual forma que Minlen, la magnitud Maxlen también será generada con el mismo procedimiento, los valores ajustados se muestran en la Tabla 4.3:

Iteración	Valor MAXLEN
1	4
2	6
3	8
4	10
5	15

Tabla 4.3 *Valores para Maxlen SGVNS*

La ecuación que más se ajusta a la proyección de puntos de la Tabla 4.3 se muestra en la Ecuación 4.6.

$$Maxlen = 2.4315*h + 4.7561$$

Ecuación 4.6 *Maxlen SGVNS*

Donde h es el valor de la iteración actual. Al igual que Minlen, esta función tiene un comportamiento lineal siendo este el que mejor describe la trayectoria y otorgando un valor muy aproximado para el propósito de Maxlen, que es cubrir desde un tamaño inicial de 4 en la lista reducida de candidatos donde hay una explotación más estricta dando oportunidad a elementos de mayor densidad, hasta un valor menor de 15 donde se hace una explotación más relajada en función de densidad en la fase de construcción.

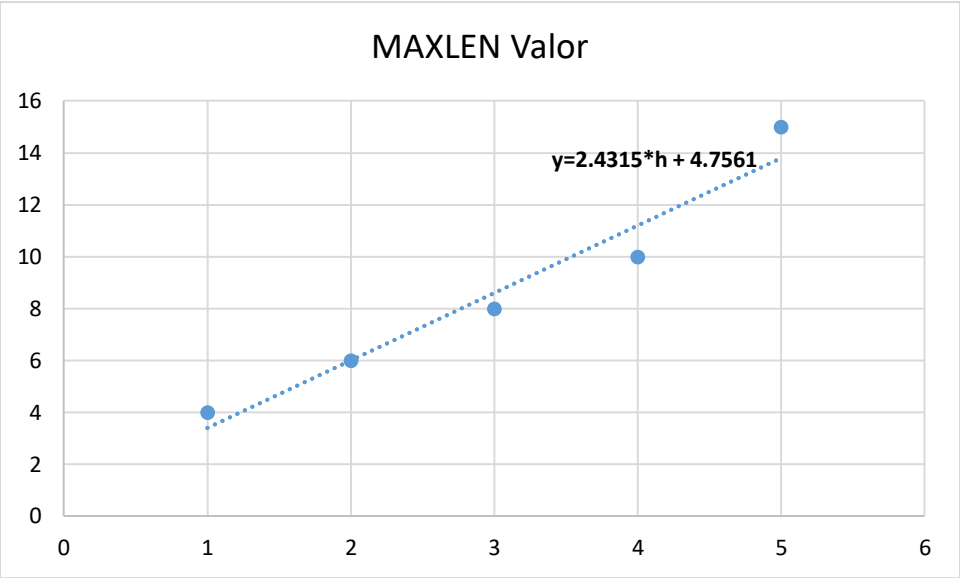

Figura 4.6 *Curva ecuación Maxlen SGVNS*

- **Fase de Construcción**

El valor de actQ calculado en la línea 22 se utiliza para controlar la exploración en la fase de búsqueda local y también para la fase de construcción con el fin de filtrar los elementos que se tienen en consideración en dicha fase y luego en la línea 23 del algoritmo SGVNS se incorpora una fase de construcción (CP por sus siglas en inglés) tipo grasp con mecanismos para realizar la elección gradual de elementos a una solución parcial, tal como lo son el filtrado de candidatos, el cálculo del tamaño de la lista reducida de candidatos (RCL por sus siglas en inglés), el criterio de calidad de un elemento, la probabilidad de elección que se otorga a estos y la forma en que se eligen para ser agregados a la mochila. En el Algoritmo 4.6 se muestra el pseudocódigo de la fase de construcción.

Algoritmo CP(Minlen, Maxlen, k, Q, X, kc, c, v, actQ,GV)

```
1.  REPETIR
2.      SI actQ < 0.9 ENTONCES
3.          ind0 = Obtener_elementos_no_pertenecen_solución(X)
4.      SI NO
5.          temp = Obtener_elementos_no_pertenecen_solución(X)
6.          ind0 = Obtener_elementos_con_frecuencia_entre(temp,Q,1, ∞)
7.      FIN SI
8.      eliminar de ind0 los elementos que no caben en la solución X
9.      n0 = número de elementos en ind0
10.     SI n0 ==0 ENTONCES
11.         Romper ciclo local
```

Algoritmo CP(Minlen, Maxlen, k, Q, X, kc, c, v, actQ,GV)

```
12.         FIN SI
13.         PARA i=1 hasta n0 ENTONCES
14.             aux(i) = densidad relativa elemento (i)
15.         FIN PARA
16.         ordenar de mayor a menor valor vector aux
17.         al = número aleatorio entero entre Minlen y Maxlen
18.         r = minimo(al,n0)
19.         Qkl = 0
20.         PARA i = 1 HASTA r HACER
21.             Qkl = Qkl + k - Q(aux(i));
22.         FIN PARA
23.         qk = vacio
24.         PARA i = 1 HASTA r HACER
25.             qkl(1,i) = k – Q(aux(i)) / Qkl;
26.         FIN PARA
27.         m = número aleatorio entero entre 1 y Qkl
28.         m = m/Qkl ;
29.         sum = 0; js = 0;
30.         PARA i = 1 HASTA r HACER
31.             sum = sum + qkl(i);
32.             SI (sum>=m) ENTONCES
33.                 js = i
34.                 romper ciclo local;
35.             FIN SI
36.         FIN PARA
37.         X(js) = 1;
38.         Agregar elemento js a GV
39.     HASTA n0 > 1
40.     Retornar X,GV;
```

Algoritmo 4.6 *Fase de Construcción*

En las líneas (2-7) del Algoritmo 4.6 realiza el proceso de preselección de candidatos para ser considerados en la fase de construcción, basándose en el vector de frecuencias, haciendo un filtrado solamente en las últimas iteraciones internas SGVNS (cuando actQ se aproxima a 1), para que la mayor parte de los llamados tengan participación de todos los elementos y en los últimos un poco más de explotación con ayuda de la información recogida en los primeros óptimos locales encontrados. Luego en la línea 8 se eliminan del vector de candidatos los que no caben en la solución para evitar procesamiento innecesario. En las líneas 13-16 se calcula la densidad relativa de estos elementos y se ordenan bajo el criterio de

mayor calidad. En la línea 17 y 18 se calcula la longitud de la lista RCL, y después entre las líneas 20 y 26 se determina la probabilidad que tiene cada elemento para ser elegido en función de las veces que ha pertenecido a un óptimo local, de forma que este proceso se encarga de dar participación un poco mayor a elementos que han sido elegidos pocas veces para explorar otras zonas del espacio de búsqueda. Finalmente, entre las líneas 27 y 37 se realiza el proceso de selección del elemento que ingresara a la solución en la iteración actual, de forma aleatoria con las probabilidades obtenidas en el proceso anterior. De esta forma se repite el procedimiento hasta que no se pueda incluir ningún elemento en la solución actual.

- **Mecanismo de Shaking o sacudida**

En la línea 24 del Algoritmo 4.5 se introduce el método usado para inducir exploración en la solución actual y como su nombre lo indica el objetivo es sacudir el cromosoma agregando o sacando h elementos en total de forma forzada. En caso de adición permite a esos elementos que nunca son considerados en los intercambios de la búsqueda de óptimos locales, ser parte de la solución al menos mientras se pasa a la reparación, y de igual forma se pueden extraer elementos engañosos que su buena densidad los mantiene arraigados en el conjunto solución, pero de verdad no pertenecen al conjunto que producen la combinación máxima de fitness. El pseudocódigo de este procedimiento se muestra a continuación en el Algoritmo 4.7.

Algoritmo Shacking(X, v, kc, h)

```
1.  PARA i=1 HASTA h HACER
2.      it = índice de elemento obtenido aleatoriamente
3.      X(it) = 1 - X(it)
4.  FIN PARA
```

Algoritmo 4.7 *Shacking*

En la variable h que se recibe como parámetro, se especifica el número de cambios que se realizarán en la sacudida, realizando movimientos de shift (línea 3), es decir si el elemento obtenido aleatoriamente (línea 2) se encuentra en la solución, entonces se extrae de la mochila, y en caso contrario se agrega. Es un algoritmo bastante simple que permite escapar de atascos en máximos locales de manera estocástica, sin ningún tipo de sesgo o criterio de calidad.

- **Algoritmo de reparación**

Se introduce una técnica heurística de reparación al algoritmo SGVNS, luego de realizar el proceso de “sacudir” la solución actual con el fin de dejar siempre la solución en un estado factible, es decir que no tenga sobrecupo y también que siempre esté al tope, por lo tanto, al terminar todo el procedimiento de reparación ninguno de los elementos que no pertenezcan a la solución podría entrar a hacer

parte de la misma. Al no realizarse ningún procedimiento de reparación, es obligación del algoritmo de búsqueda local Seq-VND realizar los intercambios en las vecindades a partir de esta solución con sobrecupo y obtener finalmente una solución que sea válida, algo que supone un gasto innecesario de cómputo ya que se tendría que reparar la solución realizando intercambios en el algoritmo de búsqueda local, para lo cual no está diseñado.

Algoritmo Reparación (X, kc, c, v, maxin)

```
1.  ind1 = Obtener_elementos_pertenecen_a_solución(x)
2.  n1 = número de elementos en (ind1)
3.  w = peso (X)
4.  auxFx = Infinito
5.  auxind = 0
6.  MIENTRAS w > kc HACER
7.      PARA i=1 HASTA n1 HACER
8.          temp = densidad relativa elemento ind1(i)
9.          SI temp < auxFx ENTONCES
10.             auxfx = temp
11.             auxind = i
12.         FIN SI
13.     FIN PARA
14.     X(auxind) = 0
15.     Actualizar peso w
16.     Actualizar ind1
17. FIN MIENTRAS
18. ind0 = Obtener_elementos_no_pertenecen_a_solución(x)
19. n0 = número de elementos en n0
20. MIENTRAS (n0 > 0 && w < kc) HACER
21.     aux = ()
22.     PARA i=1 HASTA n0 HACER
23.         aux(i) = densidad relativa elemento ind0(i)
24.     FIN PARA
25.     ordenar de mayor a menor valor vector aux
26.     in = 0
27.     PARA i=1 HASTA n0 HACER
28.         p = índice de elemento (aux(i))
29.         SI w+v(p) <= kc ENTONCES
30.             X(p)=1
31.             w = w + v(p)
32.             in = in + 1
```

```
33.                 SI in == Maxin ENTONCES
34.                     ROMPER CICLO LOCAL
35.                 FIN SI
36.             FIN SI
37.         FIN PARA
38.         actualizar ind0
39.         n0 = número de elementos en (ind0)
40.     FIN MIENTRAS
```

Algoritmo 4.8 *Reparación*

Al igual que en la solución inicial, en el algoritmo SGVNS (Algoritmo 4.5), se maneja el ingreso segmentado de elementos después de calcular la densidad relativa, este número de elementos es determinado por la variable Maxin que se recibe como parámetro, junto con las variables de instancia y la solución a la cual se va a realizar la reparación que está representada por la variable X. Inicialmente el algoritmo identifica cuales son los elementos que se encuentran en la mochila (línea 1) y luego calcula la cantidad de estos (línea 2), como también obtiene el peso del vector X para controlar la factibilidad (línea 3), luego se ingresa al primer bucle (línea 6-17) donde el objetivo es vaciar la mochila hasta el punto que no tenga objetos de más, sacando el elemento que menor beneficio aporta o dicho de otra forma el que menos densidad posee. Luego se obtienen los elementos que no pertenecen a la solución (línea 18) y se calcula el número de estos (línea 19) y se ingresa al segundo bucle (líneas 20-40), donde el objetivo es dejar la mochila lo más llena posible, de forma que no quepa ningún elemento después de terminar este proceso. Primero se calcula la densidad relativa de los elementos que no pertenecen a la solución (líneas 22-24), luego se hace un ordenamiento de mayor a menor de estos valores (línea 25), y se ingresan Maxim elementos a la solución actual o menos, si la restricción de peso no permite el ingreso de más elementos.

- **Algoritmo de Búsqueda local Seq-VND [20]**

Para la búsqueda local se utiliza un algoritmo secuencial de vecindad variable descendente denominado seq-VND (línea 26 del Algoritmo 4.5), que busca a través de dos estructuras de vecindad que implementa y con las cuales espera llegar al mejor óptimo local de la solución actual que pueda encontrarse producto del trabajo conjunto de estas estructuras. Con su comportamiento no determinista, aporta cualidades de exploración y explotación necesarias para encontrar siempre una mejor solución que la anterior, pero por caminos diferentes.

En la rutina seq-VND (Algoritmo 4.9) se comienza a trabajar siempre con un vector solución que recibe como parámetro X desde el algoritmo principal, realizando en

primera instancia una exploración en la primera vecindad swap1 o N_1 buscando entre sus vecinos el primero que mejore la solución actual (línea 7) y llamando al operador add (en la línea 8) operador que se describe más adelante, siempre después de este paso. Luego se actualiza la solución actual con esta nueva encontrada (línea 9) y se devuelve al primer vecindario (N_1 o swap1, cuando se encontró una mejor solución en el vecindario actual), y si no encuentra un vecino que mejore la solución entonces se mueve a la siguiente vecindad, que para la implementación en este trabajo es la segunda vecindad N_2 o swap 2, y además se actualiza el vector de frecuencias Q con el óptimo local encontrado para la estructura de vecindad de la cual se está saliendo. Cuando esta segunda vecindad no puede encontrar alguna mejora entonces se termina la ejecución del algoritmo y se retorna la mejor solución encontrada.

Algoritmo Seq-VND (X, c, p, w, Qf, Maxv,GO)

```
1.  h = 1
2.  MIENTAS h ≤ 2 HACER
3.      SI fitness(X) ≥ Maxv ENTONCES
4.          RETORNAR X
5.      FIN SI
6.      Si X No es óptimo local en la estructura de vecindad Nh ENTONCES
7.          X' ← Primera solución Nh X tal que f(X') > f(X)
8.          X'' ← add (X', c, p, w)
9.          X ← X''
10.         h ← 1
11.     SI NO
12.         h ← h + 1
13.         Actualizar vector de frecuencias Qf con X
14.     FIN SI
15. FIN MIENTAS
16. RETORNAR X,Qf
```

Algoritmo 4.9 *Seq-VND*

Las estructuras de Vecindad del Seq-VND (ver línea 6), en la implementación que se realizó para este proyecto se incluye una nueva estructura de vecindad llamada swap 2, y también se elimina la vecindad shift de la lista de vecindades que estaba en la propuesta original [20], se incorporó dentro de las otras dos vecindades swap1 (Algoritmo 4.11) y swap2 (Algoritmo 4.14) el operador add (Algoritmo 4.10), invocando éste después de que termina la búsqueda en cada vecindad. La descripción del operador add y las estructuras de vecindad que quedaron en la implementación de este proyecto se muestra a continuación.

- **Operador add**

Consiste en agregar los elementos que sean necesarios para terminar de llenar la mochila, luego que una solución es explorada dentro de una estructura de vecindad determinada y haya quedado espacio disponible para uno o varios elementos. En el Algoritmo 4.10 se muestra el pseudocódigo de esta rutina. Primero, se inspecciona que la mochila no esté al tope de capacidad (línea 2), después si hay objetos que puedan incluirse según el espacio que se encuentra disponible (línea 8) y finalmente, se hace la adición (líneas 5-21) bajo el criterio de densidad relativa, de mayor a menor valor. La ejecución total de este procedimiento se completa muy pocas veces llegando solamente hasta las líneas 3 o 9, debido a que siempre los intercambios en las dos vecindades swap1 y swap2, tratan de hacerse con ítems de pesos similares, dejando muy pocas veces espacio para otros elementos, y cuando esto sucede son pocos los candidatos para ser agregados, sin embargo, el movimiento add garantiza que siempre la solución está al tope de la capacidad máxima permitida.

Algoritmo add (X, c, p, w)

```
1.  wx = peso (X)
2.  SI wx == c ENTONCES
3.      RETORNAR
4.  FIN SI
5.  MIENTRAS 1
6.      ind0 = Obtener_elementos_no_pertenecen_solución(X)
7.      eliminar de ind0 los elementos que no caben en la solución X
8.      n0 = número de elementos en ind0
9.      SI n0 == 0 ENTONCES
10.         RETORNAR X
11.     FIN SI
12.     PARA i=1 HASTA n0 HACER
13.         aux (i) = densidad relativa elemento (ind0(i))
14.     FIN PARA
15.     ordenar de mayor a menor el vector aux
16.     X (aux (1)) = 1
17.     wx = wx + w ( aux(1) ) // contabiliza el peso del nuevo elemento
18.     SI (wx == c) ENTONCES
19.         romper ciclo local
20.     FIN SI
21. FIN MIENTRAS
22. RETORNAR X
```

Algoritmo 4.10 *Add*

- **Estructura de vecindad Swap**1

En el Algoritmo 4.11 se puede observar el pseudocódigo de la estructura de vecindad Swap1 que consiste en tomar un elemento que no pertenece a la solución y cambiarlo por uno que esté dentro de la mochila, manteniendo la solución factible, ya que no se realiza ningún procedimiento de reparación luego de este paso. En lo posible los elementos intercambiados deben tener un peso similar para evitar llamados al operador *add.*

Algoritmo SWAP1 (X, c, p, w, Qf, actQ)

```
1.  may = 0
2.  SI actQ < 0.85 ENTONCES
3.      ind0 = Obtener_elementos_no_pertenecen_solución (X)
4.  SINO
5.      temp = Obtener_elementos_no_pertenecen_solución (X)
6.      ind0 = Obtener_elementos_con_frecuencia_entre (temp, Qf, 1, ∞)
7.  FIN SI
8.  n0 = número de elementos en (ind0)
9.  perm = generar_permutacion_hasta (n0)
10. PARA i=1 HASTA n0 HACER
11.     a = ind0 (perm (i))
12.     ind1 = obtener elementos intercambiables con elemento a
13.     aux = número de elementos en (ind1)
14.     Si aux == 0 ENTONCES
15.         Saltar a siguiente iteración i (continuar)
16.     FIN SI
17.     perm1 = generar_permutacion_hasta (aux)
18.     PARA j = 1 HASTA aux HACER
19.         b = ind1 (perm1 (j))
20.         X(b) = 0
21.         p0= Beneficio_agregar_item (X, a, p)
22.         p1= Beneficio_agregar_item (X, b, p)
23.         SI p0 > p1 && p0 > may ENTONCES
24.             may = p0
25.             p1M = p1
26.             indout = b
27.         FIN SI
28.         X (b)=1
29.     FIN PARA
30.     SI may > 0 ENTONCES
31.         X (a)=1
```

```
32.             X (indout)=0
33.             actualizar peso
34.             actualizar Fitness
35.         FIN SI
36.     FIN PARA
37.     RETORNAR X,s
```

Algoritmo 4.11 *Swap1*

En algoritmo swap 1, se reciben como parámetros la solución actual X, las variables de instancia (c, p, w), el vector de frecuencias Q y el controlador de exploración *actQ* desde el algoritmo Seq-VND. En la línea 1 se inicializa la variable auxiliar *may* que ayuda a guardar el mayor beneficio de intercambio encontrado, y la variable *s* que sirve para saber si hubo mejora o no, si su valor es cero al retornar entonces no se pudo realizar ningún intercambio, y 1 en el caso contrario. Entre las líneas 2 y 7 la variable de entrada *actQ* determina cuáles elementos que no pertenecen a la solución serán considerados para realizar los intercambios. En la línea 9 se realiza una permutación del orden en que serán analizados los elementos candidatos, para inducir aleatoriedad en el proceso. Desde la línea 10 hasta la 36 se hace todo el análisis de posibles intercambios, comenzando por tomar cada elemento que no pertenece a la solución en el vector ind0, y se obtienen los elementos intercambiables con el elemento actual y se guardan en el vector ind1, luego aleatoriamente revisar cuál de los intercambios mejora la solución actual (líneas 18-29), y finalmente realizar el intercambio que más beneficio puede aportar (líneas 30-36).

En el Algoritmo 4.12 se muestra la rutina utilizada en la línea 6 del procedimiento *Swap1* para encontrar los elementos que tienen una frecuencia entre el rango especificado, en este caso entre *min* e infinito (∞), o sea con valores mayores que cero.

```
Algoritmo Obtener_elementos_con_frecuencia_entre (X, Qf, min, max )
1.  itms = ()
2.  PARA i=1 HASTA número de elementos en X HACER
3.      SI Q(X(i))>=min && Q(X(i))<=max ENTONCES
4.          Agregar i al vector itms
5.      FIN SI
6.  Retornar itms
7.  FIN PARA
```

Algoritmo 4.12 *Frecuencia Entre*

Con el fin de inducir aleatoriedad en el orden que se escogen los candidatos para los intercambios, se realiza una permutación con los índices de los elementos

evitando que se explore una estructura de la misma forma para una determinada solución. En el Algoritmo 4.13 se muestra el procedimiento que genera la permutación hasta un valor especificado.

```
Algoritmo Generar_permutacion_hasta( n )
1.    perm = ()
2.    Aux = ()
3.    PARA i=1 HASTA n HACER
4.        Aux(i) = i
5.    FIN PARA
6.    PARA i=1 HASTA n HACER
7.        rand = número entero aleatorio entre 1 y (n-i+1)
8.        Perm(i) = aux(rand)
9.        aux(rand) = ()
10.   FIN PARA
11.   RETORNAR perm
```

Algoritmo 4.13 *Generar Permutación*

- **Estructura de vecindad Swap 2**

En esta estructura también se realiza la acción de intercambio como en *Swap1*, pero de dos (2) elementos, es decir se sacan dos elementos de la mochila y luego se agregan otros dos elementos que no pertenecen a la solución actual. El pseudocódigo se muestra en el Algoritmo 4.14.

```
Algoritmo SWAP 2 (X, kc, c ,v ,Qf, actQ,GV)
1.    may = 0
2.    rangosup = 12; rangoinf = 30
3.    numelX = número de elementos en X
4.    LimSup = numelX+ rangosup
5.    Liminf = numelX- rangoinf
6.    SI (LimSup>n) ENTONCES
7.        LimSup = n
8.    FIN SI
9.    SI  (Liminf<1)  ENTONCES
10.       Liminf = 1
11.   FIN SI
12.   SI actQ < (0.85*(número aleatorio entre 0 y 1)) ENTONCES
13.       ind0 = Obtener_elementos_no_pertenecen_solución(X)
14.       ind0 = intersección entre (ind0 y GV(1:LimSup))
15.       temp= Obtener_elementos_pertenecen_solución(X)
16.       maxQ = máximo valor en vector(Qf)
```

Algoritmo SWAP 2 (X, kc, c ,v ,*Qf*, actQ,GV)

```
17.         ind1= Obtener_elementos_con_frecuencia_entre (temp,Qf,1,maxQ-1)
18.         temp= intersección entre(GV(Liminf:numelX) y temp)
19.         ind1 = unión entre (ind1 y temp)
20.     SINO
21.         temp= Obtener_elementos_no_pertenecen_solución(X)
22.         ind0 = Obtener_elementos_con_frecuencia_entre (temp,Qf,1, ∞)
23.         temp= intersección entre(GV(1:LimSup) y temp))
24.         ind0 = unión entre (ind0 y temp)
25.         ind1= Obtener_elementos_pertenecen_solución(X)
26.         ind1 = intersección entre(ind1,GV(Liminf:n))
27.     FIN SI
28.     n0 = número de elementos en ind0
29.     n1 = número de elementos en ind1
30.     Maxv0 = obtener mayor peso, de elementos que pertenecen a la solución
        X
31.     Minv0 = obtener menor peso, de elementos que pertenecen a la solución
        X
32.     perm0 = permutación de números de 1 hasta (n0)
33.     perm1 = permutación de números de 1 hasta (n1)
34.     PARA i=1 HASTA n0-1 HACER
35.         a0 = ind0 ( perm0 ( i ) )
36.         PARA j=i+1 HASTA n0 HACER
37.             b0 = ind0(perm0(j))
38.             sw0 = (v(a0)+v(b(0)))
39.             pw = sw0 /2;
40.             SI pw > Maxv0 || pw < Minv0 ENTONCES
41.                 Saltar a siguiente iteración j
42.             FIN SI
43.             PARA q=1 HASTA n1-1 HACER
44.                 a1 = ind1(perm1(j))
45.                 X(b) = 0
46.                 PARA k=q+1 HASTA n1 HACER
47.                     b1 = ind1(perm1(j))
48.                     sw1= v(a1)+v(b1);
49.                     SI (tw-sw1+sw0) > kc || sw0 < sw1 ENTONCES
50.                         Saltar a siguiente iteración k
51.                     FIN SI
52.                     p0=Beneficio_agregar_item(X,a0,c)+
                            Beneficio_agregar_item(X,b0,c)
```

```
Algoritmo SWAP 2 (X, kc, c ,v ,Qf, actQ,GV)
53.                         p1=Beneficio_agregar_item(X,a1,c)+
                                Beneficio_agregar_item(X,b1,c)
54.                         SI p0>p1 ENTONCES
55.                             X(a0) = 1
56.                             X(b0) = 1
57.                             X(a1) = 0
58.                             actualizar peso
59.                             actualizar Fitness
60.                             retornar X
61.                         FIN SI
62.                     FIN PARA
63.                 FIN PARA
64.             FIN PARA
65.         FIN PARA
66.         RETORNAR X
```

Algoritmo 4.14 *Swap2*

El procedimiento de *swap 2* tiene algunas similitudes con *Swap1*, como el uso del vector de frecuencias para la selección de candidatos para intercambios, las permutaciones para agregar aleatoriedad (líneas 32 y 33) y también el control con *actQ* para la selección de elementos, aunque en este algoritmo se filtran objetos que están tanto dentro fuera de la mochila, ya que si se observa en la disyuntiva de la línea 12, todos los elementos que están en la solución (*ind1*) y los que no están (*ind0*) se obtienen con un criterio diferente según sea el valor de *actQ*. En caso de que actQ sea afirmativa el condicional los elementos de ind0 serán aquellos que no pertenecen a la solución y se encuentran en el vector GV entre las posiciones 1 y el límite superior de exploración limSup, y los elementos de ind1 serán la unión del grupo de elementos que pertenecen a la solución y tienen frecuencia mayor que 1 y menor que la frecuencia máxima, y el grupo de elementos que pertenecen a la solución y están en el vector GV entre las posiciones Liminf y el número de elementos en la solución actual. De forma similar se calculan los elementos en caso del condicional negativo. En las líneas 30 y 31 se obtienen los valores de *Maxv0* y *Minv0,* que son el mayor y el menor peso respectivamente, de los objetos que pertenecen a la solución, y estos valores son usados para filtrar (línea 40) o controlar el peso de los elementos que serán intercambiados y ahorrar comparaciones innecesarias. Entre las líneas 34 y 65, se realiza todo el proceso de intercambios cuando ya se tienen los candidatos, que se compone de 4 ciclos anidados de forma que se pueda hacer tratamiento a cada uno de los 4 elementos involucrados en un intercambio de 2 por 2. En el segundo y el cuarto ciclo se aplican dos filtros que disminuyen aún más el espacio de búsqueda, descartando combinaciones de

elementos que no tienen potencial. Esta heurística retorna la primera mejora que se encuentra (líneas 54-60), y cumpla con las condiciones de factibilidad, pero aun así teniendo total capacidad de encontrar intercambios diferentes para una misma entrada dado a que es un procedimiento no determinístico debido a las permutaciones creadas en las líneas 32 y 33 para que los elementos seleccionados siempre sean considerados aleatoriamente.

- **Actualización de variables**

Al terminar la fase de búsqueda local del algoritmo SGVNS (Algoritmo 4.5), se realiza la actualización del vector de frecuencias en la línea 27 con el óptimo local encontrado en la fase anteriormente ejecutada, también se actualiza el valor de Fprom en la línea 28 con el fitness de esta solución actual debido a que el algoritmo principal MOS necesita que sus técnicas subordinadas bajo algún criterio reporten el desempeño que estas tuvieron, lo cual para este enfoque se considera conveniente llevar un promedio del fitness o del valor de la evaluación de la función objetivo de cada óptimo local obtenido en determinadas fases de cada algoritmo subordinado.

Entre las líneas 29 y 35 se realiza la actualización de la mejor solución global encontrada, el vector greedy GV con el que fue encontrada esta solución, la verificación de si se alcanza el máximo reportado y el reinicio del iterador global, si se mejora la solución actual y si no incrementa el iterador global *h* en la línea 37. Cuando se agotan las evaluaciones de función se guarda la iteración actual en la variable *ei*, como se puede observar en la línea 42, para continuar la ejecución en este mismo punto cuando se vuelva a ejecutar este algoritmo.

4.2.2 Técnica subordinada GRASPr

En el método GRASPr propuesto en este trabajo, se realizaron modificaciones bastante amplias respecto al algoritmo original [14], comenzando por la eliminación de algunas variables de control como beta, gamma, m, u y algunas otras que pueden encontrarse en la especificación del mismo en [28], debido a que no fue necesaria su función con las adecuaciones que fueron realizadas. El cálculo de MinLen, Maxlen, sigma, el valor de las iteraciones k del bucle interno, y también la fase de búsqueda local se implementan de forma totalmente diferente debido a que los resultados no eran los esperados, y con la información especificada en el artículo no fue suficiente para hacer una réplica de su trabajo. La actualización de la solución actual se elimina del algoritmo y se traslada al algoritmo principal MOS, para que allí se realice este procedimiento para ambas técnicas. GRASPr es un algoritmo de estado simple que posee la característica de inicio múltiple, es decir que al inicio de cada iteración se tiene una nueva solución, que se obtiene a partir de la fase de construcción y se explota en la fase de búsqueda local.

Se representa en un diagrama de flujo el algoritmo adaptado GRASPr (ver Figura 4.7) con las modificaciones realizadas al algoritmo original, mostrando en color verde aquellas fases o funcionalidades que fueron agregadas o modificadas.

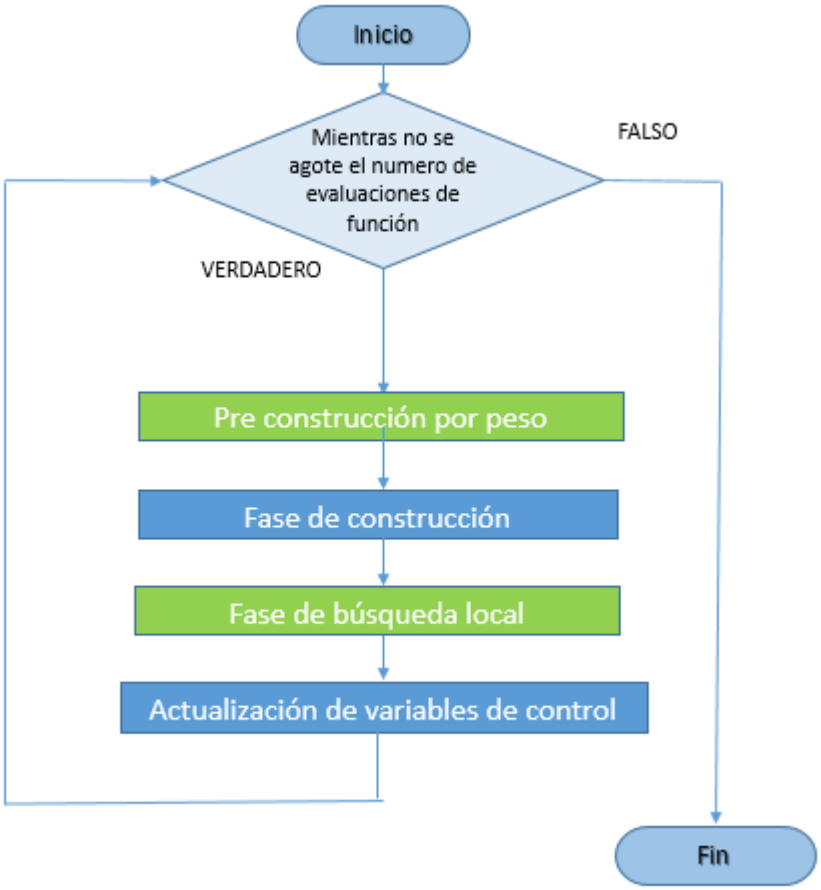

Figura 4.7 *Diagrama flujo adaptación GRASPr*

El pseudocódigo de GRASPr se muestra en el Algoritmo 4.15.

Algoritmo GRASPr (X, Xbest, c, p, w, Qf, FES, Maxv, ei , GV, fa, bestGV)

1. σ = 5
2. Fprom = 0;
3. Ind0 = obtener elementos que no pertenecen a solución parcial X
4. Ind0 = ordenar elementos en ind0, menor a mayor en función del peso
5. **MIENTRAS** (No se agoten las FES asignadas) **HACER**
6. **Para** k=1+ei **HASTA** σ **HACER**
7. X’ = X
8. GV’ = GV

Algoritmo GRASPr (X, Xbest, c, p, w, Qf, FES, Maxv, ei , GV, fa, bestGV)

```
9.          reg = número aleatorio entre 0 y 0.03
10.         MIENTRAS hallan elementos en ind0 ENTONCES
11.             i = primer elemento en ind0
12.             SI (peso de X'+ w(i) ) > (c * 0.87 + reg + fa)
                ENTONCES
13.                     Romper ciclo local
14.             FIN SI
15.             agregar elemento i a X'
16.             agregar elemento i a GV'
17.             eliminar elemento i de ind0
18.         FIN MIENTRAS
19.         MinLen = redondear(3,51*k - 3,5)
20.         MaxLen = redondear(1.2143*k*k - 0.1857*k + 3.6)
21.         actQ = (k)/ σ;
22.         [X',GV'] = CP(Minlen,Maxlen,k,Q,X',c,p,w,actQ,GV');
23.         X' = LS(X',c,p,w,actQ,maxv,GV');
24.         Actualizar vector de frecuencias Qf con X'
25.         Actualizar valor de Fprom con X'
26.         SI(f(X')> f(Xbest) ENTONCES
27.             Xbest = X'
28.             bestGV = GV'
29.             SI f(X') >= maxv ENTONCES
30.                 RETORNAR Xbest,Qf,ei,bestGV,Fprom
31.             FIN SI
32.         FIN SI
33.     FIN PARA
34.     ei = 0
35. FIN MIENTRAS
36. Asignar a ei el valor h en el cual quedó
37. RETORNAR Xbest,Qf,ei,bestGV,Fprom
```

Algoritmo 4.15 *GRASPr*

Los parámetros que recibe el algoritmo GRASPr son los mismos que recibe el algoritmo SGVNS, especificados en la sección 4.1.1 Se realiza esta configuración con el fin de agregar modularidad al algoritmo MOS implementado, de forma que cada técnica que se considere añadir solo tenga que recibir estas entradas sin importar su lógica interna y esta misma debe gestionar internamente estos parámetros a su manera.

En la línea 1 se inicializa la variable sigma, que determina el número de iteraciones que tendrá el bucle interno del algoritmo y en la línea 2 se declara la variable Fprom encargada de llevar el promedio de fitness de óptimos locales encontrados durante todas las iteraciones que se alcancen a ejecutar con las FES asignadas. Luego se obtiene el vector ind0 con los índices de los elementos ordenados de menor a mayor respecto a su peso, como insumo para la fase de pre construcción por peso (líneas 10-18), en la cual también se construye además de la solución parcial, el vector GV' que se utiliza en la fase de búsqueda local. Dentro del bucle principal que es controlado por el número de evaluaciones de función asignadas por MOS, se encuentra desde la línea 6 hasta la 33 el bucle interno GRASP que ejecuta el núcleo del algoritmo.

- **Fase de pre construcción por peso**

El funcionamiento de la fase de pre construcción por peso (líneas 7-18) en GRASPr es similar al descrito en la técnica SGNVS, a diferencia que en esta técnica la estrategia de llenado se realiza hasta un tope base de 87% de la capacidad con un índice de variación hacia arriba de 3% representado por la variable reg, subiendo el tope aleatoriamente para permitir explorar planos a un término medio-bajo de número de elementos. De igual forma el factor de ajuste *fa*, infiere en el mismo sentido que reg para que esta fase agregue más o menos elementos, permitiendo aumentar la exploración hacia otros planos de soluciones a medida que el desempeño de las técnicas subordinadas lo vayan demandando.

- **Calculo de índices Minlen y Maxlen**

Al igual que en SGVNS en el algoritmo GRASPr se tiene un tamaño dinámico para la lista reducida de candidatos en la fase de construcción, y este es controlado por los valores de Minlen y Maxlen que son proporcionados desde el algoritmo principal GRASPr en las líneas 19 y 20.

En este caso el valor de Minlen tiene un valor ascendente a medida que se incrementan las iteraciones internas(k) de GRASPr Tabla 4.4, empezando con un valor de 1 al inicio de la ejecución y creciendo gradualmente hasta 16 a medida que las iteraciones se aproximan al número de 10, permitiendo en este punto a la lista reducida de candidatos tener un valor bastante amplio en el mínimo para la adición de elementos con poca posibilidad de acceder a la solución, debido a unas situaciones especiales donde a pesar de que algunos elementos no están entre los punteros en densidad relativa, si pertenecen a la solución máxima reportada.

iteración	Valor MINLEN
1	1
2	3
3	6
4	10
5	15

Tabla 4.4 *Valores para Minlen GRASPr*

La ecuación obtenida para esta proyección es la siguiente:

$$Minlen = 3{,}51*k - 3{,}5$$

Ecuación 4.7 *Minlen GRASPr*

Donde k es la iteración actual. La proyección de iteración contra valor se muestra en la Figura 4.8, La cual una trayectoria lineal es la que más se ajusta a la unión de sus puntos

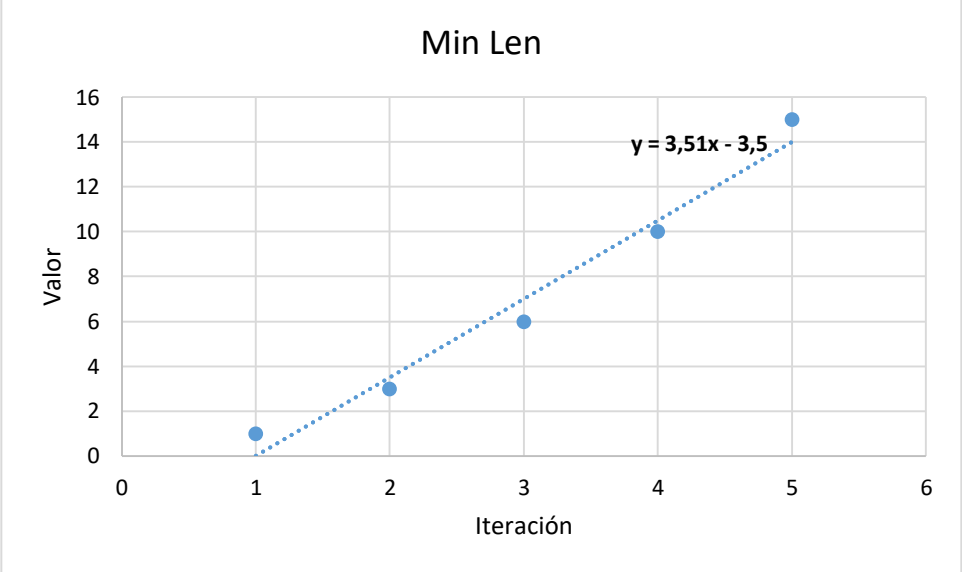

Figura 4.8 *Curva ecuación Minlen GRASPr*

Para Maxlen también se realiza la tabulación de valores para iteración y valor, donde el valor también presenta una tendencia ascendente como en Minlen, a medida que avanzan las iteraciones, esto se muestra en la Tabla 4.5.

Iteración	Valor MAXLEN
1	5
2	8
3	12
4	25
5	32

Tabla 4.5 *Valores para Maxlen GRASPr*

Una función lineal es la que en este caso menos margen de error tiene y más se ajusta a los valores de la Tabla 4.5, obteniendo la siguiente ecuación:

$$Maxlen = 1.2143*k*k - 0.1857*k + 3.6$$

Ecuación 4.8 *Maxlen GRASPr*

La trayectoria de la Ecuación 4.8, y los puntos de la Tabla 4.5, se ilustran en la Figura 4.9, donde se puede apreciar una correspondencia muy alta.

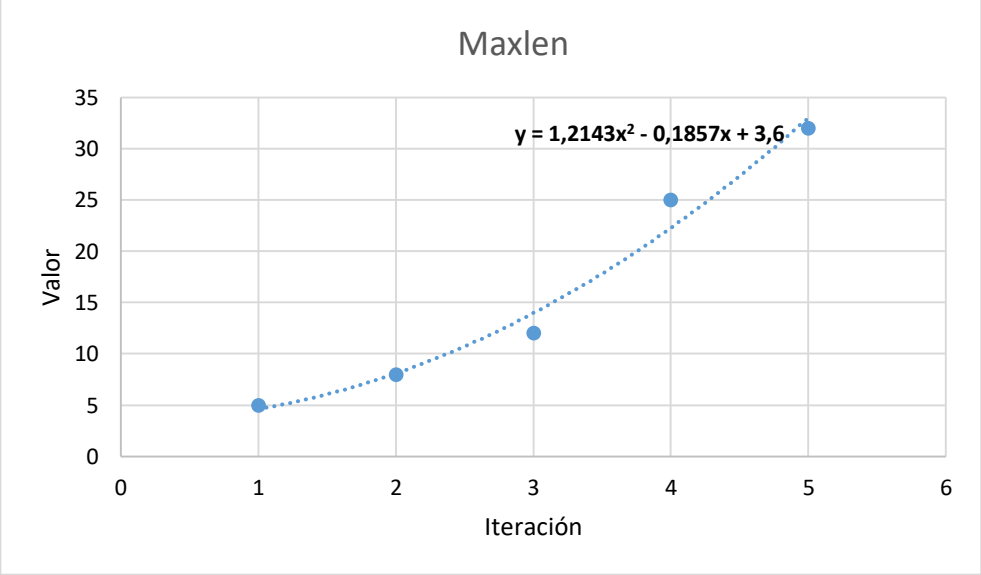

Figura 4.9 *Curva ecuación Maxlen GRASPr*

- **Fase de Construcción (CP)**

La fase de construcción GRASPr utilizada en la línea 22 es la misma que se introdujo en la técnica SGVNS (ver Fase de Construcción), siendo esta sección originalmente del algoritmo GRASPr propuesto en [14].

- **Búsqueda Local**

La fase de búsqueda local (LS por sus siglas en inglés) llamada en la línea 23 del algoritmo GRASPr, tiene como finalidad realizar un proceso de explotación principalmente para encontrar el óptimo local para la solución parcial que fue

construida en la fase anterior. Para la versión de GRASPr implementada en este trabajo se implementa de forma diferente el proceso de búsqueda local seleccionando de forma aleatoria y equitativa el orden en que se ejecutan las estructuras de vecindad, utilizando las rutinas de swap1, swap2 y add utilizadas también en la versión de SGVNS de este trabajo. En el Algoritmo 4.16 se muestra el pseudocódigo de la rutina de búsqueda local.

```
Algoritmo LS (X', kc, c, v, actQ, Maxv, GV)
1.      MIENTRAS (X no sea optimo local) HACER
2.          al = número aleatorio entre 0 y 1
3.          SI al > 0.5 ENTONCES
4.              X = SWAP 1 (X, kc, c, v, Q, actQ)
5.              X = add(X',peso(X),kc,v)
6.              X= SWAP 2 (X, kc, c, v, Q, actQ,GV)
7.              X = add(X,peso(X),kc,v)
8.          SINO
9.              X= SWAP 2 (X, kc, c, v, Q, actQ,GV)
10.             X = add(X,peso(X),kc,v)
11.             X = SWAP 1 (X, kc, c, v, Q, actQ)
12.             X = add(X',peso(X),kc,v)
13.         FINSI
14.     FIN MIENTRAS
15.     RETORNAR X
```

Algoritmo 4.16 *Búsqueda Local*

Para la búsqueda local se implementa una estructura secuencial, ejecutando un ciclo permanente mientras que la solución actual no sea un óptimo local (líneas 1-14) para la composición de búsqueda propuesta y que funciona como sigue: Primero se realiza el llamado a la rutina *swap 1(o swap 2)*, que retornara el primer intercambio que mejore la solución entrante, luego se hace un llamado al operador *add* para terminar de llenar la mochila en caso que algún intercambio haya dejado espacio para un elemento de menor tamaño, después se procede a realizar intercambios en la vecindad *swap 2(o swap 1)*, también retornando el mejor intercambio que mejore la solución entrante, y se vuelve a llamar al operador *add* una vez más con el mismo fin que el anterior llamado a esta función. Si se logra mejorar la solución en al menos uno de los cuatro procesos anteriormente descritos se inicia una nueva iteración hasta que no haya ninguna mejora por parte de estos.

- **Actualización de variables**

En la líneas 24 y 25 del algoritmo GRASPr, se prosigue con la actualización del vector de frecuencias Qf y vector de fitness promedio y luego si se mejora la solución actual se ejecuta entre las líneas 26 y 32 se actualiza la mejora solución encontrada

X_{best} y el vector greddy bestGV con el vector GV’ que genero esta solución, como también se da fin a la ejecución se alcanza el máximo que se tiene como objetivo indicado en la variable maxv. Cuando se agotan las evaluaciones de función se actualiza en la línea 36 el estado de la iteración actual en la variable ei, para simular la continuidad en las posteriores llamadas a este algoritmo.

5 EXPERIMENTACION

En este capítulo se describen las instancias usadas para la experimentación y también se presentan los resultados obtenidos por la propuesta desarrollada en este trabajo, haciendo una comparación con otros algoritmos del estado del arte, que han resuelto el problema de la mochila cuadrática binaria para las mismas instancias de alta dimensionalidad y se encuentran entre los mejores hasta la fecha.

5.1 Instancias del problema 0-1 Quadratic knapsack problema

Una instancia del problema 0-1 QKP proporciona toda la información necesaria para resolver dicho problema, es decir, la capacidad de la mochila, el número de objetos, la matriz de beneficios (contiene el valor propio aportado por cada elemento y el valor asociado con los demás elementos) y el vector de pesos. Cabe resaltar que las instancias del 0-1 QKP son generadas aleatoriamente de la siguiente manera: Los pesos son enteros tomados de una distribución uniforme entre el intervalo [1,50], las restricciones de capacidad son un número entero tomado de la distribución uniforme entre 50 y la sumatoria de los pesos de los ítems y finalmente los coeficientes objetivos, es decir, los valores se eligen aleatoriamente entre [1,100].

5.1.1 Tipos de Instancias del problema 0-1 QKP

El problema de la mochila cuadrática binaria está dividido en cuatro grupos de instancias. Estas se diferencian por su densidad, que es la cantidad de valores no ceros que contiene la matriz de beneficios, lo que representa el aporte de un elemento con relación a los demás elementos. Por lo tanto, se clasifican de la siguiente manera: 25%, 50%, 75% y 100%. En la **Figura 5.1** Tipos de Instancias se pueden observar a nivel macro las matrices de beneficios para los 4 grupos de instancias de 100 elementos, donde se puede apreciar claramente los puntos grises para los valores no cero y los puntos blancos para los valores iguales a cero en dichas matrices.

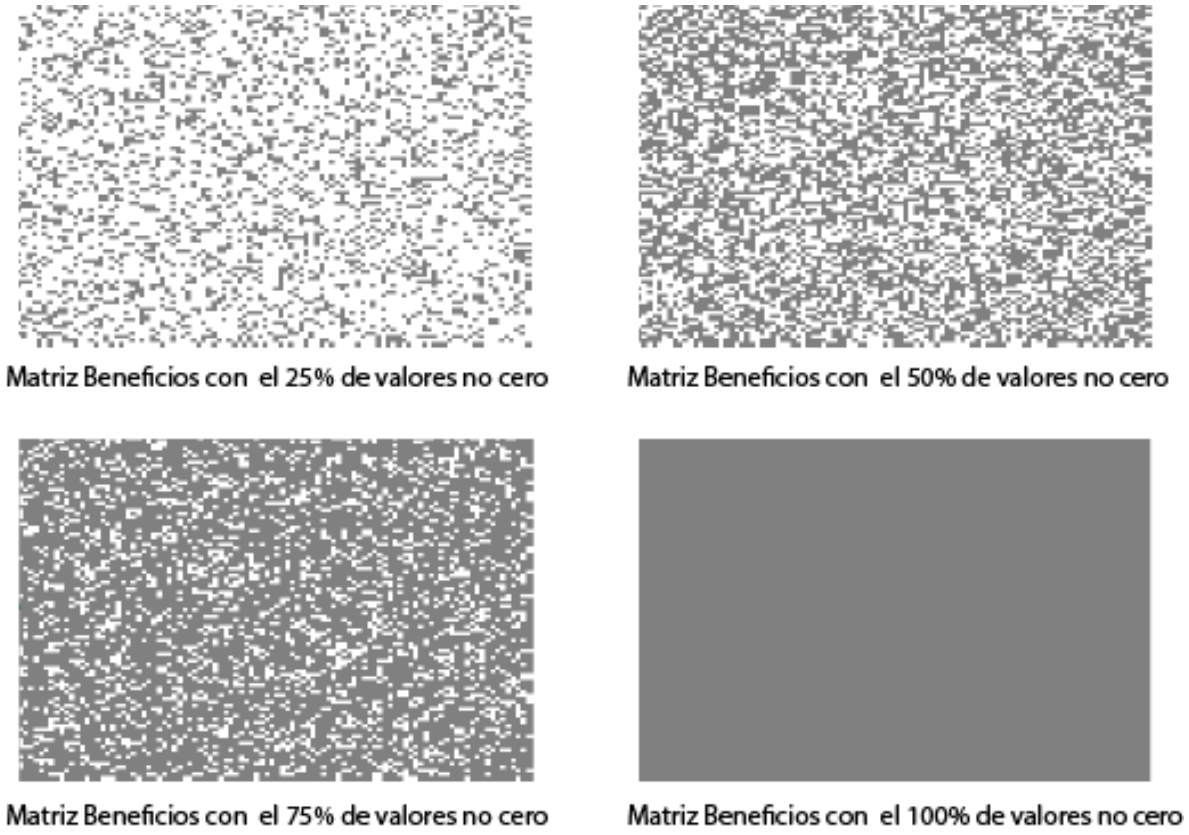

Figura 5.1 *Tipos de Instancias*

Con respecto a las instancias se pudo constatar que no hay una correlación, entre el tipo de instancia y la complejidad de la misma, debido a que por ejemplo hay instancias que pertenecen al primer grupo con 25% de valores no cero en la matriz de beneficios, y su dificultad es alta respecto a instancias que tienen un índice de densidad o valores no cero mayores. De la misma forma se pudo observar que hay instancias con menos número de elementos (1000) que son más difíciles de resolver que algunas que tienen más número de elementos (2000). También se observa que la capacidad de la mochila no tiene una relación directa con la dificultad, ni tampoco el número de elementos promedio que pertenecen a la solución. Finalmente se pudo concluir que no hay una correlación sustancial entre el grupo al que pertenece la instancia o el número de elementos de esta y la dificultad de la misma para este problema.

5.2 Resultados y discusión

En esta sección se presentan los resultados experimentales y la comparación con las metaheuristicas presentadas en el estado del arte (IHEA [15], GRASP+tabu [14]). El algoritmo desarrollado fue implementado en MATLAB y ejecutado en un portátil Lenovo G-400 con un Intel(R) Core (TM) i5–3230M CPU @ 2.60GHz, con 8

GB RAM y Windows 10, en instancias de alta dimensionalidad de 1000 y 2000 elementos.

5.3 Criterios de evaluación de las metaheuristicas

Para este trabajo se tuvieron en cuenta dos criterios de evaluación que fueron la tasa de éxito y el tiempo de ejecución.

5.4 Instancias de prueba

Para este trabajo se seleccionaron 80 instancias proveídas por el autor del artículo [17]. Estas 80 instancias están divididas en dos grupos de acuerdo al número de elementos 1000 y 2000. Estas a su vez están divididas en 4 grupos de acuerdo a la cantidad de valores no cero en la matriz de beneficios (25%, 50%, 75% y 100%)

5.5 Ajuste de tiempos de ejecución

Se realizó un ajuste a los tiempos reportados por GRASPr+tabu, tal como se hizo en la propuesta IHEA, con el fin de realizar una equivalencia respecto a los tiempos de ejecución dado a que los equipos en los cuales se ejecutaron los algoritmos tienen características totalmente distintas, por lo tanto esto dificulta en gran medida la comparación de los resultados, añadiendo también que la implementación de todos los algoritmos trabajados en este proyecto fue hecha en MATLAB, y la de la mayoría de trabajos de la literatura se realizó en C o C++, incluida GRASP+tabu y IHEA que están en C++.

Por esto se lleva a cabo una conversión de los tiempos de GRASPr+tabu e IHEA encontrando un factor que permite hacer una aproximación superficial para que los tiempos de ejecución estén determinados a una misma velocidad de procesador, y se calcula hallando una relación entre la velocidad de dos procesadores como sigue:

$$factor = frecProcA / FrecProcB$$

Ecuación 5.1 *Factor Frecuencia Procesador [15]*

Donde frecProcA es la frecuencia del primer procesador que en este caso es el que se usó en [14] y en ,y FrecProcB es la frecuencia del segundo procesador que sería el usado para la experimentación en este trabajo y el factor se aplicaría sobre el procesador A. El cálculo del factor de ajuste para el procesador de GRASP+tabu quedó así:

$$factorAjuste = 1.73 / 2.6$$
$$factorAjuste \approx 0.66$$

El cálculo del factor para el procesador de IHEA fue de:

$$factor = 2.8 / 2.6$$
$$factor \approx 1.08$$

Este valor indica que los tiempos de GRASP+tabu se multiplicaran por el factor de ajuste que es 0.66 y en IHEA por 1.08, modificando sus tiempos de ejecución para hacer una comparación más objetiva y equitativa. Cabe aclarar que el desempeño de un procesador no se limita a su frecuencia de reloj base, sino también a factores como el número de núcleos o la memoria cache en sus distintos niveles, entre muchos más, pero hacer una equivalencia más exacta implica una investigación que sobrepasa el ámbito del presente proyecto, de las cuales no se encontró algún avance en la literatura, y es de gran importancia si vamos a realizar una comparación donde se ve involucrado el tiempo de ejecución, razón por la cual se tiende en algunos estudios a reportar los resultados principalmente en evaluaciones de función y tasa de éxito, que son variables independientes del equipo donde se realice la experimentación.

De igual forma respecto al lenguaje de programación no fue posible realizar un ajuste de la misma forma que se hizo con el procesador, el cual hubiera sido conveniente ya que MATLAB por ser un lenguaje interpretado puede tener en alguna medida un desempeño menor que C++, sin embargo se omite este factor debido a que no hay una magnitud con la que se pueda realizar algún tipo de conversión o ajuste, por lo cual se deja a consideración en el momento del análisis y la comparación tener en cuenta este criterio.

5.6 Resultados del algoritmo propuesto

La evaluación del algoritmo se llevó a cabo de la siguiente manera: se ejecutaron cada una de las instancias 100 veces, y en cada una de las ejecuciones la condición de parada es determinada si se alcanza el número de evaluaciones de función total o si se alcanza el máximo de cada instancia.

En la Tabla 5.1 y en la Tabla 5.2 se muestran los resultados del algoritmo MOS propuesto en contraste con los resultados del algoritmo GRASP+tabu, el cual era el estado del arte en el momento que se inició este trabajo, y también con el algoritmo IHEA que es el estado del arte actual.

instancia	GRASPr+tabu				IHEA				MOS			
	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s
1000_25_1	6172407	100	0	17,646	6172407	100	0,00	2,986	6172407	100	0	0,398
1000_25_2	229941	66	8.3E-3	19,788	229941	100	0,00	5,821	229941	100	0	1,863
1000_25_3	172418	100	0	12,996	172418	100	0,00	6,363	172418	100	0	0,685

instancia	GRASPr+tabu				IHEA				MOS			
	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s
1000_25_4	367426	100	0	15,666	367426	100	0,00	7,876	367426	100	0	0,91
1000_25_5	4885611	100	0	22,614	4885611	100	0,00	5,986	4885611	100	0	21,248
1000_25_6	15689	100	0	4,908	15689	100	0,00	1,766	15689	100	0	0,185
1000_25_7	4945810	100	0	21,906	4945810	100	0,00	5,188	4945810	90	2,224E-6	37,879
1000_25_8	1710198	100	0	42,726	1710198	100	0,00	7,672	1710198	100	0	10,265
1000_25_9	496315	100	0	18,018	496315	100	0,00	7,442	496315	100	0	1,145
1000_25_10	1173792	100	0	35,358	1173792	100	0,00	8,179	1173792	100	0	5,77
1000_50_1	5663590	100	0	30,444	5663590	100	0,00	7,42	5663590	100	0	4,144
1000_50_2	180831	100	0	0,864	180831	100	0,00	3,987	180831	100	0	0,554
1000_50_3	11384283	100	0	19,116	11384283	100	0,00	3,605	11384283	100	0	2,817
1000_50_4	322226	100	0	13,236	322226	100	0,00	5,868	322226	100	0	0,363
1000_50_5	9984247	86	1.5E-4	24,498	9984247	98	6,0E−6	3,955	9984247	100	0	10,67
1000_50_6	4106261	100	0	34,848	4106261	100	0,00	8,306	4106261	100	0	3,372
1000_50_7	10498370	84	1.3E-4	20,058	10498370	100	0,00	3,871	10498370	100	0	2,765
1000_50_8	4981146	20	1.2E-2	69,774	4981146	99	1,0E−6	9,887	4981146	90	1,53E-6	18,79
1000_50_9	1727861	100	0	31,662	1727861	100	0,00	10,131	1727861	100	0	3,975
1000_50_10	2340724	94	2.3E-4	57,168	2340724	100	0,00	8,009	2340724	100	0	3,695
1000_75_1	11570056	65	7.6E-5	38,4	11570056	100	0,00	5,283	11570056	100	0	14,539
1000_75_2	1901389	100	0	19,482	1901389	100	0,00	7,011	1901389	100	0	1,319
1000_75_3	2096485	100	0	23,916	2096485	100	0,00	9,441	2096485	100	0	0,145
1000_75_4	7305321	100	0	33,054	7305321	100	0,00	7,394	7305321	100	0	2,685
1000_75_5	13970240	93	4.0E-4	22,434	13970842	100	0,00	6,504	13970240	100	0	8,544
1000_75_6	12288738	100	0	20,064	12288738	100	0,00	4,82	12288738	100	0	3,661
1000_75_7	1095837	100	0	13,896	1095837	100	0,00	7,689	1095837	100	0	0,089
1000_75_8	5575813	100	0	41,082	5575813	100	0,00	8,46	5575813	100	0	2,317
1000_75_9	695774	100	0	13,608	695774	100	0,00	4,994	695774	100	0	1,409
1000_75_10	2507677	100	0	28,392	2507677	100	0,00	7,412	2507677	100	0	0,506
1000_100_1	6243494	100	0	43,206	6243494	100	0,00	7,579	6243494	100	0	2,45
1000_100_2	4854086	61	1.3E-3	50,904	4854086	100	0,00	7,659	4854086	98	9,06E-6	5,659
1000_100_3	3172022	100	0	28,236	3172022	100	0,00	6,902	3172022	100	0	3,462
1000_100_4	754727	100	0	14,178	754727	100	0,00	5,624	754727	100	0	0,931
1000_100_5	18646620	99	6.9E-7	23,49	18646620	100	0,00	4,396	18646620	97	6,97E-7	17,471
1000_100_6	16018298	96	4.2E-6	24,948	16020232	100	0,00	5,62	**16020232**	100	0	0,933
1000_100_7	12936205	100	0	26,7	12936205	100	0,00	5,976	12936205	100	0	0,887
1000_100_8	6927738	100	1.1E-4	57,63	6927738	100	0,00	7,882	6927738	100	0	4,392
1000_100_9	3874959	100	0	31,368	3874959	100	0,00	7,652	3874959	100	0	0,359
1000_100_10	1334494	100	0	14,178	1334494	100	0,00	6,772	1334494	100	0	0,575
Promedio		94,1		27,062		**99,925**		6,485		99,375		**5,096**

Tabla 5.1 *Resultados Instancias de 1000 elementos GRASP+tabu, IHEA y MOS*

Como se puede apreciar en la Tabla 5.1 se muestra el máximo valor reportado para la instancia, la tasa de éxito, la densidad relativa porcentual (DRP), y el tiempo promedio para cada instancia (CPU (s)). Resaltado en negrilla se puede observar los máximos u óptimos globales los cuales mejoraron respecto al algoritmo con el que se está comparando. Se consigue alcanzar 1 máximo que mejora el que se encuentra reportado en GRASP+TABU, en la instancia '1000_100_6'.

Comparando MOS con respecto a GRASP+tabu se puede observar que se mejoran los tiempos de ejecución del 97.5% (39/40) de las instancias de 1000 elementos y GRASP+Tabu solo tiene mejor tiempo en la instancia 1000_25_7, teniendo en cuenta el respectivo ajuste de tiempo a los resultados reportados en [14] con el método descrito anteriormente basado en la frecuencia del reloj, y sin tener en cuenta el lenguaje de programación usado para la implementación. Para las instancias de 1000 elementos el tiempo total de GRASP+Tabu es de 1082,46 segundos y el de MOS de 203.82 segundos, es decir se reduce aproximadamente 5.31 veces el tiempo global con la implementación de este trabajo, lo cual es un avance significativo en este grupo de instancias dada su complejidad.

Respecto a MOS e IHEA también se logran mejorar en un 80% (32/40) los tiempos de ejecución, siendo superado en un 20%(8/40) por IHEA, mejorando desde esta perspectiva MOS al estado del arte en este grupo de instancias. Globalmente IHEA reporta un tiempo total de 259.39 segundos contra los 203.82 de MOS, reduciendo 1.27 veces el tiempo de este primero, superando ligeramente al estado del arte heurístico desde esta otra perspectiva.

Para ver de manera más detallada el progreso obtenido en cada instancia, en la Figura 5.2 y en la Figura 5.3 se muestra la comparación de tiempo de forma individual de las tres técnicas.

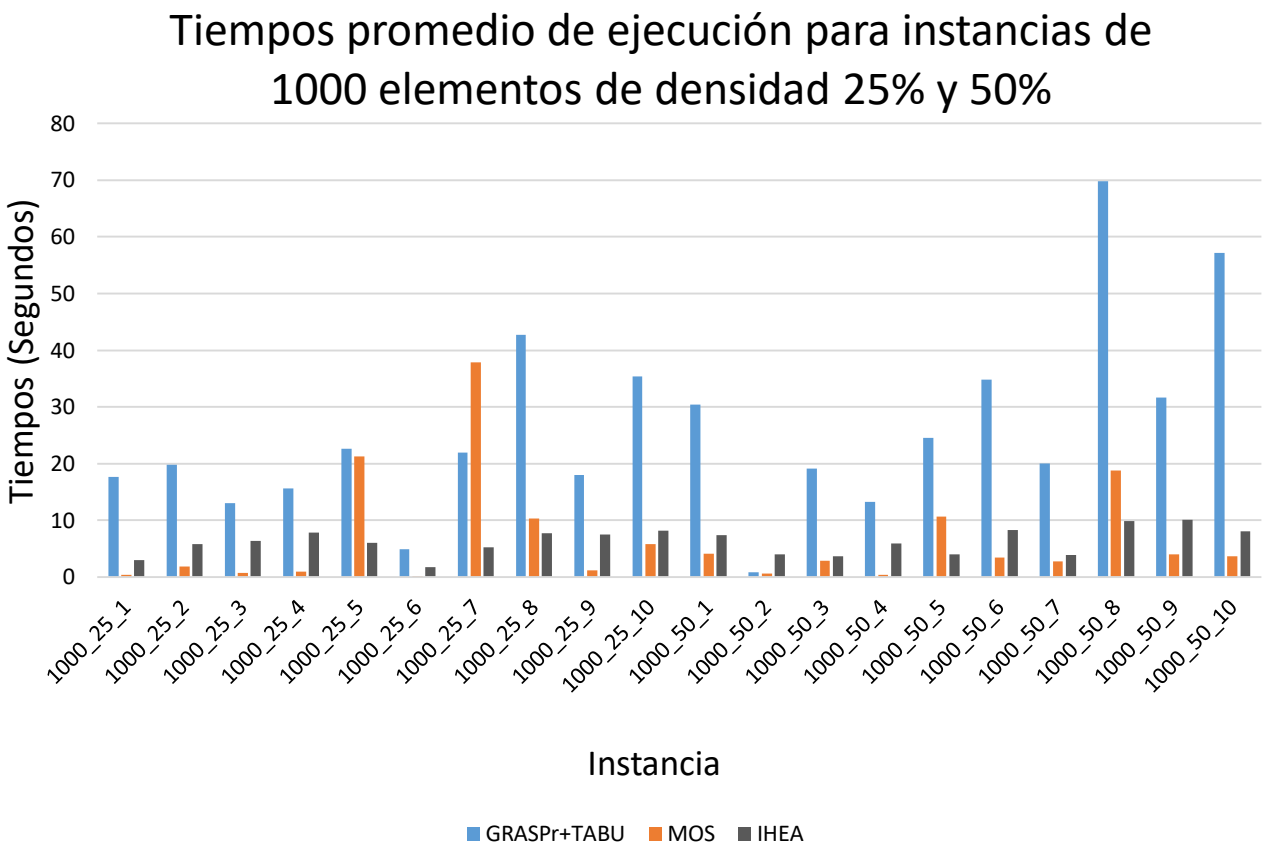

Figura 5.2 *Tiempos promedio de ejecución para Instancias de 1000 elementos de densidad 25% y 50%*

En la Figura 5.2 se muestra en un gráfico de barras el progreso obtenido para las instancias de densidad de 25% y 50% de forma individual en función del promedio de tiempo obtenido en las 100 ejecuciones, mostrando en el eje Y el tiempo en segundos y en el eje X las respectivas instancias, ilustrando las diferencias entre las tres técnicas dejando a GRASP+Tabu con los tiempos más elevados relativamente a los tiempos de MOS e IHEA por alta diferencia, y mostrando entre MOS e IHEA tiempos muy parejos a excepción de 2 instancias.

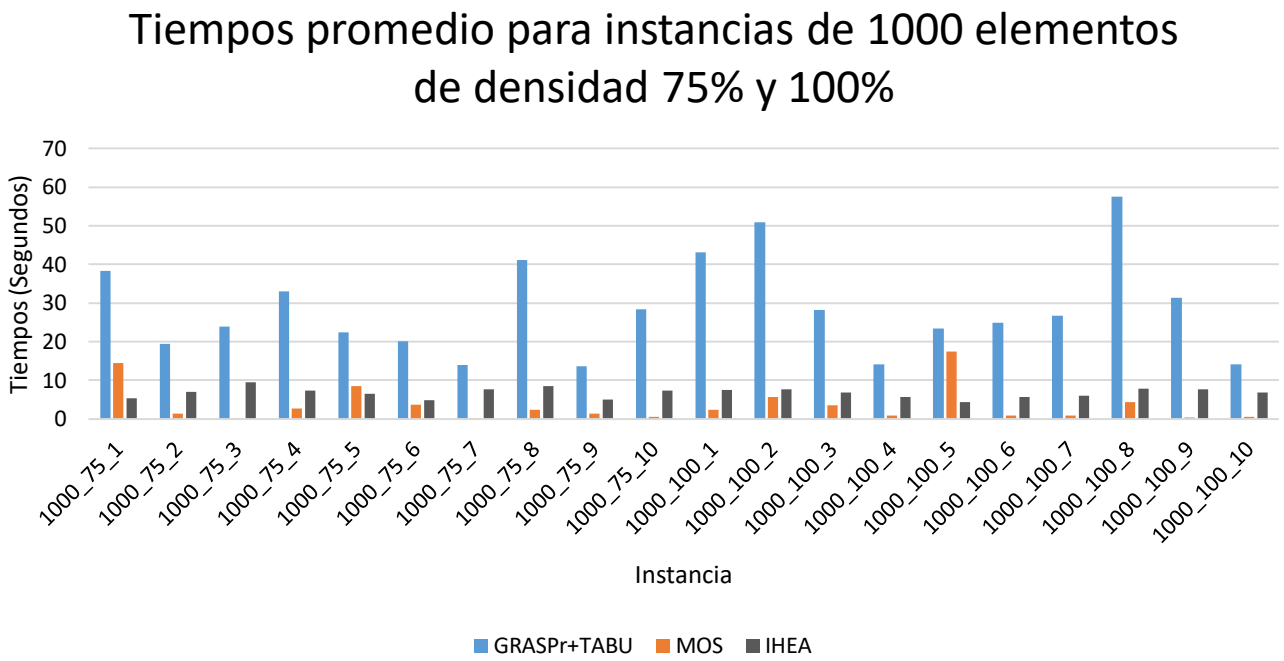

Figura 5.3 *Tiempos promedio para instancias de 1000 elementos de densidad 75% y 100%*

También se hace la misma representación para las instancias con densidades de 75% y 100% de 1000 elementos la cual puede verse en la Figura 5.3, donde se vuelve a repetir en gran medida el comportamiento de las técnicas en las instancias de 25% y 50% elementos. Para apreciar la cantidad porcentual del tiempo de cada instancia respecto a un total igual a la suma de los tres tiempos, en la Figura 5.4 se presenta un gráfico para analizar la proporción con que cada técnica contribuye al total de tiempo.

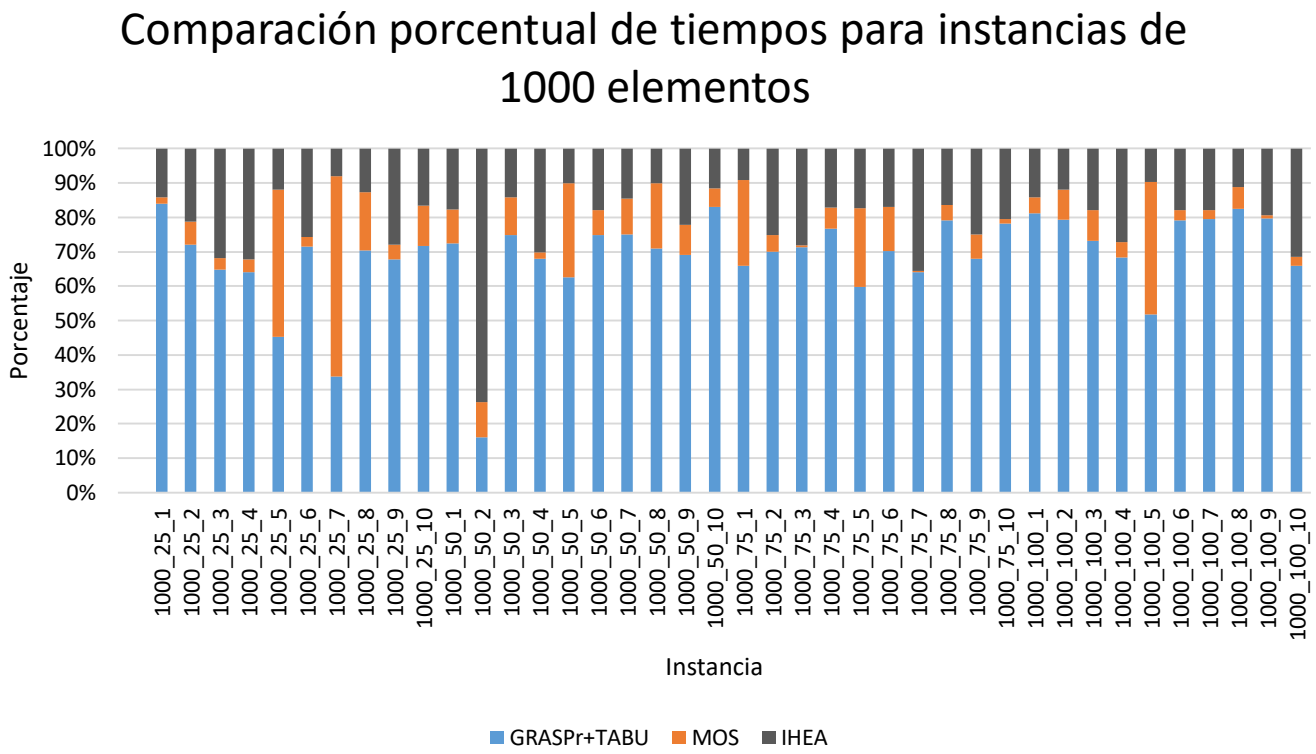

Figura 5.4 *Comparación porcentual de tiempos para instancias de 1000 elementos*

En cada columna de la Figura 5.4 se pueden ver apilados en magnitud de proporción la contribución de cada técnica al total de tiempo resultante de sumar las tres instancias, donde GRASP+Tabu obtiene un 68.9% de porcentaje promedio para todas las instancias de 1000 elementos, seguido de IHEA con un 20.3% y MOS con un 10.7%.

Respecto a la tasa de éxito general MOS mejora el número de aciertos, obteniendo una tasa de éxito de 99.37% (3975) en las ejecuciones de las 40 instancias de 1000 elementos y en GRASP+Tabu reporta una tasa de éxito de 94% (3764), es decir una diferencia de 221 (3975-3764) aciertos a óptimos locales fijados como objetivo. MOS y GRASP+Tabu tienen tasa de éxitos iguales en el 72.5% (29/40) de las instancias, MOS gana en el 22.5% (9/40) y GRASP+Tabu en el 5% (2/40), en la Figura 5.5, se puede visualizar la proporción de estos datos.

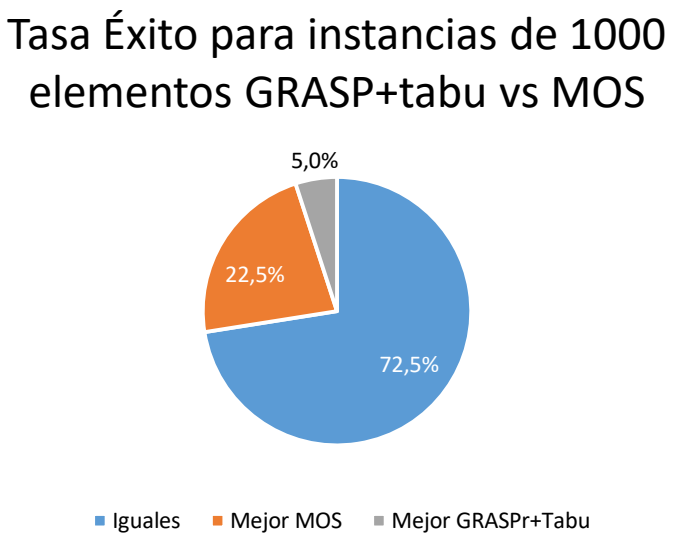

Figura 5.5 *Tasa Éxito para instancias de 1000 elementos GRASP+tabu vs MOS*

Cabe aclarar que tanto para algunas instancias en las que gana MOS y en las que gana GRASP+Tabu en tasa de éxito, es con diferencias mínimas, por lo tanto, debido al comportamiento no determinístico de estas metaheuristicas debe tenerse a consideración que la tasa de éxito de estas instancias puede variar para otro conjunto de pruebas y cambiar los resultados, pero igualmente para el análisis en este trabajo se tuvieron en cuenta sin importar su diferencia.

La tasa de éxito general de IHEA es de 99.92% (3997), superando a MOS que tiene un 99.37%(3975) en número de aciertos para las instancias de 1000 elementos, teniendo tasas de éxito iguales en el 87,5% (35/40), IHEA gana en un 10% (4/40) y MOS en un 2.5% (1/40). En la Figura 4.8 se muestran las proporciones mencionadas anteriormente donde se puede apreciar la superioridad de IHEA en tasa de éxito en este grupo de instancias.

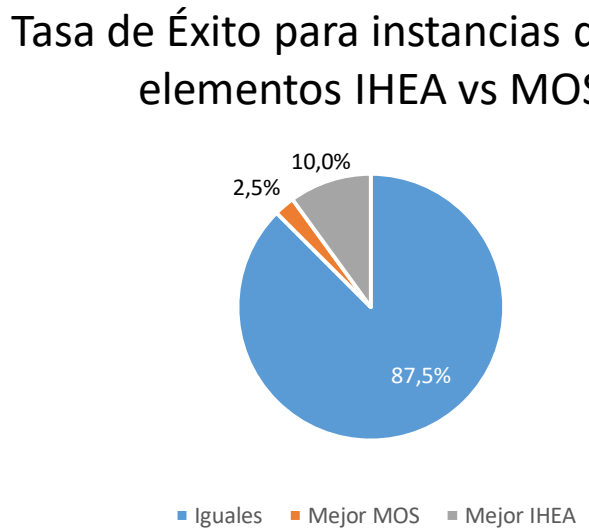

Figura 5.6 *Tasa de Éxito para instancias de 2000 elementos IHEA vs MOS*

Para el segundo grupo de instancias de 2000 elementos ver Figura 5.6 también se tuvo la misma configuración de parámetros que el de 1000 elementos. Los resultados de las 100 ejecuciones para instancias de 2000 elementos se muestran en la Tabla 5.2**Figura 5.6**.

Instancia	GRASP+tabu				IHEA				MOS			
	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s)
2000_25_1	5268188	100	0	309,942	5268188	100	0,00	24,045	5268188	91	5,6E-5	36,946
2000_25_2	13294030	100	0	198,438	13294030	100	0,00	26,91	13294030	100	0	28,731
2000_25_3	5500433	56	4.45E-4	480,078	5500433	100	0,00	31,248	5500433	100	0	31,061
2000_25_4	14625118	100	0	208,134	14625118	100	0,00	18,414	14625118	100	0	28,932
2000_25_5	5975751	100	0	442,998	5975751	100	0,00	30,35	5975751	94	2,2E-4	48,91
2000_25_6	4491691	100	0	284,76	4491691	100	0,00	25,317	4491691	100	0	14,62
2000_25_7	6388756	100	0	334,926	6388756	100	0,00	27,192	6388756	100	0	13,023
2000_25_8	11769873	100	0	268,17	11769873	100	0,00	24,391	11769873	100	0	23,541
2000_25_9	10960328	100	0	269,886	10960328	100	0,00	24,214	10960328	100	0	28,043
2000_25_10	139236	100	0	65,874	139236	100	0,00	8,155	139236	100	0	3,98
2000_50_1	7070736	39	2.7E-2	284,592	7070736	100	0,00	30,257	7070736	100	0	3,439
2000_50_2	12587545	100	0	320,922	12587545	100	0,00	25,858	12587545	100	0	10,43
2000_50_3	27268336	100	0	185,328	27268336	100	0,00	24,506	27268336	100	0	14,947
2000_50_4	17754434	100	0	469,596	17754434	100	0,00	26,466	17754434	94	3,9E-7	24,378
2000_50_5	16805490	90	9.3E-4	894,132	16806059	100	0,00	34,622	16805490	96	2,5E-4	18,35
2000_50_6	23076155	50	5.1E-4	276,054	23076155	100	0,00	23,305	23076155	100	0	16,348
2000_50_7	28759759	6	8.1E-3	428,508	28759759	100	0,00	27,394	28759759	84	5,2E-5	38,265
2000_50_8	1580242	100	0	99,108	1580242	100	0,00	15,052	1580242	100	0	3,178

Instancia	GRASP+tabu				IHEA				MOS			
	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s)	Máximos	T.E	DRP	CPU(s)
2000_50_9	26523791	100	0	205,272	26523791	100	0,00	21,271	26523791	100	0	14,366
2000_50_10	24747047	100	0	245,034	24747047	100	0,00	22,262	24747047	100	0	11,342
2000_75_1	25121998	100	0	484,23	25121998	100	0,00	24,539	25121998	100	0	25,677
2000_75_2	12664670	89	4.7E-4	306,03	12664670	100	0,00	23,311	12664670	100	0	6,037
2000_75_3	43943994	100	0	165,834	43943994	100	0,00	20,221	43943994	100	0	8,128
2000_75_4	37496613	100	0	212,478	37496613	100	0,00	21,493	37496613	100	0	9,161
2000_75_5	24834948	73	2.1E-4	410,598	24835349	100	0,00	29,634	**24835349**	100	0	7,932
2000_75_6	45137758	100	0	183,882	45137758	100	0,00	22,531	45137758	100	0	4,471
2000_75_7	25502608	100	0	294,084	25502608	100	0,00	23,596	25502608	100	0	6,259
2000_75_8	10067892	100	0	206,898	10067892	100	0,00	23,285	10067892	100	0	6,481
2000_75_9	14171994	97	1.6E-5	319,236	14177079	100	0,00	34,569	**14174691**	100	0	0,823
2000_75_10	7815755	78	8.7E-5	195,132	7815755	100	0,00	22,18	7815755	93	4,6E-6	38,12
2000_100_1	37929909	100	0	261,426	37929909	100	0,00	23,352	37929909	100	0	8,107
2000_100_2	33647322	95	8.2E-5	474,906	33665281	100	0,00	37,068	**33648003**	100	0	15,558
2000_100_3	29952019	34	0.003	893,574	29952019	100	0,00	25,109	29952019	100	0	17,98
2000_100_4	26949268	100	0	426,474	26949268	100	0,00	25,704	26949268	100	0	20,927
2000_100_5	22041715	70	3.0E-4	451,212	22041715	95	1,1 E−5	25,214	22041715	97	1,2E-5	58,623
2000_100_6	18868887	100	0	328,914	18868887	100	0,00	24,1	18868887	100	0	13,153
2000_100_7	15850597	100	0	346,908	15850597	100	0,00	24,359	15850597	100	0	26,677
2000_100_8	13628967	100	0	224,442	13628967	100	0,00	24,03	13628967	100	0	5,943
2000_100_9	8394562	97	2.2E-4	182,586	8394562	100	0,00	20,181	8394562	100	0	12,207
2000_100_10	4923559	92	1.4E-4	120,03	4923559	100	0,00	16,244	4923559	100	0	24,438
Promedio		89,15		319,016		**99,87**		24,549		98,72		**18,238**

Tabla 5.2 *Resultados Instancias de 2000 elementos GRASP+Tabu, IHEA y MOS*

Se reportan los mismos campos de las instancias de 1000 elementos, resaltando también aquellos máximos que se mejoran. Para este grupo de instancias se encontraron 3 nuevos máximos que mejoran los que se encuentran reportados en GRASP+TABU, que son los de las instancias ‘2000_75_5’, ‘2000_75_9’ y ‘2000_100_2’.

En las pruebas de las instancias de 2000 elementos se mejora el tiempo promedio de cada instancia en un 100% respecto a GRASP+Tabu, es decir todas las instancias tienen un menor tiempo de ejecución, donde también se hizo el respectivo ajuste con el factor 0.66 a cada tiempo reportado reduciendo su valor en este porcentaje. El tiempo total de GRASP+Tabu es 12760.626 segundos contra 729.53 segundos de MOS, reduciendo en una magnitud de 17.5 veces el tiempo global.

MOS mejora en un 72.5% (29/40) a IHEA en el número de instancias que tiene un tiempo menor, dejando a IHEA con un 27.5% (11/40) de instancias con mejor

tiempo. Se logra mejorar también el tiempo global de MOS respecto a IHEA en un factor de 1.34, obteniendo un tiempo global para este grupo de instancias de 729.53 segundos contra los 981.95 segundos de IHEA. La comparación de los tiempos de ejecución promedio por instancia de MOS, IHEA y GRASP+Tabu se muestran en la Figura 5.7.

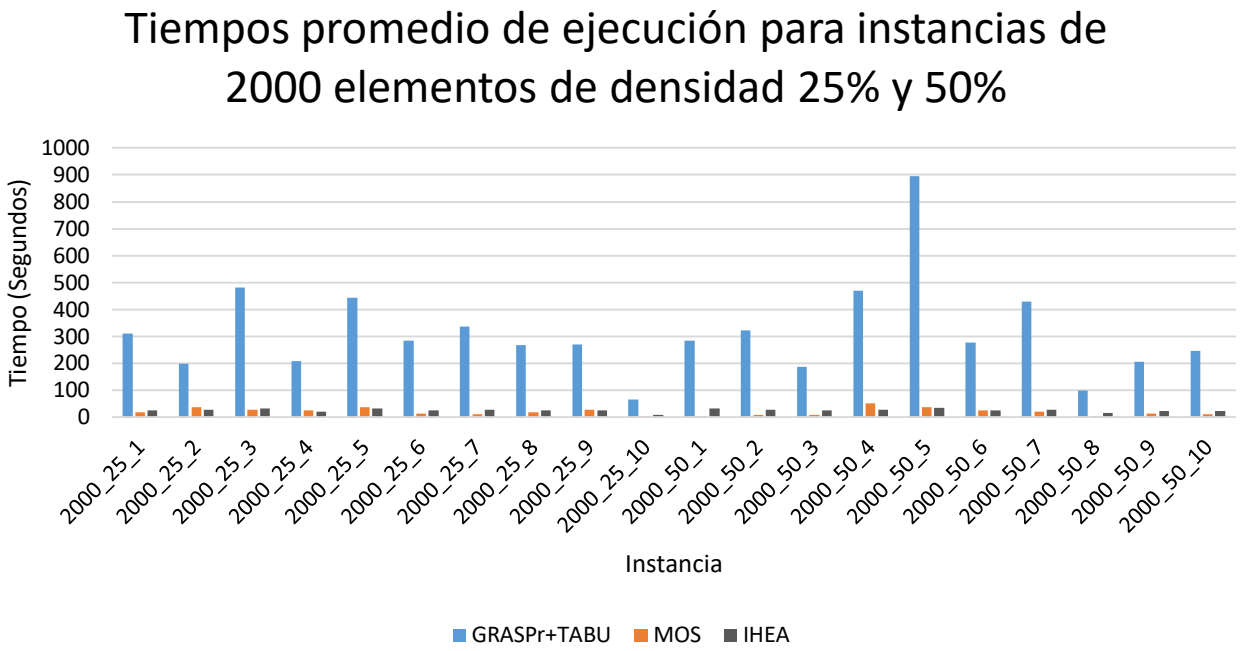

Figura 5.7 *Tiempos promedio de ejecución para instancias de 2000 elementos de densidad 25% y 50%*

La Figura 5.7 muestra la diferencia proporcional de tiempos de ejecución en el cual se puede apreciar la variación de tiempo de GRASP+TABU respecto a MOS e IHEA, en donde estos dos últimos se destacan siendo inferiores en gran dimensión logrando dejar la mayoría de estos valores por debajo de los 50 segundos, lo cual es una mejora muy importante debido al tamaño de estas instancias y lo complejo que sería utilizar un algoritmo exacto para encontrar óptimos globales en tiempos cercanos a los reportados.

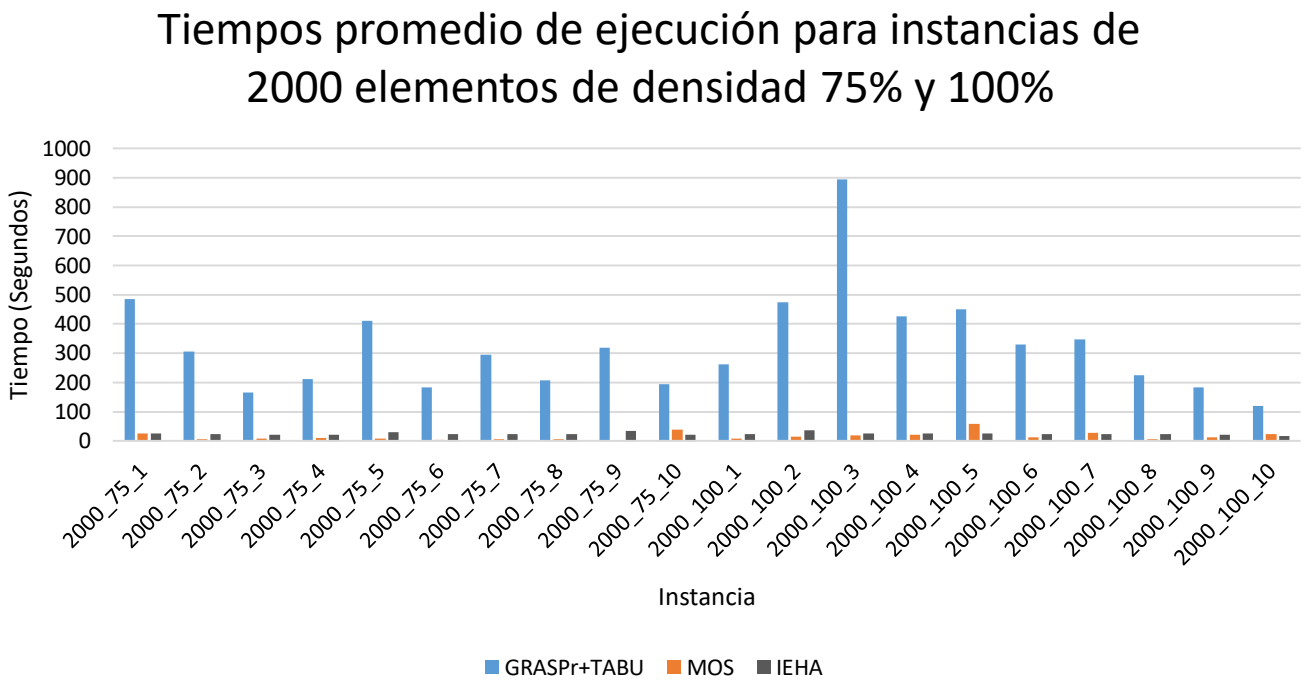

Figura 5.8 *Tiempos promedio de ejecución para instancias de 2000 elementos de densidad 75% y 100%*

En la Figura 5.8 muestra el resto de instancias de este grupo de prueba de 2000 elementos con densidades de 75% y 100%, donde se puede apreciar que también se consigue reducir los tiempos de ejecución promedio por debajo de los 50 segundos con diferencia considerable a GRASP+Tabu en la mayoría de los casos y manteniendo tiempos cercanos con IHEA. En la Figura 5.9 se muestra una gráfica de barras apiladas para observar la proporción de tiempo que cada técnica contribuye al total resultante de la suma de los tres tiempos de cada técnica, obteniendo como resultado para GRASP+Tabu un 86.91% de contribución promedio para el grupo de prueba de 2000 elementos, seguido de IHEA con un 7.7% y un poco por debajo MOS con un 5.4%.

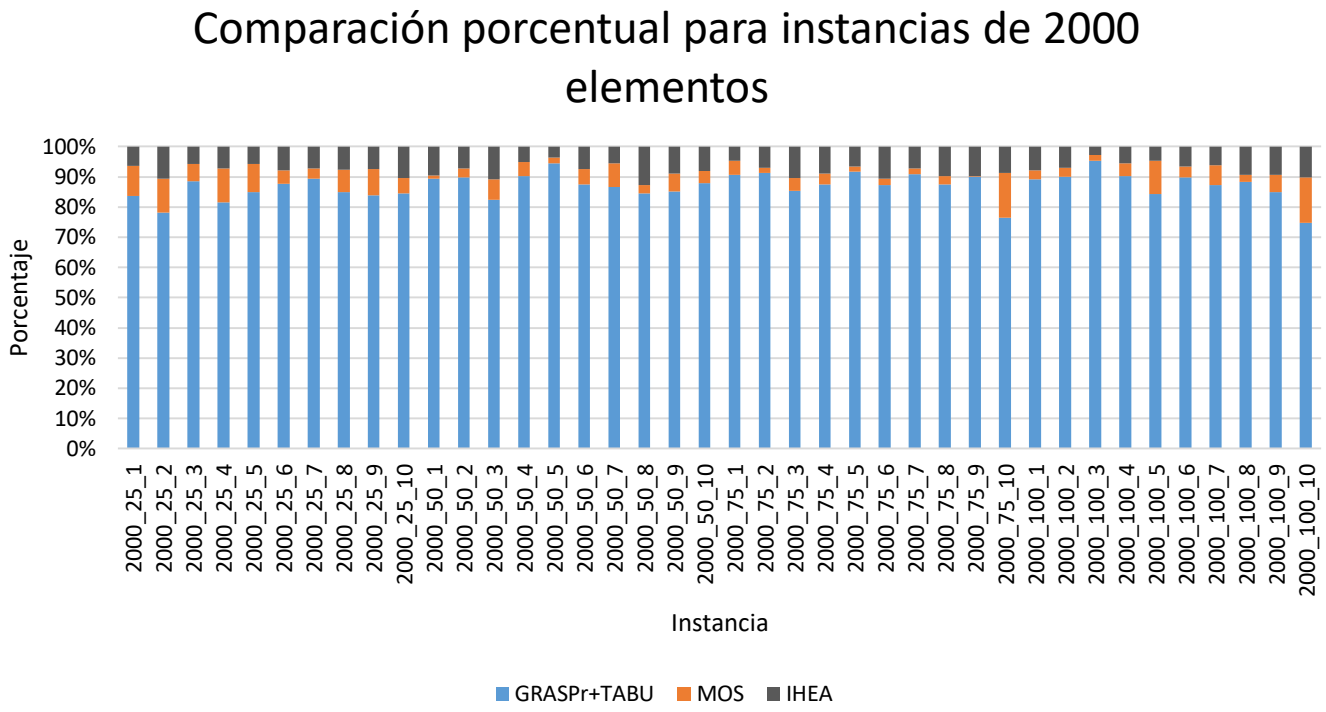

Figura 5.9 *Comparación porcentual para instancias de 2000 elementos*

Realizando una suma total del tiempo promedio de todas las instancias de 1000 y 2000 elementos se puede ver también la gran diferencia que se obtiene en tiempo de ejecución, como se muestra en la Figura 5.10. Se reduce aproximadamente el 93.25% del tiempo de ejecución total desde 13843,1 segundos hasta 942,65 segundos respecto a GRASP+Tabu y un 24,81% en relación con IHEA, estableciendo desde esta otra perspectiva superioridad en el tiempo de ejecución.

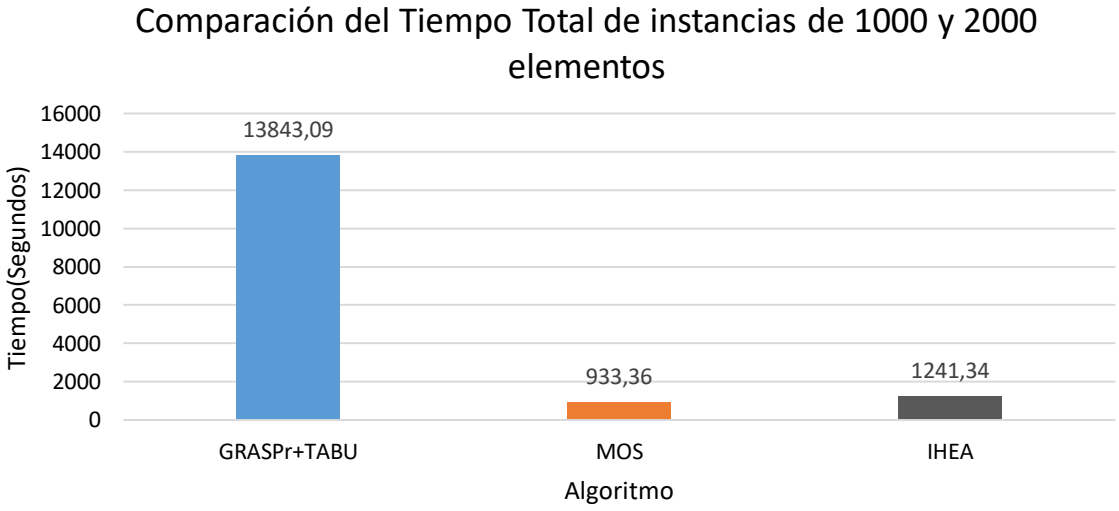

Figura 5.10 *Comparación del Tiempo Total de instancias de 1000 y 2000*

Analizando la tasa de éxito en las instancias de 2000 elementos se logra obtener una tasa de éxito de 98.7% (3949) en las ejecuciones de las 40 instancias y en GRASP+Tabu reporta una tasa de éxito de 89.15%(3566), con una diferencia de 383 ejecuciones (3975-3764) donde se logra alcanzar el máximo reportado. MOS y GRASP+Tabu tienen tasa de éxitos iguales en el 57.5%(23/40) de las instancias, MOS gana en el 35% (14/40) y GRASP+Tabu en el 7.5% (3/40), ver Figura 5.11.

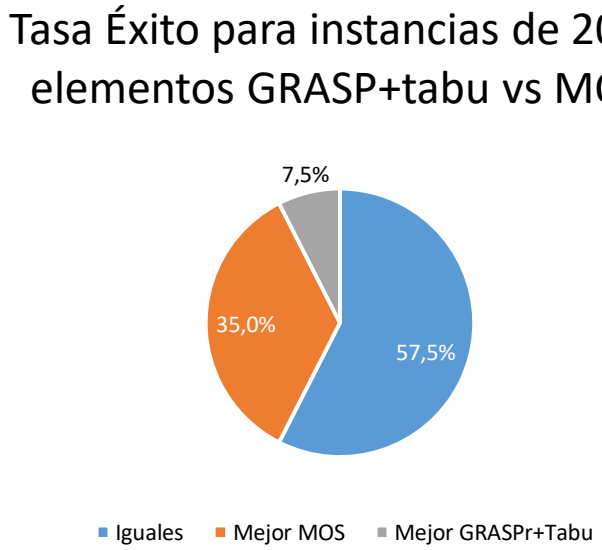

Figura 5.11 *Tasa Éxito para instancias de 2000 elementos GRASP+tabu vs MOS*

En la Figura 5.12 IHEA alcanza una efectividad del 99.92% (3995), por encima de MOS que tiene un 98.72%(3949) en número de aciertos para las instancias de 2000 elementos superando la efectividad de MOS en 46 ejecuciones, y visto desde otro ángulo se observa que la tasa de éxito es igual en el 82,5% (33/40) de los casos, IHEA gana en un 15% (6/40) y MOS en un 2.5% (1/40).

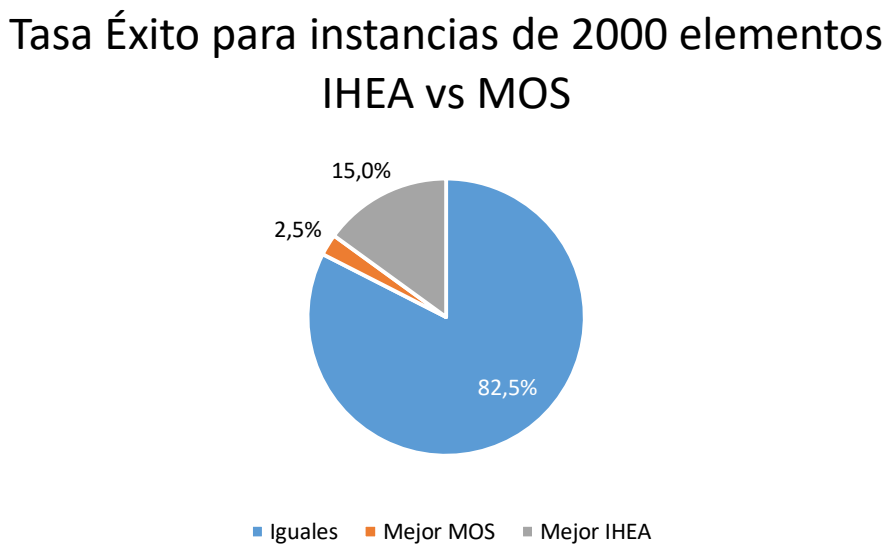

Figura 5.12 *Tasa Éxito para instancias de 2000 elementos IHEA vs MOS*

MOS logra obtener tiempos de buena calidad en la mayoría de las instancias debido a que la mayor parte de aciertos que se registraron para instancias de 1000 y 2000 elementos se hacen en la primera iteración alcanzando un 71.1% de los máximos reportados, y siguiendo en el orden de porcentaje está la segunda iteración con un 16.5%, y continuando una secuencia decreciente del porcentaje a medida que avanzan las iteraciones ver Figura 5.13.

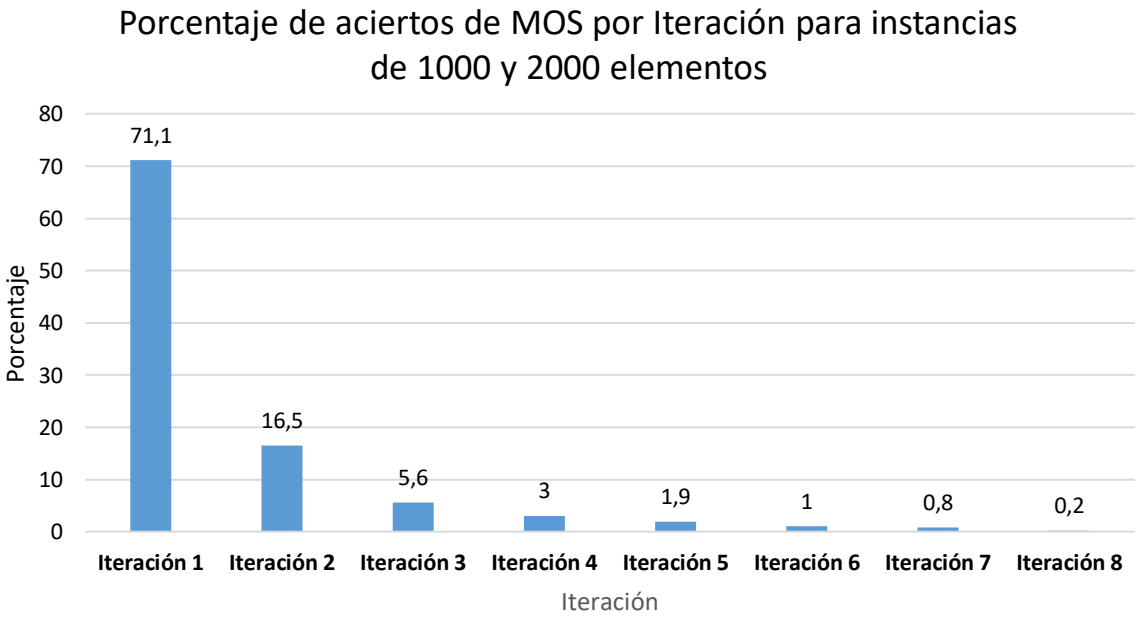

Figura 5.13 *Porcentaje de aciertos de MOS por Iteración para instancias de 1000 y 2000 elementos*

En la Figura 5.13, se puede observar los porcentajes de aciertos por iteración que se alcanzan con la metaheuristica MOS, donde se puede concluir la efectividad de las técnicas subordinadas y la eficiente coordinación de participación y administración de datos de MOS para alcanzar en las siguientes iteraciones óptimos locales de mayor complejidad.

La técnica SGVNS consigue el 56.7% de todos los aciertos alcanzados, y GRASPr el 43.3 % restante, de lo que se puede interpretar además de la efectividad que pueda tener una técnica subordinada sobre la otra, es que cada instancia tiene óptimos locales en distintos planos de soluciones que una técnica puede alcanzar más fácilmente que la otra debido a la configuración de las diferentes fases de cada algoritmo, debido a esto se omite una posible comparación entre las técnicas ya que algunas alcanzaran soluciones que seguramente la otra no alcanzaría de forma eficiente, y por ello este trabajo se enfoca en construir un algoritmo compacto que consiga abarcar diferentes espacios de soluciones delegando a cada técnica su dimensión de búsqueda adecuada.

6 CONCLUSIONES Y TRABAJO FUTURO

6.1 Conclusiones

En este proyecto de investigación se diseñó, implementó y evaluó un algoritmo hibrido que combina las adaptaciones de los algoritmos SGVNS [20] y GRASPr [14] bajo un gestor de técnicas denominado MOS para resolver el problema 0-1 QKP. La primera técnica, SGVNS, cuenta con un algoritmo de búsqueda local VND con dos vecindarios, un componente de sacudida para escapar de estancamientos en el espacio de soluciones y se incorpora una fase de construcción tipo GRASP para añadir una funcionalidad de inicio múltiple, que es precedida por otra fase de construcción por peso para la exploración de diferentes espacios de soluciones. La segunda técnica denominada GRASPr cuenta también con una fase de pre-construcción por peso, construcción por densidad con una lista reducida de candidatos (RCL), la búsqueda local de selección de estructura de vecindad aleatoria, moviendo la fase de actualización de la solución con que contaba el algoritmo original a MOS, el cual actualiza la solución al final de cada iteración y sirve como coordinador de la participación de las metaheuristicas como técnicas subordinadas a las cuales se les provee una participación dinámica de acuerdo a su desempeño en cada iteración, competencia que inicia equitativamente.

Los resultados arrojados indican que la actual propuesta obtiene mejores resultados en comparación con GRASP+tabu respecto a tasa de éxito ya que MOS alcanza el 99.05% de los máximos contra un 91.62% para todos los grupos de instancias, y también se mejora el tiempo de ejecución reduciendo en un 93% aproximadamente el valor total de las instancias de 1000 y 2000 elementos reportado por GRASP+Tabu.

También se obtienen resultados competitivos respecto al tiempo de ejecución en comparación al estado del arte IHEA (presentado en 2017, luego de haber iniciado el presente trabajo de grado), mejorando desde la perspectiva de tiempo total reduciendo en un 24.8% el obtenido por IHEA y también el número de instancias en las que se obtiene un mejor tiempo con una cantidad de 61 de 80. IHEA mejora ligeramente la tasa de éxito de MOS en un total de 68 ejecuciones, es decir el 0.85% de la prueba global ya que MOS reporta una tasa de éxito de 98.72% e IHEA un 99.87%.

La actual propuesta reporta 4 nuevos máximos en comparación a los máximos reportados por GRASP+tabu, un máximo en el grupo de 1000 elementos (instancia 1000_100_6) y tres máximos en el grupo de 2000 elementos (instancias 2000_75_5, 2000_75_9, 2000_100_2).

Se realizaron adaptaciones a las técnicas subordinadas consideradas del estado del arte dado a que no fue posible replicar los algoritmos originales a un nivel óptimo debido a ambigüedades en la descripción de los algoritmos por parte de los autores y que no fue posible ponerse en contacto con ellos, y entonces para adaptar los algoritmos bajo el marco de trabajo MOS y mejorar el rendimiento general fue necesario implementar una gran variedad de modificaciones como en la fase de construcción GRASP que se incorporó a SGVNS, el intercambio de información entre las técnicas a través de MOS, la fase de construcción por peso que se agregó a las dos técnicas, las dos estructuras de vecindad que se introducen en este trabajo con las variables de control de exploración y selección de candidatos necesarias para su funcionamiento, el uso del vector de frecuencias para acotar el escalamiento a soluciones de calidad y construir las nuevas soluciones en cada iteración MOS. En cuanto a MOS se agregó una fase como la que se usó en [14], en la que se actualiza la solución de la siguiente iteración tomando la información recogida durante toda la ejecución global del algoritmo, y que será la entrada de las técnicas subordinadas en dicha iteración.

El trabajo realizado en este proyecto presenta un buen punto de partida para posicionar un algoritmo hibrido que utilice un gestor de técnicas MOS como estado del arte para resolver el 0-1 QKP, debido al sobresaliente comportamiento que tuvieron las técnicas subordinadas al trabajar colaborativamente con el gestor de técnicas MOS, asignando a cada técnica un sector del espacio de soluciones y que juntas pueden guiar la exploración a sectores más prometedores y encontrar más rápidamente soluciones de alta calidad.

De acuerdo con los resultados obtenidos y recordando la pregunta de investigación que motivo el proyecto: ¿Cómo se puede mejorar el desempeño (mayor tasa de éxito y menor tiempo) de dos técnicas trabajando sinérgicamente bajo un enfoque MOS para resolver problemas 0-1 QKP en instancias reconocidas por la comunidad académica y científica del área (disponibles en [14])?

Se concluye que si es posible mejorar el desempeño respecto a la tasa de éxito y menor tiempo utilizando un algoritmo hibrido que combine dos técnicas subordinadas bajo el enfoque MOS con el fin de resolver el 0-1 QKP debido a que es el primer trabajo que resuelve el problema con este enfoque y logra mejorar considerablemente los resultados con relación a GRASP+tabu en tasa de éxito y tiempo de ejecución y en cuanto al algoritmo IHEA hay una mejora considerable respecto a tiempos de ejecución siendo superado por muy poco en tasa de éxito; por lo que si se invierte más esfuerzo en investigar este tipo de enfoques se pueden lograr resultados mucho mejores a los que se entregan en este trabajo.

6.2 Trabajo Futuro

Debido a la gran variedad de aplicaciones que puede tener el problema la mochila cuadrática binaria en la vida real, es importante seguir mejorando los resultados, aunque estos hayan tenido un progreso enorme en la última década respecto a los primeros trabajos basados principalmente en métodos exactos, y esta mejora sustancial se produce gracias al enfoque heurístico y metaheurístico y a los avances en la capacidad de los procesadores de los ordenadores.

Dada la alta complejidad que pueden tener los problemas del mundo real en función del número de elementos que puedan tener, es indispensable seguir trabajando en la resolución de instancias de mucho más tamaño, como se hace en [15], que da lugar a que nuevas investigaciones se hagan sobre dichas instancias, y continuar progresivamente el mejoramiento de la eficiencia sobre esas instancias y se obtengan mejores técnicas heurísticas o de cualquier enfoque que se pueda dar un nuevo salto o aumento en el número de elementos en 0-1 QKP.

Debido a que algunas instancias de este problema tienen un comportamiento especial dada su complejidad, es necesario investigar en técnicas que permitan identificar este tipo de instancias y sean capaces de resolverlas de manera eficiente, hallando cuales son los parámetros o características que estas tienen y abordar de una mejor manera la búsqueda de su solución.

En esta investigación se desarrolló un enfoque distinto para el procedimiento que realiza los intercambios de artículos en la mochila para las diferentes fases de los algoritmos trabajados, por ende se plantea el mecanismo presentado en este trabajo que seguramente se puede explotar más y mejorar con el objetivo de reducir aún más el espacio de búsqueda, por lo cual se propone investigar más en este ámbito debido a que una buena elección al momento de realizar el intercambio es la diferencia entre una buena solución y una mala.

Respecto a MOS, pueden realizarse investigaciones nuevas con un número mayor de técnicas, combinando la naturaleza de estas heurísticas y su tamaño con el fin de abarcar todas las dificultades que pueden presentar las instancias, que puedan no cubrirse con una metaheurística o dos, y haciendo esto aportar sustancialmente a la literatura en materia de enfoques ya sea para soluciones iniciales, construcción de soluciones, búsquedas locales, creación de descendencia o cruce de individuos, limitación del espacio de búsqueda, mutación o modificación de soluciones, entre otros. También en cuanto a la repartición de evaluaciones de función se necesita experimentar más respecto al criterio con que se calcula la calidad de una técnica en una iteración MOS, ya sea estudiando otras formas de hacer operaciones con el

fitness de los óptimos locales o implementar otro mecanismo para determinar qué tan buena o mala fue una técnica y asignar un valor que MOS pueda tomar para la realización de sus cálculos de asignación de participación.

Se insta a la comunidad de investigadores en el área, a seguir adaptando nuevas metaheuristicas que hayan sido usadas en otro tipo de problemas con resultados sobresalientes, para experimentar su desempeño en el problema de la mochila cuadrática binaria, tanto en algoritmos poblacionales como de estado simple y creando nuevas líneas de trabajo sobre este problema con las nuevas aplicaciones reales que vayan surgiendo con el avance de la tecnología.

7 REFERENCIAS

- [1] G. Gallo, P. L. Hammer, and B. Simeone, "Quadratic knapsack problems," in *Combinatorial Optimization*, M. W. Padberg, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 132-149.
- [2] P. Chaillou, P. Hansen, and Y. Mahieu, "Best network flow bounds for the quadratic knapsack problem," in *Combinatorial Optimization: Lectures given at the 3rd Session of the Centro Internazionale Matematico Estivo (C.I.M.E.) held at Como, Italy, August 25–September 2, 1986*, B. Simeone, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 225-235.
- [3] A. Caprara, D. Pisinger, and P. Toth, "Exact Solution of the Quadratic Knapsack Problem," *INFORMS J. on Computing,* vol. 11, pp. 125-137, 1999.
- [4] A. Billionnet, A. Faye, and É. Soutif, "A new upper bound for the 0-1 quadratic knapsack problem," *European Journal of Operational Research,* vol. 112, pp. 664-672, 2/1/ 1999.
- [5] A. Billionnet and É. Soutif, "An exact method based on Lagrangian decomposition for the 0–1 quadratic knapsack problem," *European Journal of Operational Research,* vol. 157, pp. 565-575, 2004/09/16/ 2004.
- [6] A. Billionnet and F. Calmels, "Linear programming for the 0–1 quadratic knapsack problem," *European Journal of Operational Research,* vol. 92, pp. 310-325, 1996/07/19 1996.
- [7] A. B. R. a. David Pisinger and R. Sandvik, "Solution of Large Quadratic Knapsack Problems Through Aggressive Reduction," *INFORMS Journal on Computing,* vol. 19, pp. 280-290, 2007.
- [8] D. J. Rader Jr and G. J. Woeginger, "The quadratic 0–1 knapsack problem with series–parallel support," *Operations Research Letters,* vol. 30, pp. 159-166, 6// 2002.
- [9] B. A. Julstrom, "Evolving heuristically difficult instances of combinatorial problems," presented at the Proceedings of the 11th Annual conference on Genetic and evolutionary computation, Montreal, Québec, Canada, 2009.
- [10] B. A. Julstrom, "Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem," presented at the Proceedings of the 7th annual conference on Genetic and evolutionary computation, Washington DC, USA, 2005.
- [11] B. A. Julstrom, "Naive and heuristic permutation-coded genetic algorithms for the quadratic knapsack problem," presented at the Proceedings of the 14th annual conference companion on Genetic and evolutionary computation, Philadelphia, Pennsylvania, USA, 2012.
- [12] X. F. Xie and J. Liu, "A Mini-Swarm for the Quadratic Knapsack Problem," in *2007 IEEE Swarm Intelligence Symposium*, 2007, pp. 190-197.
- [13] M. A. K. Azad, A. M. A. C. Rocha, and E. M. G. P. Fernandes, "A simplified binary artificial fish swarm algorithm for 0–1 quadratic knapsack problems," *Journal of Computational and Applied Mathematics,* vol. 259, Part B, pp. 897-904, 3/15/ 2014.
- [14] Z. Yang, G. Wang, and F. Chu, "An effective GRASP and tabu search for the 0–1 quadratic knapsack problem," *Computers & Operations Research,* vol. 40, pp. 1176-1185, 5// 2013.
- [15] Y. Chen and J.-K. Hao, "An iterated “hyperplane exploration” approach for the quadratic knapsack problem," *Computers & Operations Research,* vol. 77, pp. 226-239, 2017/01/01/ 2017.

- [16] C. Patvardhan, S. Bansal, and A. Srivastav, "Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems," *Swarm and Evolutionary Computation,* vol. 26, pp. 175-190, 2// 2016.
- [17] C. Patvardhan, S. Bansal, and A. Srivastav, "Solving the 0–1 Quadratic Knapsack Problem with a competitive Quantum Inspired Evolutionary Algorithm," *Journal of Computational and Applied Mathematics,* vol. 285, pp. 86-99, 9// 2015.
- [18] B. Jarboui, A. Sifaleras, A. Rebai, S. Toumi, M. Cheikh, and B. Jarboui, "The 3rd International Conference on Variable Neighborhood Search (VNS'14)0 – 1 Quadratic Knapsack Problem solved with VNS algorithm," *Electronic Notes in Discrete Mathematics,* vol. 47, pp. 269-276, 2015/02/01 2015.
- [19] S. Pulikanti and A. Singh, "An Artificial Bee Colony Algorithm for the Quadratic Knapsack Problem," in *Neural Information Processing: 16th International Conference, ICONIP 2009, Bangkok, Thailand, December 1-5, 2009, Proceedings, Part II*, C. S. Leung, M. Lee, and J. H. Chan, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 196-205.
- [20] B. Jarboui, A. Sifaleras, A. Rebai, S. Toumi, M. Cheikh, and B. Jarboui, "0 – 1 Quadratic Knapsack Problem solved with VNS algorithm," *Electronic Notes in Discrete Mathematics,* vol. 47, pp. 269-276, 2015/02/01 2015.
- [21] A. LaTorre, S. Muelas, J. M. Pe, and x00F, "Multiple Offspring Sampling in Large Scale Global Optimization," in *2012 IEEE Congress on Evolutionary Computation*, 2012, pp. 1-8.
- [22] D. Pisinger, "David Pisinger's optimization codes Instances QKP," p. http://www.diku.dk/~pisinger/codes.html.
- [23] C. Helmberg, F. Rendl, and R. Weismantel, "A Semidefinite Programming Approach to the Quadratic Knapsack Problem," *Journal of Combinatorial Optimization,* vol. 4, pp. 197-215, 2000.
- [24] L. Létocart, A. Nagih, and G. Plateau, "Reoptimization in Lagrangian methods for the 0–1 quadratic knapsack problem," *Computers & Operations Research,* vol. 39, pp. 12-18, 1// 2012.
- [25] S. Luke, "Essentials of Metaheuristics," vol. Second Edition, 2014.
- [26] A. B. a. E. Soutif, "Instances QKP," p. http://cedric.cnam.fr/~soutif/QKP/, 19/04/2005 2005.
- [27] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research,* vol. 24, pp. 1097-1100, 1997/11/01/ 1997.
- [28] T. A. Feo and M. G. C. Resende, "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization,* vol. 6, pp. 109-133, March 01 1995.