Universidad Veracruzana

Instituto de Investigaciones en Inteligencia Artificial

Un estudio de operadores genéticos de cruza para el Problema de Empacado de Objetos en Contenedores

T E S I S

que para obtener el título de:

Maestra en Inteligencia Artificial

presenta:

Stephanie Amador Larrea

Directora:

Dra. Marcela Quiroz Catellanos

Codirector:

Dr. Guillermo de Jesús Hoyos Rivera

Xalapa, Veracruz, México 2022

A mi madre, quien fue mi motivo e inspiración, que en el lugar en donde se encuentre, esté orgullosa de lo que he logrado.

Agradecimientos

Una vez alguien me dijo que todo trabajo tiene una recompensa, y hoy no tengo duda de ello.

Quiero agradecer a CONACYT, por el apoyo económico brindado durante la realización de este proyecto de investigación correspondiente a mis estudios de maestría. También quiero agradecer a la Universidad Veracruzana y especialmente al Instituto de Investigaciones en Inteligencia Artificial, quienes me apoyaron durante mi estancia de estudios.

Quiero agradecer enormemente a mi directora de tesis, la Dra. Marcela Quiroz Castellanos, quien estuvo apoyándome, ayudándome, enseñándome y sobre todo guiándome durante este proyecto de investigación. Agradezco su paciencia y esfuerzo, ya que sin ella este trabajo no podría haber sido concluido de manera satisfactoria en tiempo y forma.

Quiero agradecer al Dr. Guillermo de Jesús Hoyos Rivera, mi codirector de tesis, por todo su apoyo y enseñanzas durante este proyecto de investigación.

Quiero agradecer a todos mis profesores del programa de Maestría en Inteligencia Artificial, por su apoyo y paciencia en los momentos tan difíciles que pasé al comenzar el proyecto, así como también agradezco el conocimiento que me transmitieron y la ayuda que me brindaron para concluir provechosamente mis experiencias educativas.

Agradezco a mi padre, mi hermana, y mis abuelas por toda su comprensión, amor y apoyo para lograr esta meta en mi camino, siempre alentándome, dándome ánimos para seguir adelante cuando más triste me encontraba y creyendo en mí y el potencial que tengo. Agradezco todo el esfuerzo que realizaron para verme llegar tan lejos, cumpliendo mis sueños.

Agradezco a mi madre por alentarme a tomar la decisión de realizar mi proceso para ingresar al programa y estar completamente segura de que sería admitida, le agradezco infinitamente ser una motivación e impulso, en donde quiera que se encuentre.

Agradezco a toda mi familia por estar ahí de manera incondicional, motivándome y siendo un pilar sumamente importante.

Finalmente, quiero agradecer a mis amigos, por toda su ayuda, paciencia y amor en este proceso, por compartirme sus conocimientos, por ayudarme a aprender cosas nuevas, explicarme y por estar ahí siempre para escucharme en los momentos más difíciles.

Resumen

En este trabajo de investigación se realiza un estudio experimental sobre operadores genéticos de cruza para el Algoritmo Genético de Agrupación con Transmisión de Genes Controlada (GGA-CGT) que resuelve el problema de empacado de objetos en contenedores de una dimensión (1D-BPP), un problema muy estudiado durante las últimas cuatro décadas, con múltiples aplicaciones en la logística, la industria, las telecomunicaciones, el transporte, entre otros. Debido a su complejidad, este problema ha sido abordado usando diferentes técnicas, destacando el desempeño de los algoritmos híbridos y las heurísticas. Dentro de las propuestas del estado del arte más competitivas se encuentra el GGA-CGT, que ha superado la efectividad de los mejores algoritmos en bancos de prueba de alta dificultad. Sin embargo, estudios experimentales revelaron que, para un importante número de nuevos casos de prueba de 1D-BPP, el operador de cruza del GGA-CGT no parecía conducir a mejores soluciones. Por esta razón se realizó un estudio experimental de operadores de cruza orientados a grupos dentro del estado del arte, utilizando casos de prueba de 1D-BPP con diferentes características y un alto grado de dificultad, con el objetivo estudiar el impacto que cada operador puede tener en el rendimiento final del GGA-CGT. Con la finalidad de identificar las características que debe tener un operador de cruza para contribuir de manera significativa al desempeño del GGA-CGT se analizaron diferentes características involucradas en las operaciones de cruza. Como resultado de este estudio experimental, se obtuvo conocimiento del dominio del problema y se utilizó para diseñar un nuevo operador de cruza. Los resultados experimentales indican que el rendimiento del algoritmo GGA-CGT mejora considerablemente al reemplazar el operador de cruza original por el nuevo, con respecto al número de casos de prueba resueltos óptimamente y al número de generaciones utilizadas, con una tasa de mejora en la efectividad del 25 % en

el banco de prueba BPPv_u_c, considerado uno de los más difíciles del estado del arte. El GGA-CGT con el nuevo operador de cruza supera la efectividad de los mejores algoritmos del estado del arte y su eficiencia es muy alta en comparación con el número de iteraciones requeridas por dichos algoritmos.

Aportaciones del trabajo de tesis

Las aportaciones que se obtuvieron durante la realización de este trabajo de investigación son mencionadas en los siguientes puntos:

1. Artículo "An Experimental Study of Grouping Crossover Operators for the Bin Packing Problem" aceptado para ser publicado en el número especial **Emerging Issues and Applications of Fuzzy Systems Neural Networks, and Metaheuristics**, de la revista Computación y Sistemas. 2022,

2. Participación en el International Seminar of Computational Intelligence (ISCI 2021), Tijuana, con el trabajo: "An Experimental Study of Grouping Crossover Operators for the Bin Packing Problem".

3. Participación en el 9th International Workshop on Numerical and Evolutionary Optimization (NEO2021), México, con el trabajo: "An Experimental Study of Grouping Crossover Operators for the Bin Packing Problem".

4. Participación en el 3nd Workshop on New Trends in Computational Intelligence and Applications (CIAPP 2021), Xalapa, con el trabajo: "An Experimental Study of Grouping Crossover Operators for the Bin Packing Problem".

5. Participación en el 7.o Seminario De Socialización del Aprendizaje Computacional (SAC), Xalapa, con el trabajo: "Inteligencia Artificial para problemas duros".

Índice general

Índice de figuras

Capítulo 1

Introducción

El problema del Empacado de Objetos en Contenedores en Una Dimensión (1D-BPP por sus siglas en inglés One-Dimensional Bin Packing Problem) ha sido estudiado a lo largo de las últimas cuatro décadas, debido a su importancia y múltiples aplicaciones en la logística, la industria, las telecomunicaciones, el transporte, entre otros. Está clasificado como un problema NP-Duro, en virtud de la complejidad que se tiene para encontrar una solución óptima en un corto tiempo. Debido a lo anterior, un conjunto de estrategias que se utiliza para resolverlo pertenecen al paradigma de Cómputo Evolutivo.

A lo largo de los años, se han propuesto distintos algoritmos Genéticos para resolver el 1D-BPP, uno de los más sobresalientes es el Algoritmo Genético de Agrupación (GGA) propuesto por Falkenauer y Delchambre (1992). Este Algoritmo Genético, a diferencia de los tradicionales, plantea tres componentes: (1) un nuevo esquema de representación de soluciones basado en grupos que son vistos como genes, (2) operadores de variación para representación de grupos adecuados al problema y (3) una función de aptitud que evalúa qué tanto se aprovecha la capacidad de los contenedores.

En Quiroz-Castellanos *et al.* (2015), se propuso el Algoritmo Genético de Agrupación con Transmisión de Genes Controlada (GGA-CGT, por sus siglas en inglés). Está inspirado en el trabajo realizado por Falkenauer, también se propone ver los grupos como genes y utiliza operadores de variación orientados a grupos, pero con la característica de promover la transmisión de los mejores grupos y controlar el balance entre la presión selectiva y la diversidad de la población, obteniendo un mejor desempeño que el GGA original.

A pesar de la eficacia del GGA-CGT, estudios experimentales han mostrado que algunos operadores de variación no tienen un impacto significativo en los resultados obtenidos al resolver clases de pruebas con diferentes características, específicamente los operadores de cruza. Este trabajo se centra en el estudio del impacto de las heurísticas y operadores de variación que involucran al algoritmo GGA-CGT, puesto que el buen desempeño del algoritmo depende de la correcta interacción entre operadores y heurísticas.
En el proyecto propuesto se busca la creación de un operador de cruza nuevo con el objetivo de mejorar el impacto de dicho operador en el desempeño del algoritmo GGA-CGT para poder encontrar la solución óptima de un mayor número de casos de prueba de los bancos de prueba que han resultado difíciles para los algoritmos dentro del estado del arte, además de reducir el número de generaciones necesarias para llegar a los resultados deseados.

1.1. Descripción del problema

El GGA es una extensión del Algoritmo Genético (GA) tradicional, que utiliza un esquema de representación basado en grupos y operadores de variación adaptados para trabajar a nivel de grupos. Desde su aparición, los GGAs se han utilizado para abordar varios problemas de agrupamiento con características distintas. Por lo tanto, en la actualidad existen diferentes operadores de variación desarrollados para resolver problemas con diversas restricciones y condiciones de agrupamiento. La calidad de los operadores de variación es un factor de suma importancia en la eficacia y eficiencia de un GGA. Por lo tanto, se requiere integrar operadores de variación adecuados, diseñados de acuerdo con el dominio del problema.
Dada la complejidad del 1D-BPP, considerado NP-Duro, se busca crear un nuevo operador de cruza que mejore el desempeño del algoritmo GGA-CGT, propuesto por Quiroz-Castellanos *et al.* (2015), con el objetivo de resolver un mayor número de casos de prueba de las clases más difíciles de manera óptima en un tiempo menor con respecto al número de generaciones requeridas. Algunas clases de pruebas son muy difíciles de resolver debido a la capacidad del contenedor, la dispersión de los pesos de los objetos y la diferencia entre el mayor y el menor peso de los objetos. Dentro de los casos de prueba para poner a prueba

el nuevo operador de cruza, se encuentran los del banco de prueba clásico para 1D-BPP y los de alta dificultad BPPv_u_c propuestas en Carmona-Arroyo *et al.* (2021).

1.2. Justificación

Como se ha mencionado anteriormente, el 1D-BPP es considerado un problema del tipo NP-Duro para el cual no existe un algoritmo exacto que sea capaz de encontrar una solución óptima de manera eficiente en todos los casos de prueba. Cuando se habla del 1D-BPP, se tiene una conexión inmediata con el problema de la explosión combinatoria. Esto quiere decir que, si se aumenta el número de variables, la cantidad de soluciones posibles crece de manera exponencial. Debido a este problema, se recurre a metaheurísticas tal como lo es el algoritmo GGA-CGT, el cual es considerado uno de los mejores dentro del estado del arte para el 1D-BPP, por su eficiencia en tiempo y en número de casos de prueba resueltos óptimamente. Además, ha sido uno de los mejores del estado del arte cuando se trata de resolver casos de prueba complejos como las pertenecientes a la clase Hard28 (debido a que los contenedores de la solución deben estar llenos casi al 100 % y el porcentaje de objetos grandes en el problema) (Quiroz-Castellanos *et al.*, 2015).

A pesar de sus buenos resultados con el banco de prueba clásico, no muestra el mismo desempeño para todos los casos de prueba, ya que además de las pertenecientes a el banco clásico, cuando se ha ejecutado para resolver las pertenecientes al nuevo banco de prueba de alta dificultad BPPv_u_c no se tiene un buen desempeño. El nuevo banco de prueba BPPv_u_c ha resultado ser un reto para los mejores algoritmos del estado del arte, los cuales han mostrado un pobre desempeño para encontrar la solución óptima de un gran número de casos de prueba (Carmona-Arroyo *et al.*, 2021; González-San-Martín, 2021).

Debido a lo anterior, se tiene como objetivo crear un nuevo operador de cruza para el algoritmo GGA-CGT, con lo que se busca resolver de manera óptima los casos de prueba más complejas y los casos de prueba nuevos, ya que los estudios han demostrado que el operador de cruza que tiene implementado como predeterminado el algoritmo GGA-CGT, no tiene un impacto significativo en el desempeño final del mismo.

Es importante resaltar que este trabajo dará un aporte a potenciales aplicaciones en el

campo de las ciencias de la computación, matemáticas, ingeniería, manufactura, procesos de producción, así como problemas aplicados, entre otros, debido al extenso número de aplicaciones del 1D-BPP que, además de que aparece con frecuencia como un subproblema en varios problemas prácticos en diferentes áreas.
Más aún, se espera que este trabajo motive el desarrollo de nuevos operadores de variación para abordar otros problemas de agrupamiento. Asimismo, el conocimiento presentado en este trabajo puede ser útil para mejorar el desempeño de los GGAs existentes, adaptando sus operadores de acuerdo a las conclusiones alcanzadas en esta tesis.

1.3. Objetivos

1.3.1. General

Mejorar el desempeño del algoritmo GGA-CGT para la resolución del 1D-BPP en términos del número de soluciones óptimas encontradas y el número de generaciones que le toma encontrarlas, esto mediante la modificación del operador de cruza.

1.3.2. Particulares

- Estudiar el desempeño del algoritmo GGA-CGT con el operador de cruza predeterminado.
- Revisar los operadores de cruza existentes dentro del estado del arte.
- Implementar los operadores de cruza orientados a grupos dentro del estado del arte.
- Comparar el desempeño del GGA-CGT con los distintos operadores de cruza implementados.
- Comparar las características de los operadores implementados.
- Buscar la estrategia de reemplazo adecuada para el mejor operador.
- Identificar las características que debe tener el operador de cruza para contribuir de manera significativa al algoritmo.

- Encontrar las características que causan que un operador sea mejor que los otros.
- Diseñar un nuevo operador de cruza que integre las mejores características para superar el desempeño de los operadores del estado del arte.

1.4. Hipótesis

H1: Es posible diseñar un operador de cruza que mejore el desempeño del algoritmo GGA-CGT, resolviendo óptimamente un mayor número de casos de prueba y reduciendo el número de generaciones requeridas para los casos de prueba de cada clase.

1.5. Alcances y limitaciones

Este trabajo está enfocado exclusivamente en el algoritmo GGA-CGT, propuesto por Quiroz-Castellanos *et al.* (2015), específicamente para el problema del empacado de objetos en contenedores de la variante clásica unidimensional.

1.6. Organización del documento

El trabajo presentado está organizado de la siguiente manera. En el Capítulo 2 se describen los conceptos necesarios para entender el problema, el método de solución y las definiciones básicas referentes al objeto de investigación. En el Capítulo 3 se presentan y describen a los mejores operadores de cruza para grupos dentro del estado del arte, se realiza un estudio experimental sobre el desempeño de estos operadores de cruza, se comparan los resultados obtenidos y se destacan las principales conclusiones. En el Capítulo 4 se muestra la metodología utilizada para llevar a cabo la creación del nuevo operador de cruza, así como los resultados obtenidos en cada etapa de la misma. En el Capítulo 5 se describe e ilustra el funcionamiento del nuevo operador de cruza, así mismo, se presenta la nueva configuración de parámetros, la prueba de robustez y los resultados de la ejecución a largo plazo. Finalmente en el Capítulo 6 se presentan las conclusiones de esta investigación

y se propone el trabajo futuro.

Capítulo 2

Marco teórico

2.1. Problemas de optimización

Un problema de optimización se define como aquel en el que se busca minimizar o maximizar el valor de una función, es decir, dado un espacio S de soluciones factibles (espacio de búsqueda) y $f : S \rightarrow R$ una función objetivo a optimizar, $\forall s \in S \; f(s) = x$. La función objetivo define un orden total entre cualquier par de soluciones en el espacio de búsqueda.

Definición *Una solución $s* \in S$ se llama óptimo global si:*

$$\forall s \in S, f(s*) \leq f(s) \tag{2.1}$$

En los problemas de optimización se tiene como objetivo encontrar el mínimo global, es decir $s*$ (Talbi, 2009).

Para solucionar problemas de optimización, existen dos tipos de métodos, (1) los métodos exactos, los cuales proporcionan soluciones óptimas, sin embargo, son muy costosos computacionalmente; algunos ejemplos de métodos exactos son: programación dinámica, fuerza bruta, corte y poda, entre otros. (2) Los métodos aproximados, que generan soluciones de buena calidad en un periodo de tiempo razonable, e incluso, es posible obtener el óptimo, pero no se garantiza; estos algoritmos se dividen en algoritmos de aproximación y algoritmos heurísticos (Talbi, 2009).

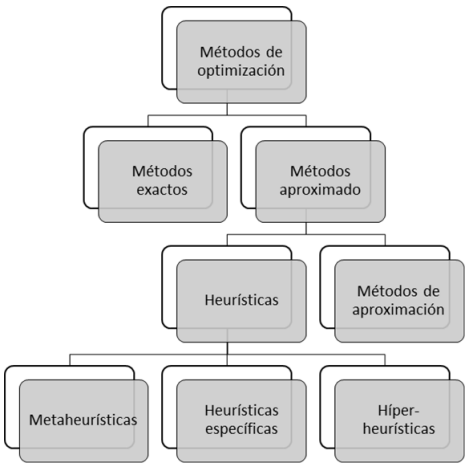


Figura 2.1: Métodos de optimización

Los métodos de aproximación generan soluciones de buena calidad que pueden ser demostrables, realizándolo con límite en los tiempos, el cual también puede ser demostrable. Con respecto a los algoritmos heurísticos, éstos son utilizados cuando se tienen problemas de gran tamaño. Dentro de los algoritmos heurísticos, existe una subclasificación: las metaheurísticas y las heurísticas específicas. Las metaheurísticas son algoritmos generales que pueden ser utilizados para resolver casi cualquier problema de optimización, mientras que las heurísticas específicas suelen utilizarse para problemas específicos. Por otro lado, las híper-heurísticas, son técnicas que se utilizan para seleccionar de dentro de un conjunto de heurísticas, la mejor para resolver el problema o caso de prueba (Sim *et al.*, 2012). En la Figura 2.1 se muestra un diagrama de los métodos de optimización.

2.2. El Problema de empacado de objetos en contenedores

El Problema de Empacado de Objetos en Contenedores de una dimención (One-Dimensional Bin Packing Problem, 1D-BPP) se define como se describe a continuación. Dado un conjunto de n objetos y dados n contenedores, siendo w_j correspondiente al peso del j-ésimo objeto y c correspondiente a la capacidad máxima de los contenedores, se desea asignar cada objeto a un contenedor, tal que la suma de los pesos de los objetos no excede la ca-

pacidad c del contenedor y además, se tiene como objetivo que el número de contenedores usados sea el mínimo. La formulación matemática se desarrolla a continuación:

$$\text{minimizar } z = \sum_{i=1}^{n} y_i \tag{2.2}$$

En donde z corresponde al número de contenedores usados, y_i será 1 si el contenedor i se encuentra usado y 0 en otro caso. Además, estará sujeto a las siguientes restricciones:

$$\sum_{j=1}^{n} w_j x_{ij} \leq c y_i, \ i \in N = \{1, \ldots, n\} \tag{2.3}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \ j \in N = \{1, \ldots, n\} \tag{2.4}$$

$$y_i = 0 \text{ or } 1, \ i \in N = \{1, \ldots, n\} \tag{2.5}$$

$$x_{ij} = 0 \text{ or } 1, \ i, j \in N = \{1, \ldots, n\} \tag{2.6}$$

La condición de la Ecuación 2.3 indica que la suma de los pesos de los objetos j que sean asignados a cada contenedor i ($x_{ij} = 1$) deberá ser menor o igual a la capacidad c del contenedor, la condición de la Ecuación 2.4 asegura que cada objeto j esté empacado exactamente en un contenedor i; en donde x_{ij} tomará el valor de 1 si el objeto j se encuentra en el contenedor i y tomará el valor de 0 en otro caso (Martello, 1990). Por otro lado, y_{ij} indica a que contenedor será utilizado, se le asignará un 1 si el contenedor es usado y un 0 en otro caso.

La dificultad principal de este problema se centra en la cantidad de recursos utilizados para resolver la solución, esto debido a que es considerado un problema en donde para encontrar una solución óptima, se deben considerar todas las combinaciones de los n objetos en m contenedores, en donde la explosión combinatoria se hace presente. El número de posibles combinaciones está definido como $(n/2)^{n/2}$ (Basse, 1998). Debido a lo anterior, se recurre a metaheurísticas que proporcionan soluciones de calidad en un periodo de tiempo corto.

Tabla 2.1: Variantes del problema de empacado de objetos en contenedores, con las características de los objetos, los contenedores y el objetivo general.

Variantes	Características		Objetivo
	Objetos	Contenedores	
1D-BPP	-Un peso w_{ij}	-Una capacidad de peso c	Almacenar los objetos en los contenedores de modo que los contenedores usados sean lo mínimos y su capacidad no sea excedida.
2D-BPP	-Son rectangulares -Una anchura w_j -Una altura h_j	-Definidos con medidas de altura H y anchura W	Almacenar los objetos en los contenedores de modo que los contenedores usados sean lo mínimos, sujeto a la condición de que los objetos no pueden colocar uno encima de otro y los objetos tienen una orientación fija.
3D-BPP	-Son rectangulares -Una altura h_j -Una anchura w_j -Una profundidad d_j.	-Definidos con medidas de altura H, anchura W y profundidad D.	Almacenar los objetos en los contenedores de modo que los contenedores usados sean lo mínimos, sujeto a la condición de que los objetos no pueden girarse y los objetos no necesitan estar estar almacenados en niveles.

Generalmente los métodos de solución son estrategias inteligentes y sencillas.
En la Tabla 2.1 se muestran las tres variantes del BPP, en donde se presentan como son las características de los objetos, los contenedores y el objetivo de cada variante, con la finalidad de observar las diferencias entre cada una de ellas.

2.3. Algoritmos del estado del arte para la solución de 1D-BPP

Al ser uno de los problemas de mayor interés en el área de la optimización, en los últimos años, se han implementado distintas técnicas para encontrar la mejor solución del 1D-BPP. Dentro de las técnicas que más destacan, se encuentran los algoritmos híbridos y las heurísticas (Quiroz-Castellanos *et al.*, 2015). En Kämpke (1988) se propone un algoritmo de recocido simulado para la solución del 1D-BPP, sin embargo, éste necesita de mucho tiempo computacional y el ajuste de los parámetros tiene que hacerse con cuidado, mediante prueba y error, lo que resultó en desventajas. Dentro del estado de arte, uno de los algoritmos utilizados para resolver el 1D-BPP, es el Algoritmo Genético Híbrido de Agrupación (HGGA), fue propuesto en Falkenauer (1996), el cual logró ser uno de los más eficientes para la solución de los casos de prueba de prueba de la literatura.
Más tarde, en Alvim *et al.* (2004) se propuso una heurística de mejora híbrida (HI_BP). En esta heurística, se usaron las siguientes características: (1) uso de estrategias de límites

inferiores, (2) la generación de soluciones iniciales por referencia al problema dual min-max de la carga basada en la dominancia, (3) la diferenciación y el desequilibrio y (4) la inclusión de un proceso de mejora que utiliza la búsqueda tabú (Alvim *et al.*, 2004). Otro algoritmo fue propuesto en Singh y Gupta (2007), en el que se presenta un algoritmo genético de agrupación (H-SGGA), combinando un algoritmo y una heurística. La heurística se deriva de la propuesta en Fleszar y Hindi (2002). En este algoritmo se utiliza una representación de los cromosomas como un conjunto de contenedores y, por tanto, el resultado de la aplicación de los operadores genéticos depende sólo del contenido de los contenedores y no del ordenamiento relativo de los mismos. El algoritmo de búsqueda de vecinos que hace uso del recocido de pesos para escapar de óptimos locales (WA) fue propuesto en Loh *et al.* (2008). Un algoritmo que tuvo resultados importantes es el Pertubation-SAW MBS, el cual es una versión mejorada del algoritmo MBS (Fleszar & Hindi, 2002), este algoritmo fue propuesto en Fleszar y Charalambous (2011). En este algoritmo, se propone un método para controlar el peso medio de los objetos empacados de una heurística para los contenedores, además, introduce una heurística constructiva y una heurística de mejora, esto con el objetivo de mejorar las soluciones, ya que guardar objetos muy pequeños en los contenedores, producía soluciones pobres (Fleszar & Charalambous, 2011). La Hiper Heurística con módulo clasificador (HHC-BP) se propuso en Sim *et al.* (2012), esta híper heurística utiliza un algoritmo evolutivo con el que evoluciona los atributos que caracterizan los casos de prueba. Un algoritmo genético híbrido (HGGA-BP) se propone en Cruz-Reyes *et al.* (2012). Este algoritmo utiliza el esquema propuesto por Falkenauer (1996) como inspiración, además, propone heurísticas eficientes para generar la población y realizar las operaciones de cruza y muta, también utiliza una estrategia híbrida para la reinserción de los objetos. En Sotelo-Figueroa *et al.* (2013), se propone el Algoritmo de Evolución Diferencial y Representación Indirecta (μDE), en donde se implementa la evolución micro diferencial para evolucionar una representación indirecta. En Bayraktar *et al.* (2014), los autores propusieron un algoritmo de colonia de abejas artificiales con una búsqueda local (MEABC), el cual mostró resultados alentadores. Otro Algoritmo Genético Híbrido fue propuesto en Kaaouache y Bouamama (2015), el HGBF_BP. Este algoritmo utiliza la heurística Best Fit Drecreasing (BFD) (Martello, 1990) para reparar las solucio-

nes inconsistentes. En Brandão y Pedroso (2016), los autores presentan una técnica para la solución del 1D-BPP. Este método exacto tiene como estrategia principal el flujo de arco y un algoritmo de compresión de gráficos; se observaron muy buenos resultados para resolver los casos de prueba pertenecientes al banco de prueba clásico. Buljubašić y Vásquez (2016) propusieron el algoritmo CNS_BP , el cual consiste en una búsqueda de vecindarios para la solución del 1D-BPP para encontrar una solución factible, cuando ya se ha dado un número de contenedores. Este algoritmo, es considerado uno de los mejores del estado del arte, ya que ha logrado resolver óptimamente 1612 de 1615 casos de prueba de prueba de la literatura. Ozcan *et al.* (2016a) propusieron un algoritmo genético de agrupación en GPU utilizando CUDA (1D-BPP-CUDA), en el que utilizan la GPU para realizar las operaciones de cruza y muta para reducir el tiempo de ejecución. En Kucukyilmaz y Kiziloz (2018) propusieron el Algoritmo Genético de Agrupación Paralelo a la Isla (IPGGA). En Borgulya (2020), se propuso un algoritmo evolutivo híbrido, con la particularidad de que no utiliza un operador de cruza y utiliza dos operadores de mutación, mostrando tener un buen resultado con las clases de pruebas más difíciles. González-San-Martín (2021), propusieron un Algoritmo Genético de Agrupación con diversificación (GGA-CGT/D), una mejora para el GGA-CGT (Quiroz-Castellanos *et al.*, 2015), en este algoritmo se propuso una reducción del problema y una técnica de diversificación, mostrando muy buenos resultados para la solución de los casos de prueba más difíciles para el 1D-BPP.

Como se describió anteriormente, dentro de la literatura se encuentran distintos métodos de solución, como lo son métodos exactos, métodos heurísticos y métodos híbridos. Se han obtenido buenos resultados, en los que sobresale el desempeño de los métodos heurísticos. De los algoritmos anteriores, sobre salen el algoritmo GGA, el cual fue el primero en introducir una representación adecuada para los problemas de agrupación, así como operadores de variación adecuados a la misma, el algoritmo GGA-CGT, el cual obtuvo un gran desempeño para resolver el banco de prueba clásico, obteniendo 1602 óptimos de los 1615 casos de prueba, la modificación del algoritmo anterior, el algoritmo GGA-CGT/D el cual logró obtener los 1615 óptimos, el algoritmo CNS_BP que obtuvo un total de 1612 óptimos de 1615 y el algoritmo de flujo de arco, el cual también muestra desempeño significativos para resolver el banco de prueba clásico, encontrando los 1615 óptimos. Estos

algoritmos los últimos tres algoritmos son consideras los mejores dentro del estado del arto, por lo que serán los utilizados para comparar los resultados obtenidos en este trabajo de investigación.

2.4. Bancos de prueba para 1D-BPP

2.4.1. Banco de pruebas clásico para 1D-BPP

Este banco de pruebas está conformado por 1615 casos de prueba, divididas en 9 clases. Para estos casos de prueba, el valor de la solución óptima es conocido. Las clases de pruebas Uniform y Triplets fueron propuestas por Falkenauer (1996).

- **Uniform** : Compuesta por 80 casos de prueba, en los cuales los pesos de los objetos están distribuidos de manera uniforme en el intervalo [20,100], la capacidad de los contenedores es $c = 150$, con objetos $n = 120$, 250, 500 y 1000, se tienen 20 casos de prueba por cada número de objetos: u_20, u_250, u_500, u_1000.

- **Triplets**: Compuesta por 80 casos de prueba, con la característica de que cada contenedor debe guardar 3 objetos que lo llenan a su máxima capacidad (completamente), por lo que la solución óptima está dada por $n/3$, en donde n es el número de objetos. Además, $n = 60$, 120, 249 y 501. Los pesos están distribuidos entre 25 y 50, la capacidad del contenedor es $c = 100$. Se tienen 20 casos de prueba por cada número de objetos: t_60, t_120, t_249 y t_501.

Loas clases de prueba Dataset 1, Dataset 2 y Dataset 3, fueron propuestas por Scholl *et al.* (1997). Aquí varían el número de objetos, la capacidad de los contenedores y los pesos de los objetos.

- **Dataset 1**: La clase está compuesta por 720 casos de prueba, con distintos n objetos, $n = 50$, 100, 200 y 500. Se cuentan con capacidades $c = 100$, 120 y 150. Los pesos w se encuentran generados en distribuciones uniformes en los siguientes intervalos: [1,100], [20,100] y [30,100].

- **Dataset 2**: La clase está compuesta por 480 casos de prueba, con pesos generados mediante una distribución uniforme. Se tiene n objetos, con n= 50, 100, 200 y 500. La capacidad de los contenedores es $c = 100$. Los pesos se encuentran uniformemente distribuidos en el intervalo [100, 300].

- **Dataset 3**: Esta clase está formada por 10 casos de prueba. Se tiene un número de objetos $n = 200$, en donde la capacidad $c = 100000$. Los pesos se encuentran uniformemente distribuidos en el intervalo [20000, 35000]. A cada contenedor le cabe un total de entre 3 y 5 objetos.

El tercer grupo está conformado por las clases was 1, was 2 y gau 1. Estas fueron propuestas por Schwerin y Wäscher (1997) y Wiischer y Gau (1996).

- **Was 1**: Compuesta por 100 casos de prueba, con un total de $n = 100$ objetos cada una y contenedores de capacidad $c = 1000$. Los pesos están distribuidos en el intervalo [150, 200].

- **Was 2**: Compuesta por 100 casos de prueba, con un total de $n = 120$ objetos cada una y contenedores de capacidad $c = 1000$. Los pesos están distribuidos en el intervalo [150, 200].

- **Gau 1**: Compuesta por 17 casos de prueba, con objetos n entre 57 y 239, y contenedores de capacidad $c = 10000$. Los pesos están distribuidos en el intervalo [2,7332].

La última clase de pruebas, es la Hard 28, estos casos de prueba fueron propuestas por Schoenfield (2002).

- **Hard 28**: Compuesta por 28 casos de prueba, en donde la capacidad de los contenedores es $c = 1000$ y los pesos de los objetos están distribuidos de manera uniforme en el intervalo [1,800]. Además, el número de objetos n varía entre 160 y 200. Es considerada la clase de pruebas clásica más difícil.

2.4.2. Banco de prueba BPPv_u_c

Este banco de prueba fue propuesto en Carmona-Arroyo *et al.* (2021). Está compuesto por clases que contienen 7 clases de pruebas cada una, y los conjuntos incluye 100 casos de

prueba por cada capacidad de contenedor c diferente. La capacidad de los contenedores esta dada con $c \in (10^2, \ldots, 10^8)$, sus características particulares se mencionan a continuación.

- **BPP.25** En la primera clase el número de objetos n varía en un rango de [110,154], los pesos son distribuidos uniformemente entre $(0, 0.25c]$ y la solución óptima es igual a 15 contenedores.
- **1BPP.5** En la segunda clase el número de objetos n varía en un rango de [124,167], los pesos son distribuidos uniformemente entre $(0, 0.5c]$ y la solución óptima es igual a 30 contenedores.
- **BPP.75** En la tercera clase el número de objetos n varía en un rango de [132,165], los pesos son distribuidos uniformemente entre $(0, 0.75c]$ y la solución óptima es igual a 45 contenedores.
- **BPP1** En la cuarta clase el número de objetos n varía en un rango de [148,188], los pesos son distribuidos uniformemente entre $(0, c]$ y la solución óptima es igual a 60 contenedores.

2.5. Algoritmo Genético de Agrupación con Transmisión Controlada de Genes Controlada GGA-CGT

Dentro de los métodos de solución más utilizados para resolver los problemas de agrupamiento destacan los Algoritmos Genéticos (GA, por sus siglas del inglés Genetic Algorithm). Los GA son una metaheurística perteneciente al paradigma del cómputo evolutivo, están inspirados en la teoría de la Evolución Natural y la Evolución Genética. Un GA cuenta con los siguientes componentes: un esquema de representación para las soluciones (cromosoma), una población de soluciones y operadores genéticos para transformar las soluciones actuales en nuevas soluciones (cruza, mutación, etc.). Dentro de los GA se encuentra el Algoritmos Genéticos de Agrupación (GGA, por sus siglas del inglés Grouping Genetic Algorithm), un algoritmo especializado para resolver problemas de agrupamiento. El GGA fue propuesto por Falkenauer y Delchambre (1992). Con este algoritmo se busca

eliminar los problemas que se tenían cuando se utilizaba el algoritmo Genético al trabajar con estructuras que son grupos. Para esto, en este algoritmo el esquema utilizado los grupos se ven como genes y además se utilizan operadores de variación adaptados a los problemas de agrupamiento.

En el Algoritmo 1 se muestra el procedimiento general del algoritmo GGA-CGT (Quiroz-Castellanos *et al.*, 2015). En este algoritmo, la variable P corresponde a la Población de individuos y la variable max_gen corresponde al máximo de generaciones a realizarse. Para los operadores de variación la variable correspondiente a el porcentaje de la población a cruzar es n_c y la del porcentaje a mutar es n_m.

Algoritmo 1: Algoritmo Genético de Agrupación con Transmisión de Genes Controlada

Resultado: Mejor Solución encontrada
Generar población inicial P, con FF-$\tilde{n}$;
mientras *generación* $<$ *max_gen y Tamaño (mejor_solución)* $> L_2$ **hacer**
  Seleccionar n_c individuos para cruzar por medio de Selección_Controlada;
  Aplicar Cruzamiento_Nivel_Gen+FFD a los n_c individuos seleccionados;
  Aplicar Reemplazo_Controlado para introducir los hijos;
  Seleccionar n_m individuos y clonar soluciones élite por medio de Selección_Controlada;
  Aplicar Mutación_Adaptativa+RP a los mejores n_m individuos;
  Aplicar Reemplazo_Controlado para introducir los clones;
  Actualizar *mejor_solucion_global*
fin

2.5.1. Función de aptitud

En el 1D-BPP, se busca minimizar el número de contenedores utilizados para guardar los objetos. Por su parte, el GGA-CGT tiene como objetivo maximizar los valores de la aptitud de la población. En la función de aptitud utilizada, se espera que algunos contenedores estén casi vacíos y otros casi llenos para poder acomodar los objetos restantes sin necesidad de abrir nuevos contenedores. La función de aptitud se describe como en la Ecuación 2.7, mostrada a continuación:

$$F_{BPP} = \Sigma_{i=1}^{m} \frac{(S_i/C)^2}{m}. \tag{2.7}$$

En donde m es el número de contenedores en la solución, S_i es la suma de los pesos de los objetos en el contenedor i-ésimo y c corresponde con la capacidad de los contene-

dores. Esta ecuación propuesta por Falkenauer y Delchambre (1992), tiene como objetivo maximizar los valores de la aptitud de los individuos de la población, es decir, maximizar el llenado promedio de los contenedores de la solución.

2.5.2. Población inicial

El algoritmo GGA-CGT genera la población inicial utilizando la heurística FF-$\tilde{n}$: esta heurística beneficia al mejorar el empacado de los objetos. El proceso es el siguiente: primero los objetos de peso mayor al 50 por ciento de la capacidad del contenedor, son acomodados en contenedores diferentes cada uno. Después, los objetos restantes son guardados de forma aleatoria en el primer contenedor que tenga la capacidad para guardarlos, es decir, con la bien conocida heurística First Fit sobre una permutación aleatoria de los objetos (Quiroz-Castellanos, 2014).

2.5.3. Operador de cruza

El operador de cruza consiste en combinar el material genético de dos soluciones para generar descendencia. En el GGA-CGT el operador de cruza usado es el Gene Level Crossover (Quiroz-Castellanos *et al.*, 2015; Ramos-Figueroa *et al.*, 2021), un operador que se centra en la contribución a nivel gen por parte de las soluciones padre. Este operador, a partir de dos soluciones padre p_1 y p_2 genera dos soluciones hijos c_1 y c_2. Primero se ordenan ambos padres de manera descendente, con respecto al llenado de cada gen (contenedor), con el objetivo de aumentar las probabilidades de conservar los mejores genes de los padres. Después los genes de ambos padres se comparan en paralelo, uno a uno, priorizando la transmisión del gen más lleno. Si ambos genes están igual de llenos, para el hijo c_1 se le da prioridad al padre p_1 y para el hijo c_2 al padre p_2. Si se da el caso que alguno de los padres tiene un mayor número de genes, éstos se heredan directamente a ambos hijos. Los hijos pueden ser inconsistentes, por lo que si algún gen tiene un objeto que ya está en la solución, se elimina y se liberan los objetos, después mediante la heurística First Fit Decreasing (FFD) se realiza la reinserción a la solución de éstos.

2.5.4. Heurística First Fit Decreasing

La heurística First Fit Decreasing (FFD) es clásica para resolver el 1D-BPP, es un método de solución sencillo. Durante el proceso, los objetos son considerados de forma decreciente con respecto a los pesos de los objetos; los objetos son asignados en el contenedor indexado más bajo que cuente con la capacidad para almacenarlo, solo en caso de no existir uno en el que quepa, abre un nuevo contenedor (Martello, 1990).

2.5.5. Operador de mutación

El operador de mutación tiene como objetivo realizar cambios menores a las soluciones, de manera aleatoria. El operador utilizado en el GGA-CGT es el Adaptative Mutation, el cual trabaja al nivel de los genes, promoviendo la transmisión de los mejores genes en el cromosoma (Ramos-Figueroa *et al.*, 2021). Este operador elimina genes (contenedores) de la solución. El número de contenedores a eliminar de una solución con m contenedores, se obtiene con una relación entre el tamaño de la solución y el número de genes incompletos. La ecuación se define a continuación:

$$n_B = [\imath \cdot \epsilon \cdot p_\epsilon] \tag{2.8}$$

En donde $\imath$ corresponde con los contenedores incompletos de la solución, ϵ la porción de eliminación y p_ϵ la probabilidad de eliminación.

$$\epsilon = \frac{(2 - (\imath/m))}{\imath^{(1/k)}} \tag{2.9}$$

$$p_\epsilon = 1 - U(0, \frac{1}{\imath^{1/k}})) \tag{2.10}$$

El parámetro que define la tasa de cambio de ϵ y p_ϵ es k ($k > 0$). La proporción de eliminación es inversamente proporcional a la cantidad de contenedores que se tienen incompletos $\imath$ y el porcentaje de contenedores incompletos $(\imath/m)$, lo que indica que mientras más pequeña es la solución, el porcentaje de contenedores a eliminar es mayor y viceversa.

2.5.6. Reacomodo por pares

Cuando se realiza la mutación, algunos objetos son liberados al modificar a la solución, los cuales deben ser reinsertados. El reacomodo por pares (RP, por sus siglas del inglés Rearrangement by Pairs) se realiza en dos etapas: (1) recorre todos los contenedores buscando un intercambio de pares de objetos empacados y libres, de modo que se mejore el llenado, (2) los objetos libres, son reinsertados usando una heurística FF. Este reacomodo, muestra mejorar las soluciones, además de generar un equilibrio entre la exploración y explotación del espacio de búsqueda.

2.5.7. Esquema de selección

Para seleccionar qué individuos de la población transmitirán su material genético, GGA-CGT utiliza un esquema de selección controlada para la cruza y mutación.
Al aplica el operador de cruza, n_c padres son seleccionados para generar n_c hijos. Los n_c padres son seleccionados en la siguiente forma: se generan dos conjuntos de padres. El grupo G, generado de seleccionar $n_c/2$ individuos aleatoriamente de los mejores n_c individuos de la población con una distribución uniforme y el grupo R generado de seleccionar $n_c/2$ individuos de manera aleatoria de la población (sin incluir a los individuos del grupo élite), sin repetir, con una distribución uniforme. Los elementos se van combinando por pares, tomando un individuo G_i de G y uno R_i de R $(0 \leq i < (n_c)/2)$. Para los n_m individuos a mutar, el operador es aplicado en dos fases: (1) clonación del grupo élite y mutación de los mejores. La clonación de individuos se aplica a aquellos individuos del conjunto B que no hayan superado la edad máxima (Quiroz-Castellanos, 2014).

2.5.8. Método de remplazo

El método de reemplazo utilizado es controlado. Para la cruza, el método, consiste en introducir a la población a los n_c hijos de modo que $n_c/2$ reemplacen a los individuos en el conjunto de padres aleatorios R y los otros $n_c/2$ reemplacen, primero a los individuos con aptitud repetida, para después, si aún quedan hijos sin reinsertar y no hay soluciones con aptitud repetida, se agregan sustituyendo a las soluciones de peor aptitud (Quiroz-

Castellanos *et al.*, 2015).

Con respecto a la mutación, al aplicar este operador algunas de las mejores soluciones son clonadas. Los clones son ingresadas a la población de dos maneras: primero, sustituyendo a las soluciones con aptitud repetida, después si aún quedan hijos sin reinsertar y no hay soluciones con aptitud repetida, se agregan sustituyendo a las soluciones de peor aptitud (Quiroz-Castellanos *et al.*, 2015).

Capítulo 3

Operadores de cruza del estado del arte

Cuando se trabaja con Algoritmos Genéticos, uno de los operadores de variación más usados es el de cruza debido a que con él se puede combinar el material genético con las mejores características de dos soluciones. Pero para el buen desempeño de los algoritmos es necesario un operador planeado correctamente, es decir, que permita hacer una buena exploración del espacio de búsqueda y evite caer el óptimos locales. El diseño del operador dependerá en principio del tipo de representación usada y las características del problema. Como este trabajo gira en torno al 1D-BPP cuando se utiliza como estrategia de solución el algoritmo genético de agrupamiento GGA-CGT, se tiene bien definido el tipo de representación usada: la representación de grupos. A partir de lo mencionado anteriormente, se propone un diseño experimental paso a paso para construir un nuevo operador de cruza. Antes se presenta una descripción de los mejores operadores de cruza dentro del estado del arte para la representación de grupos e identificando sus características a partir de una serie de experimentos propuestos que son descritos más adelante en esta sección; esto servirá como un punto de partida en la creación del operador de cruza.

3.1. Operadores de cruza dentro del estado del arte

A continuación, se describen los cuatro operadores dentro del estado del arte para la representación basada en grupos: Uniform Crossover, Exon Shuffling Crossover, Greedy Partition Crossover y Gene Level Crossover.

Los cuatro operadores fueron implementados en el GGA-CGT y el algoritmo fue ejecutado con la configuración propuesta por Quiroz-Castellanos *et al.* (2015), en todos los operadores se utilizó la heurística FFD para reinsertar los objetos libres.

3.1.1. Uniform Crossover (UX)

El operador Uniform Crossover (UX) es un operador simple, que combina el material genético de dos soluciones padres P_1 y P_2 para generar dos soluciones hijos C_1 y C_2. A continuación se describe el funcionamiento del operador: considere los genes de los padres en paralelo, después, para cada par de genes i, genere un número aleatorio r en el intervalo (0,1) con una distribución uniforme. Si r tiene un valor menor o igual a 0.5 indica que los hijos C_1 y C_2 reciben los genes del grupo i del padre P_1 y P_2 respectivamente, en otro caso, el hijo C_1 recibe los genes del grupo i por parte del padre P_2 y C_2 por parte del padre P_1. Se puede obtener soluciones inconsistentes, específicamente, objetos que se encuentran almacenados en más de un contenedor, en ese caso, en este trabajo, se elimina completamente el contenedor y los objetos que aún no forman parte de ningún grupo quedan libres, para después ser reinsertados utilizando la heurística FFD (descrita en la sección 2.5.4). Este operador no había sido utilizado anteriormente para resolver el 1D-BPP, pero si para el Problema de Mochila Múltiple (Fukunaga, 2008). Se muestra un ejemplo ilustrativo en la Figura 3.2.

Considere una instancia de 1D-BPP en donde la capacidad de los contenedores es igual a 1000, con 18 objetos N = {1,...,18} en donde los pesos son:

objeto	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
peso	366	268	366	430	263	307	414	287	299	495	251	254	474	252	274	370	269	361

Y suponga que tiene las siguientes soluciones padre

padre 1 : objetos	1,3	9,15,18	2,4,5	7,10	6,8,11	12,14,17	13,16
llenado	732	934	961	909	845	775	844

padre 2: objetos	1,17,18	10,13	4,7	2,3,5	6,9,11	8,12,14	15,16
llenado	996	969	844	897	857	793	644

Figura 3.1: Instancia de prueba para los ejemplos de los operadores de cruza para un problema de 1D-BPP: UX (Uniform Crossover), ESX (Exon shuffling Crossover), GPX (Greedy partition Crossover), y GLX (Gene-level Crossover)

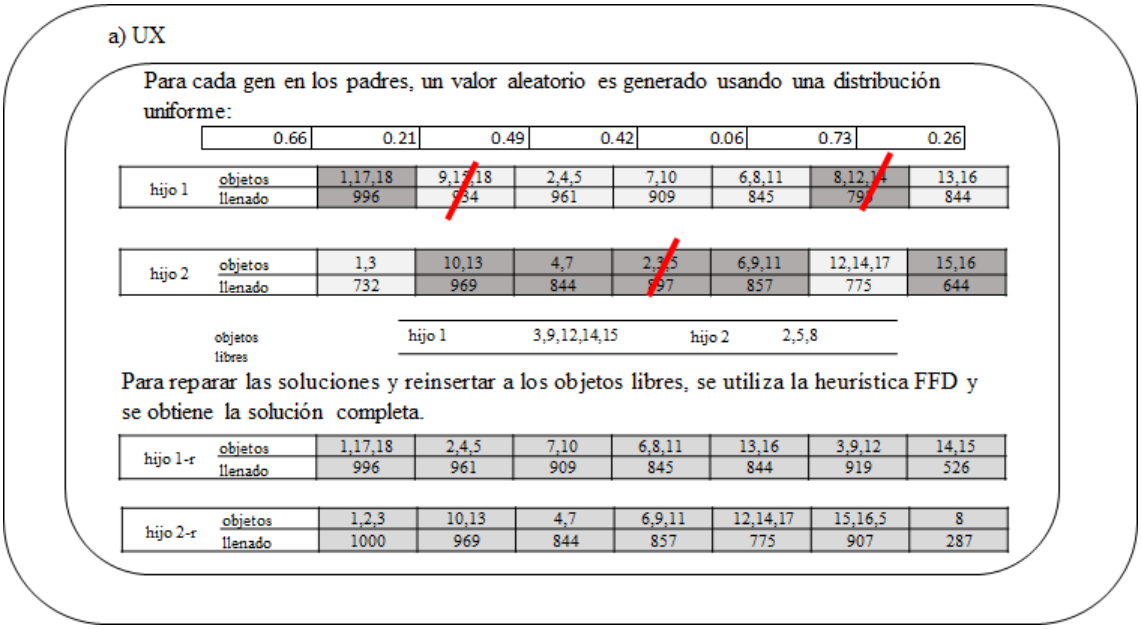


Figura 3.2: Un ejemplo de cada operador de cruza dentro del estado del arte para un problema de 1D-BPP: UX (Uniform Crossover).

3.1.2. Exon Shuffling Crossover (ESX)

El segundo operador dentro del estado del arte es el Exon Shuffling Crossover (ESX), este operador a partir de dos soluciones padre P_1 y P_2 genera una única solución hijo C. ESX trabaja de la siguiente manera: los padres son mezclados y ordenados de manera descendente con respecto al llenado de sus genes (contenedores), después se comienzan a heredar los genes, si alguno de los objetos que se encuentre en el gen i a heredar ya ha sido heredado en otro gen, se omite el gen i completamente y se liberan los objetos. Una vez que se ha terminado de construir la solución c, se reinsertan los objetos libres mediante la heurística FFD. Este problema ha sido utilizado para resolver el 1D-BPP en distintas ocasiones (Dokeroglu & Cosar, 2014; Ozcan *et al.*, 2016b; Wilcox *et al.*, 2011). Se muestra un ejemplo ilustrativo en la Figura 3.3.

3.1.3. Greedy Partition Crossver (GPX)

El tercer operador dentro del estado del arte, es el operador Greedy Partition Crossover (GPX). Este operador, a partir de dos soluciones padre P_1 y P_2, genera dos soluciones hijo C_1 y C_2 usando una heurística greedy. El operador funciona como a continuación se

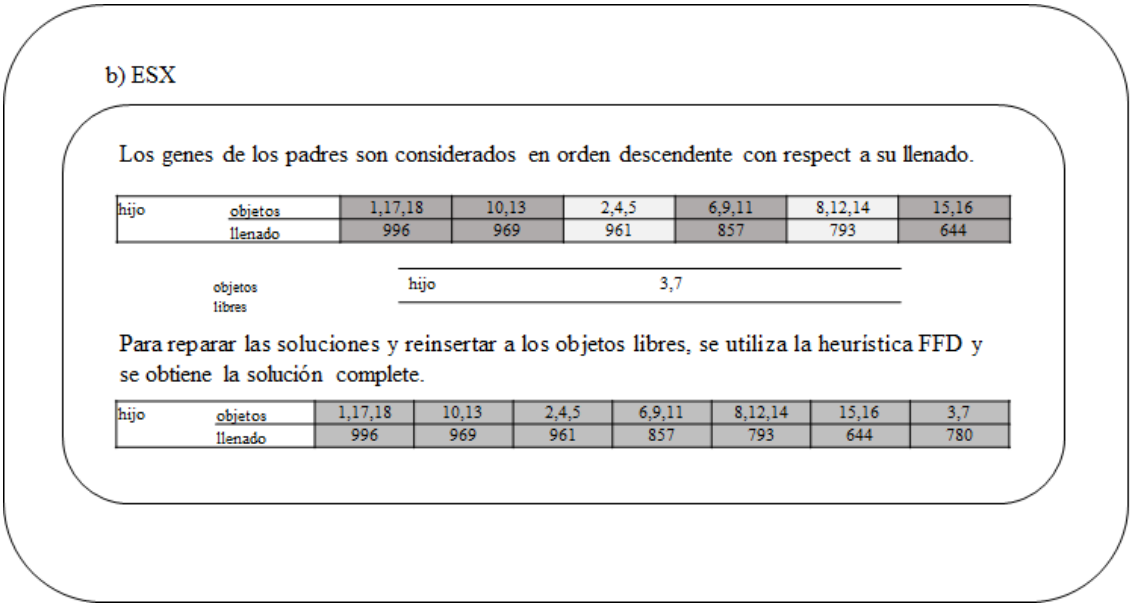


Figura 3.3: Un ejemplo de cada operador de cruza dentro del estado del arte para un problema de 1D-BPP: ESX (Exon shuffling Crossover).

describe: primero se deben ordenar los padres P_1 y P_2 de manera descendente con respecto al llenado de sus genes. Después se genera un par de vectores con tamaño igual al del padre con más genes, el cual estará constituido de valores en el intervalo (0,1) mediante una distribución uniforme. Se asigna un vector de probabilidades a cada hijo. Para el hijo C_1 si su correspondiente vector en la posición i tiene un valor menor o igual a 0.5 indica que el hijo recibe el gen i del padre P_1, en otro caso, del padre P_2. Se realiza el mismo procedimiento para el hijo C_2. Las soluciones pueden ser inconsistentes. En la versión original del operador, si algún gen tiene un objeto que ya está en la solución, se elimina únicamente el objeto del gen, siendo liberado, para después heredar el gen al hijo. Para los objetivos de este trabajo, se utiliza una versión del operador en la que no se elimina únicamente el objeto, sino todo el gen, liberando sus objetos. Después, los objetos libres son reinsertados usando una heurística FF. Este operador ha sido utilizado para resolver 1D-BPP en una ocasión (Singh & Gupta, 2007). Se muestra un ejemplo ilustrativo en la Figura 3.4.

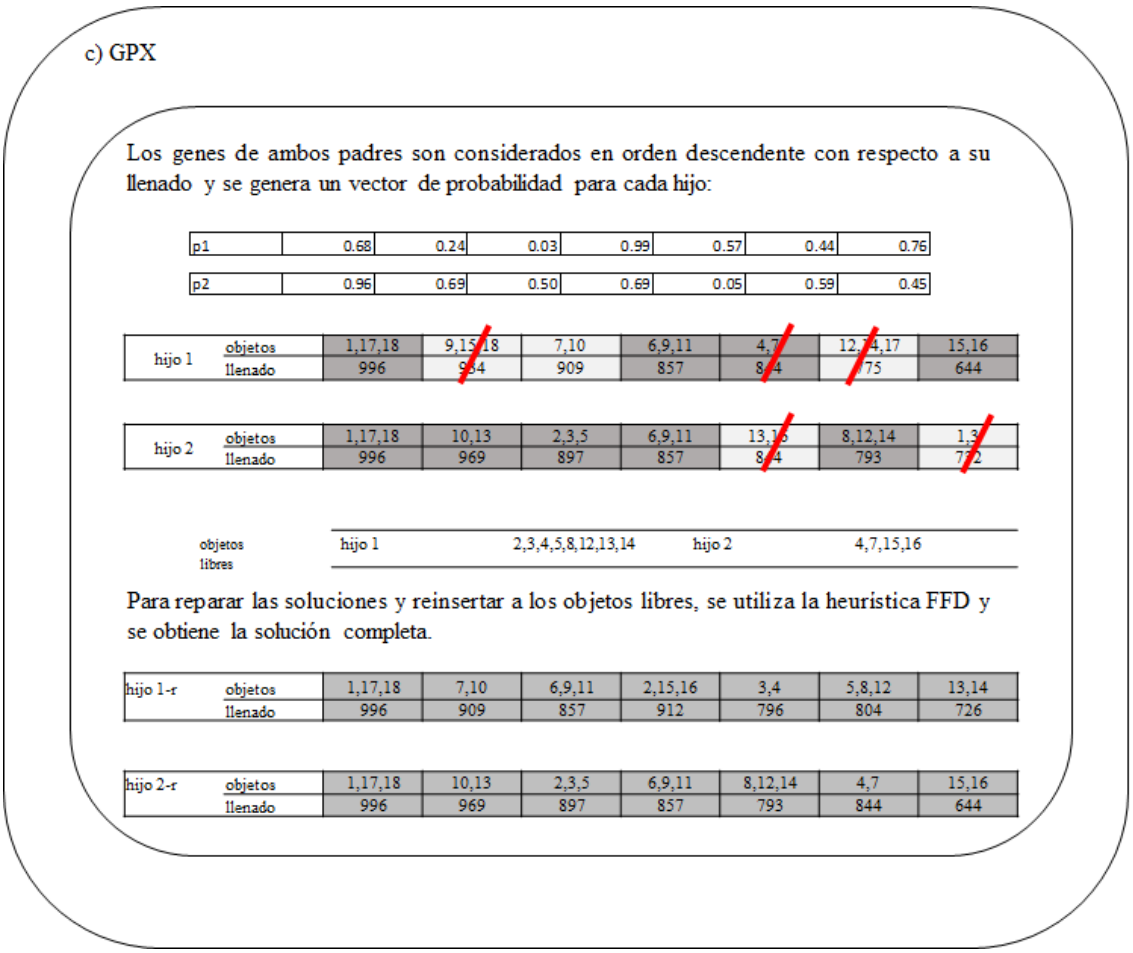


Figura 3.4: Un ejemplo de cada operador de cruza dentro del estado del arte para un problema de 1D-BPP: GPX (Greedy partition Crossover).

3.1.4. Gene Level Crossover (GLX)

Este operador fue propuesto por Quiroz-Castellanos *et al.* (2015) y es el operador predeterminado en el GGA-CGT. A partir de dos soluciones padre P_1 y P_2 genera dos soluciones hijos C_1 y C_2. Primero se ordenan ambos padres de manera descendente, con respecto al llenado de cada gen (contenedor). Después los genes de ambos padres se comparan en paralelo, uno a uno, priorizando la transmisión del gen más lleno. Si ambos genes están igual de llenos, para el hijo C_1 se le da prioridad al padre P_1 y para el hijo C_2 al padre P_2. Si se da el caso que alguno de los padres tiene un mayor número de genes, éstos se heredan directamente a ambos hijos. Los hijos pueden ser inconsistentes, por lo que si algún gen tiene un objeto que ya está en la solución, se elimina y se liberan los objetos.

Después, los objetos libres son reinsertados usando una heurística FFD. Este operador ha sido utilizado para resolver el problema del 1D-BPP en diversas ocasiones (Kucukyilmaz & Kiziloz, 2018; Rivera *et al.*, 2020; Tan *et al.*, 2020). El operador GLX ha mostrados un buen desempeño con respecto a los otros operadores, sin embargo, su impacto en el desempeño del GGA-CGT no es significativo. Se muestra un ejemplo ilustrativo en la Figura 3.5.

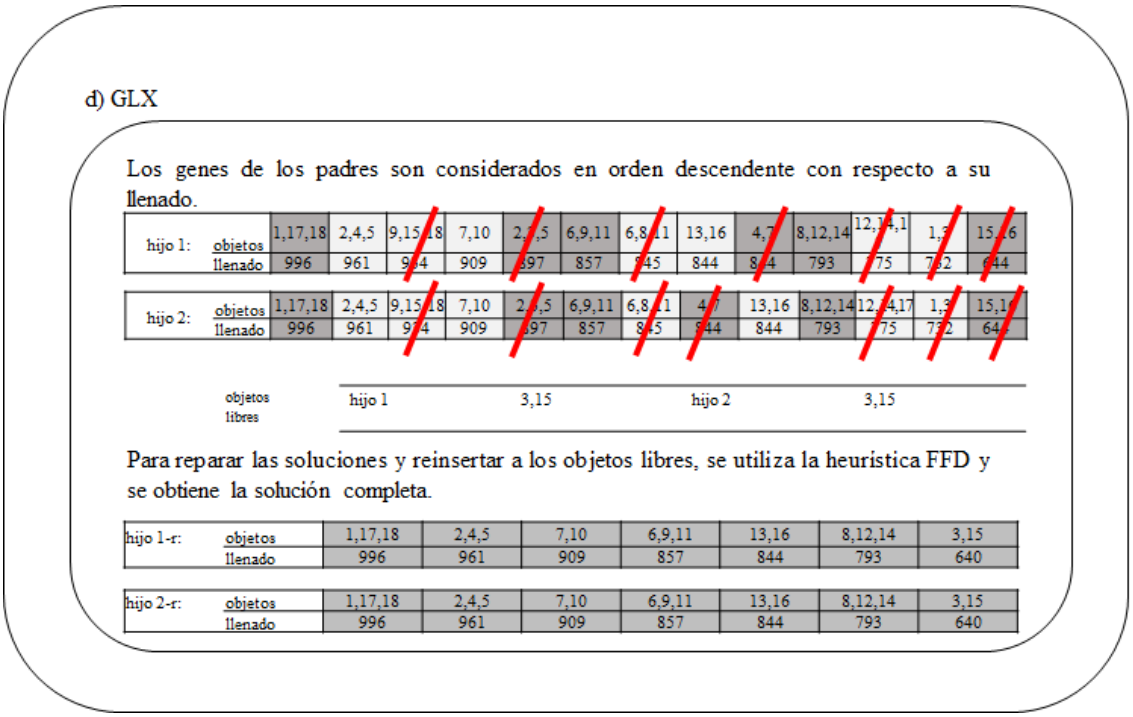


Figura 3.5: Un ejemplo de cada operador de cruza dentro del estado del arte para un problema de 1D-BPP: GLX (Gene-Level Crossover).

3.2. Comparación del desempeño con el banco de prueba clásico

Como se desea medir el desempeño del algoritmo con los distintos operadores para poder realizar una comparación, la primera fase consistió en realizar ejecuciones del algoritmo GGA-CGT con los operadores UX, ESX, GPX, GLX utilizando porcentajes de cruza que van del 0 % al 90 % incrementando de 10 en 10. Las cuatro versiones del GGA-CGT

fueron ejecutadas bajo las mismas condiciones, con los parámetros definidos por Quiroz-Castellanos *et al.* (2015) para todos los parámetros del algoritmo, pero revisando diferentes porcentajes de cruza, con el objetivo de observar el impacto de cada variante. Se muestran los resultados a continuación.

3.2.1. Resultados experimentales

Para el operador predeterminado en el algoritmo GGA-CGT, GLX, se realizaron las primeras ejecuciones, con el objetivo de medir el desempeño del algoritmo al modificar los porcentajes de cruza. Los resultados se muestran en la Tabla 3.1. Para cada clase de pruebas, la Tabla muestra primero el número de casos de prueba, seguido del número de soluciones óptimas encontradas con cada porcentaje de cruza. Como se observa, a pesar de alcanzar un buen desempeño, siendo 1602 los óptimos encontrados, el operador parece no tener un impacto en el algoritmo, ya que cuando no se realiza una cruza, es decir, igual al 0 %, se obtienen 1581 óptimos. En la Figura 3.6 se puede observar cómo varían los óptimos encontrados al cambiar los porcentajes de cruza, observando que a mayor porcentaje, disminuye el número de óptimos encontrados. Por otra parte, en la Figura 3.7 se muestra cómo van cambiando el número de generaciones iteradas por el GGA-CGT, en la mejor configuración que es un 20 % de cruza, el número promedio de generaciones es el menor.

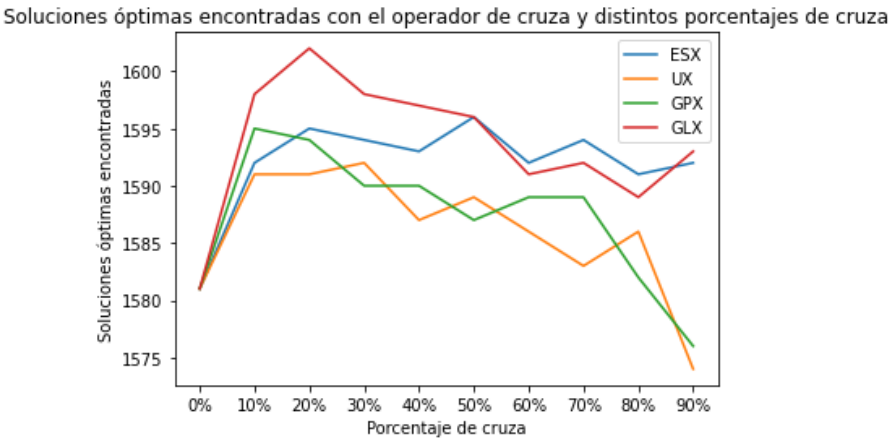


Figura 3.6: Resultados de óptimos encontrados por porcentajes con los operadores: ESX, UX, GPX y GLX.

Tabla 3.1: Resultados obtenidos por el algoritmo GGA-CGT con el operador Gene-level crossover (GLX) para los diferentes porcentajes de cruza.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	719	720	719	718	718	718	718	718	719
Dataset 2	480	480	480	480	480	480	480	480	480	480	480
Dataset 3	10	9	9	10	9	9	9	9	9	8	9
triplets	80	80	80	80	79	79	78	76	77	74	78
uniform	80	80	80	80	80	79	80	79	79	79	80
hard28	28	9	14	16	15	16	15	13	14	14	12
was 1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	16	16	16	16	16	16	15	16	15
Total	1615	1581	1598	**1602**	1598	1597	1596	1591	1592	1589	1593

El segundo operador en ser ejecutado fue el UX. Los resultados obtenidos son mostrados en la Tabla 3.2. Como se observa, el mayor número de óptimos encontrados es con un 30 % de cruza, siendo un total de 1592 casos de prueba de los 1615. Aquí se puede observar cómo con un mayor porcentaje de cruza, el algoritmo se ve perjudicado encontrando menos óptimos que cuando no hay cruza.

Tabla 3.2: Resultados obtenidos por el algoritmo GGA-CGT con el operador de cruza Uniform crossover (UX) para los distintos porcentajes de cruza.

Clase	Casos	Porcentaje de Cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	718	719	719	718	718	718	718	719	718
Dataset 2	480	480	480	480	480	480	480	480	479	480	479
Dataset 3	10	9	9	10	10	10	10	10	9	10	10
triplets	80	80	80	80	80	79	80	78	78	78	69
uniform	80	80	80	80	80	79	80	80	80	79	80
hard28	28	9	11	9	10	8	8	7	6	7	5
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	13	13	13	13	13	13	13	13	13
Total	1615	1581	1591	1591	**1592**	1587	1589	1586	1583	1586	1574

Los resultados del operador ESX se muestran en la Tabla 3.3. Este operador consigue su mejor configuración con un 50 % de cruza, resolviendo un total de 1596 óptimos del

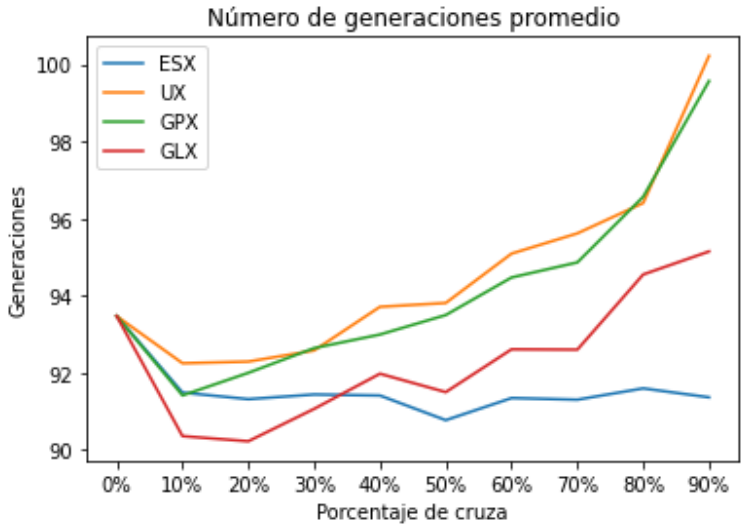


Figura 3.7: Generaciones promedio iteradas por el GGA-CGT para las ejecuciones con los distintos porcentajes de cruza con los operadores: ESX, UX, GPX y GLX.

total de 1615. Para ESX los óptimos encontrados varían solamente entre 1591-1596 y en cualquier porcentaje mayor a cero, se logra observar una mejora con respecto a los óptimos encontrados.

Tabla 3.3: Resultados obtenidos por el algoritmo GGA-CGT con el operador Exon Shuffling crossover (ESX) para los distintos porcentajes de cruza.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	719	719	719	718	718	719	719	718	720
Dataset 2	480	480	480	480	480	480	480	480	480	480	480
Dataset 3	10	9	9	10	9	9	10	9	10	9	9
triplets	80	80	80	80	80	80	80	80	80	80	80
uniform	80	80	80	80	80	80	80	80	80	80	80
hard28	28	9	9	10	10	10	12	9	10	9	8
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	15	16	16	16	16	15	15	15	15
Total	1615	1581	1592	1595	1594	1593	**1596**	1592	1594	1591	1592

Para el operador GPX los resultados se muestran en la Tabla 3.4. Este operador logra 1595 óptimos en su mejor configuración con un 10 % de cruza. Al igual que el operador UX, este operador empeora el desempeño del algoritmo cuando aumentamos el porcentaje

de cruza un 90 %.

Tabla 3.4: Resultados obtenidos por el algoritmo GGA-CGT con el operador Greedy Partition crossover (GPX) para los diferentes porcentajes de cruza.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	720	719	718	719	718	719	719	718	718
Dataset 2	480	480	480	480	480	480	480	480	480	480	478
Dataset 3	10	9	9	10	10	10	10	10	10	10	9
triplets	80	80	80	79	80	79	80	79	80	75	73
uniform	80	80	80	80	80	80	80	79	80	80	79
hard28	28	9	11	12	8	9	6	8	6	6	6
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	15	14	14	13	13	14	14	13	13
Total	1615	1581	**1595**	1594	1590	1590	1587	1589	1589	1582	1576

De los experimentos realizados con los operadores de cruza GLX, UX, ESX y GPX, se puede concluir que los operadores ESX y GLX son los que tienen un mayor impacto en el desempeño del algoritmo genético con respecto a los óptimos encontrados. Aunque la diferencia entre los dos operadores mencionados anteriormente y los demás operadores es de pocos óptimos más, se puede concluir que se obtiene una mejora, esto debido a la complejidad de los casos de prueba. Por otro lado, el peor operador es el UX, resolviendo únicamente un máximo de 1592 casos de prueba óptimamente. El operador UX es el único operador que no ordena los genes de los padres con respecto al llenado de lo mismos, lo que podría indicar que el ordenamiento es una característica que ayuda al desempeño del algoritmo.

El algoritmo GGA-CGT en conjunto con los operadores de cruza ESX y GLX obtiene un mejor desempeño con respecto al número de óptimos encontrados. Estos operadores no generan el mismo número de hijos, mientras que el operador ESX genera un solo hijo, el operador GLX genera dos hijos. Además, en el GGA-CGT el operador predeterminado, GLX, como se mencionó anteriormente, genera dos hijos y utiliza dos métodos de reemplazo (sección 2.5.8). Debido a lo mencionado anteriormente, se realizó un experimento

para probar la diferencia en el desempeño del algoritmo GGA-CGT cuando se genera un hijo o dos hijos en el operador de cruza y cuando se realiza la reinserción de los hijos con ambos métodos de reemplazo, es decir, cuando se reemplazaban las soluciones dentro del conjunto de los padres aleatorios o cuando se reemplazaban al conjunto de padres con peor aptitud y aptitud repetida.

Dado que el operador ESX genera solamente un hijo el experimento se pudo realizar sin efectuar cambios en su procedimiento. Con respecto al operador GLX, se realizó una modificación para que únicamente generara un hijo obteniendo dos versiones, en donde cada una utiliza un método de remplazo distinto. Se le llama ESX-1 y GLX-V1 a la versión de los operadores donde el hijo producto de la cruza es reinsertado a la población sustituyendo a un padre del conjunto aleatorios. Se le llama ESX-2 y GLX-V2 a la versión de los operadores donde el hijo producto de la cruza es reinsertado sustituyendo a una solución con aptitud repetida o a una de las peores soluciones de la población.

Este experimento fue análogo al realizado con los mejores operadores del estado del arte: se realizó una ejecución del algoritmo con los operadores anteriormente mencionados y con porcentajes de cruza de 0 % a 90 % incrementando en 10 para cada ejecución. Los resultados del experimento para el operador ESX-1 son los mismos que el ESX ya que esa reinserción se utilizó en la primera experimentación, los resultados son presentados en la Tabla 3.3. Los resultados restantes son mostrados a continuación.

Los resultados obtenidos con el operador ESX-2 se muestran en la Tabla 3.5. Se puede ver que en esta versión del operador, se logra un total de 1593 óptimos de los 1615 casos de prueba, este resultado es obtenido con una configuración de un 30 % y un 40 % de cruza. Sin embargo, la versión ESX-1 logra 1596 óptimos, siendo 3 casos de prueba más, lo que estaría mostrando un mejor desempeño del algoritmo con ese operador.

En la Tabla 3.6 se presentan los resultados obtenidos de la ejecución del algoritmo GGA-CGT con el operador GLX-V1. En este caso, el algoritmo logra encontrar 1599 óptimos en su mejor configuración con un 20 % de cruza. Este operador al igual que el GLX original, encuentran su mejor configuración con un 20 % de cruza.

Tabla 3.5: Resultados obtenidos por el algoritmo GGA-CGT con el operador ESX-2 y con diferentes porcentajes de cruza.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	718	719	719	718	718	718	718	718	718
Dataset 2	480	480	480	480	480	480	480	480	480	480	480
Dataset 3	10	9	9	9	10	10	9	10	9	9	10
triplets	80	80	80	80	80	80	79	79	79	80	79
uniform	80	80	80	80	80	80	80	80	80	80	80
hard28	28	9	8	9	9	10	9	10	10	10	9
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	14	15	15	15	16	15	15	15	15
Total	1615	1581	1589	1592	**1593**	**1593**	1591	1592	1591	1592	1591

Tabla 3.6: Resultados obtenidos por el algoritmo GGA-CGT con el operador GLX-V1 y con diferentes porcentajes de cruza.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	719	720	719	720	719	718	718	718	719
Dataset 2	480	480	480	480	480	480	480	480	480	480	480
Dataset 3	10	9	10	10	9	9	9	10	9	9	9
triplets	80	80	79	79	79	78	79	76	78	75	79
uniform	80	80	80	80	80	80	80	80	79	79	79
hard28	28	9	13	14	14	15	13	14	15	15	11
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	16	16	16	16	16	15	16	15	16
Total	1615	1581	1597	**1599**	1597	1598	1596	1593	1595	1591	1593

El último operador de estos experimentos es el GLX-V2, los resultados se presentan en la Tabla 3.7. Para este caso, se logra la mejor configuración con tres porcentajes de cruza: 40 %, 60 % y 80 % y un total de 1598 óptimos. Aunque tiene un buen desempeño, el algoritmo con el operador GLX-V1, logra obtener un óptimo más, y se observa que está teniendo un mejor desempeño.

Tabla 3.7: Resultados obtenidos por el algoritmo GGA-CGT con el operador GLX-V2 y con diferentes porcentajes de cruza.

ClasE	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	719	719	719	718	719	719	719	720	719
Dataset 2	480	480	480	480	480	480	480	480	480	480	480
Dataset 3	10	9	10	9	9	10	10	10	9	10	9
triplets	80	80	80	80	80	80	80	80	80	80	79
uniform	80	80	80	80	80	80	80	80	80	80	80
hard28	28	9	12	13	13	14	12	13	11	12	13
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	15	16	16	16	16	16	16	16	15
Total	1615	1581	1596	1597	1597	**1598**	1597	**1598**	1595	**1598**	1595

3.3. Comparación del desempeño de los mejores operadores con el banco de prueba BPPv_u_c

Se realizó el mismo experimento que el realizado al banco de prueba clásico, también con el objetivo de medir el desempeño del algoritmo GGA-CGT con los operadores ESX, GLX y GLX-V1 ya que mostraron un mejor desempeño en los experimentos anteriores. Se desea hacer comparaciones del desempeño de los operadores en el banco de pruebanuevas BPPv_u_c debido a que este banco de prueba es más difícil que el clásico. Al igual que en los experimentos anteriores, se utilizaron porcentajes de cruza que van del 0 % al 90 % incrementando de 10 en 10. Se muestran los resultados a continuación.

Los resultados obtenidos por algoritmo GGA-CGT con el operador de cruza GLX son presentados en la Tabla 3.8, con un 80 % de cruza, resuelve 1723 casos de prueba óptimamente, un promedio del 61.5 %. Por su parte los resultados del algoritmo GGA-CGT con el operador ESX se muestran en la Tabla 3.9, para este caso, se obtiene su mejor configuración con un 80 % de cruza, logrando resolver óptimamente un total de 1657 casos de prueba de los 2800 del total, siendo un 59.1 % del total del banco de prueba. Finalmente, para el algoritmo GGA-CGT con el operador GLX-V1, los resultados se muestran en la Tabla 3.10, con un 90 % de cruza es el operador que más casos de prueba resuelve

Tabla 3.8: Resultados de las ejecuciones del algoritmo GGA-CGT cuando usa el operador GLX y con los distintos porcentajes de cruza para los casos de prueba del banco de prueba BPPv_u_c.

Operador GLX											
Clase	**Casos**	**Porcentaje de Cruza**									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
BPP.25	700	373	400	415	463	475	475	467	470	467	456
BPP.05	700	300	303	312	323	323	316	317	319	316	311
BPP.75	700	200	210	247	302	323	324	364	360	372	348
BPP1	700	202	374	469	542	541	554	556	563	568	538
Total	2800	1075	1287	1443	1630	1662	1669	1704	1712	**1723**	1653

Tabla 3.9: Resultados de las ejecuciones del algoritmo GGA-CGT con el operador ESX y los distintos porcentajes de cruza para los casos de prueba del banco de prueba BPPv_u_c.

Operador ESX											
Clase	**Casos**	**Porcentaje de Cruza**									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
BPP.25	700	373	399	404	420	441	446	454	459	467	458
BPP.05	700	300	298	300	300	305	306	304	304	309	307
BPP.75	700	200	200	214	241	265	292	333	331	334	331
BPP1	700	202	336	432	498	523	546	540	549	547	544
Total	2800	1075	1233	1350	1459	1534	1590	1631	1643	**1657**	1640

óptimamente, con un total de 1950 óptimos de las 2800 pertenecientes al banco de prueba, siendo un 69.6 % del total. Como se observa en los resultados obtenidos (presentados en las Tablas 3.8 para GLX, 3.9 para ESX y 3.10 para GLX-V1), cuando se utiliza el operador ESX con cualquier configuración de cruza, siempre obtiene un menor número de óptimos, que cuando se utiliza el operador GLX; para el caso cuando se utiliza el operador GLX-V1, siempre se obtiene un mayor número de óptimos que con el operador ESX, mientras que obtiene un mayor número de óptimos con respecto al operador GLX cuando se utiliza un porcentaje de cruza mayor al 40 % Obteniendo un mayor número de óptimos con el operador GLX-V1.

En la Figura 3.8, se muestra una gráfica del comportamiento del algoritmo GGA-CGT con los operadores GLX, ESX y GLX-V1 y los porcentajes de cruza del 0 % al 90 % de

Tabla 3.10: Resultados de las ejecuciones del algoritmo GGA-CGT con el operador GLX-V1 y los distintos porcentajes de cruza para los casos de prueba del banco de prueba BPPv_u_c.

Operador GLX-V1											
Clase	**Casos**	**Porcentaje de Cruza**									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
BPP.25	700	373	359	384	400	420	441	464	480	483	482
BPP.05	700	300	302	311	316	336	337	334	343	341	342
BPP.75	700	200	206	230	271	340	398	429	465	484	503
BPP1	700	202	373	486	541	567	583	595	609	617	623
Total	2800	1075	1240	1411	1528	1663	1759	1822	1897	1925	**1950**

cruza, incrementando de 10 en 10, los cuales han sido discutidos anteriormente. Además, como también se observa, para este banco de prueba, se requiere de un mayor porcentaje de cruza para encontrar un mayor número de óptimos, a diferencia de lo que se observó con el banco de prueba clásico.

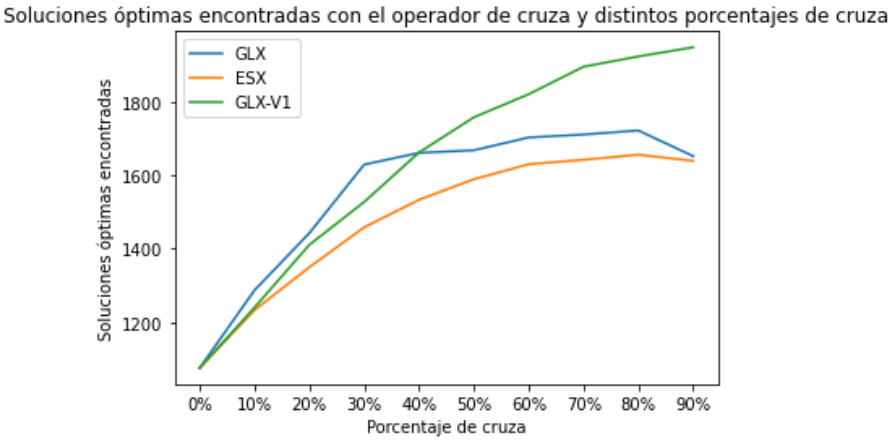


Figura 3.8: Resultados de óptimos encontrados con distintos porcentajes de cruza con los operadores: GLX, ESX y GLX-V1, para el banco de prueba BPPv_u_c.

Los resultados presentados en Carmona-Arroyo *et al.* (2021), muestran que el GGA-CGT con su configuración predeterminada, logra obtener un total de 1549 óptimos de los 2800. El algoritmo GLX-V1 encuentra 1950 óptimos, 401 casos de prueba más resueltos óptimos, lo que indica una mejora significativa cuando se genera solo un hijo.

Tabla 3.11: Principales características de los operadores UX, ESX, GPX, GLX y GLX-V1.

Operador	No. de hijos	Ordenamiento de genes	Probabilidad	Todos los genes
Uniform Crossover	2	×	✓	×
Exon Shuffling Crossover	1	✓	×	✓
Gene Level Crossover	2	✓	×	✓
Greedy Partition Crossover	2	✓	✓	×
Gene Level Crossover V1	1	✓	×	✓

3.4. Características de los operadores de cruza

En la Tabla 3.11 se describen las características principales de cada operador de cruza: (1) el número de hijos que genera, (2) si ordena o no los genes de los padres antes de transmitirlos a los hijos, (3) si utiliza o no alguna probabilidad para decidir a qué hijo heredar cual gen, y finalmente, (4) si cada padre tiene la posibilidad de transmitir todo el material genético a ambos hijos o solo una parte a cada uno de ellos. De los cinco operadores, los operadores GLX, ESX y GLX-V1 muestran los mejores resultados, obteniendo un mayor número de óptimos; teniendo como características en común el ordenar los genes de los padres antes de comenzar la transmisión genética, el transmitir todo el material genético a su descendencia y no utilizar probabilidades para decidir a cuál hijo transmiten los genes. El último par de características anteriores trabajan en conjunto debido a que con el vector se decide que genes van a cada hijo. Por lo que se puede asumir, que para el 1D-BPP, transmitir todo el material genético es una característica que favorece el desempeño del algoritmo GGA-CGT con su respectivo operador de cruza.

De los resultados experimentales anteriores, se concluyó lo siguiente:

- Para el banco de prueba clásico, el algoritmo GGA-CGT con el operador GLX y su configuración original, continúa mostrando los mejores resultados y se observa un mejor desempeño con respecto al número de casos de prueba que resuelve óptimamente. Sin embargo, cuando se aumenta el porcentaje de individuos a cruza, el desempeño se reduce. Algunas de las características más destacables del operador

GLX son: el ordenamiento de los genes de los padres antes de heredar, el método de transmisión de los genes y que hereda todo el material genético al par de hijos que genera.

- Para el banco de prueba BPPv_u_c, los operadores parecen mostrar que tienen un mayor impacto cuando se utiliza un porcentaje de cruza alto. Especialmente, el operador GLX-V1, muestra un gran desempeño con respecto al número óptimos obtenidos, utilizando un porcentaje de cruza muy alto. Este operador a diferencia del operador GLX, únicamente genera un hijo, con respecto al operador ESX, aunque ambos generan un hijo, este operador compara los genes de los padres en paralelo (ordenados previamente) para realizar la transmisión de genes, conservando a los mejores genes en el hijo generado.

- Las características de las clases de pruebas son distintas, por lo que los resultados utilizando parámetros con el mismo valor varían dependiendo de con cual conjunto se esté trabajando. Se observó que, para el banco de prueba clásico, se requiere de un porcentaje de cruza bajo, ya que, al incrementarlo, los óptimos encontrados disminuyen. A diferencia del banco de prueba clásico, el banco de prueba BPPv_u_c obtiene un mayor número de óptimos si el porcentaje de cruza es alto, mientras que cuando se cruza poco, los óptimos obtenidos disminuyen radicalmente.

- El ordenamiento de los genes de los padres con respecto a su llenado en el proceso de cruza, muestra ser una característica que mejora al desempeño del algoritmo GGA-CGT.

Debido a la conclusión de que el ordenamiento de los genes de las soluciones padre antes de la transmisión del material genético muestra ser una característica que beneficia el desempeño del GGA-CGT con el operador de cruza y a que el operador UX es el único que no la utiliza, se decidió realizar un experimento con este operador, dato también que es el operador con el que se obtiene el menor número de óptimos.

3.5. UX-Sorted

Este operador es una versión modificada del operador UX. Se propuso realizar el ordenamiento de los genes de los padres de manera descendente con respecto al llenado de cada gen antes de heredarlos. Una vez que se tiene a los padres con los genes ordenados, el operador UX-Sorted trabaja del mismo modo que el operador UX tradicional. En la Figura 3.9 se muestra un ejemplo ilustrativo para este operador.
Los resultados del experimento con el operador UX-Sorted con el banco de prueba clásico se presentan en la Tabla 3.12. Como se observa, se obtiene un total de 1596 casos de prueba resueltos de manera óptima.

Tabla 3.12: Resultados obtenidos por el algoritmo GGA-CGT con el operador UX-sorted y diferentes porcentajes de cruza para el banco de prueba clásico.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	708	719	720	719	719	719	719	718	718	718
Dataset 2	480	480	480	480	480	480	480	479	480	480	478
Dataset 3	10	9	10	10	10	10	10	10	10	10	9
triplets	80	80	80	80	80	78	79	77	77	76	71
uniform	80	80	80	80	80	80	80	79	79	79	79
hard28	28	9	13	10	8	9	7	7	7	7	6
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	14	13	13	13	13	13	14	14	14
Total	1615	1581	**1596**	1593	1590	1589	1588	1584	1585	1584	1575

Para el caso del banco de prueba BPPv_u_c los resultados se presentan en la Tabla 3.13. Para este banco de prueba los resultados no son favorables ya que se resuelve óptimamente un mayor número de casos de prueba cuando no se realiza cruza, es decir, se tiene un 0 % de cruza. Cuando si utiliza cruza, únicamente se logra resolver 1003 caso de prueba óptimamente de los 2800.

Además de lo anterior, se realizó una comparación con los operadores UX y UX-Sorted, para observar las diferencias entre el mayor número de óptimos encontrados y el menor con respecto al banco de prueba clásico. Para el caso del mayor número de óptimos encontrados con los distintos porcentajes de cruza, estos resultados se presentan en la Tabla

Tabla 3.13: Resultados obtenidos por el algoritmo GGA-CGT con el operador UX-sorted y diferentes porcentajes de cruza para el banco de prueba BPPv_u_c.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
BPP.25	700	373	342	304	300	300	300	300	300	294	278
BPP.05	700	300	243	202	200	200	200	200	200	200	191
BPP.75	700	200	200	201	200	200	200	201	201	201	200
BPP1	700	202	218	222	216	220	217	211	210	200	198
Total	2800	**1075**	1003	929	916	920	917	912	911	895	867

Tabla 3.14: Resultados del mayor número de óptimos encontrados con el algoritmo GGA-CGT y los operadores UX y UX-Sorted.

Clase	Casos	UX	UX-sorted
		Porcentaje de Cruza	
		30 %	10 %
Dataset 1	720	719	719
Dataset 2	480	480	480
Dataset 3	10	10	10
triplets	80	80	80
uniform	80	80	80
hard28	28	10	13
was1	100	100	100
was 2	100	100	100
gau 1	17	13	14
Total	1615	1592	1596

3.14. Con respecto a la configuración de cruza con la que se obtiene el menor número de óptimos, estos son presentados en la Tabla 3.15.

Como conclusión del experimento anterior, se tiene una mejora en el desempeño del algoritmo GGA-CGT con el operador de cruza UX-Sorted con respecto al UX, lo que indica que ordenar a los padres antes de heredar los genes es una característica que estaría beneficiando al operador de cruza y por consecuente al algoritmo.

En el capítulo siguiente se presenta un análisis de cada una de las características más relevantes identificadas en cada operador, para dar paso a la construcción del nuevo operador de cruza, que permita mejorar el desempeño del algoritmo GGA-CGT.

Tabla 3.15: Resultados del menor número de óptimos encontrados con el algoritmo GGA-CGT y los operadores UX y UX-Sorted.

Clase	Casos	UX	UX-sorted
		Porcentaje de cruza	
		90 %	90 %
Dataset 1	720	718	718
Dataset 2	480	479	478
Dataset 3	10	10	9
triplets	80	69	71
uniform	80	80	79
hard28	28	5	6
was1	100	100	100
was 2	100	100	100
gau 1	17	13	14
Total	1615	1574	1575

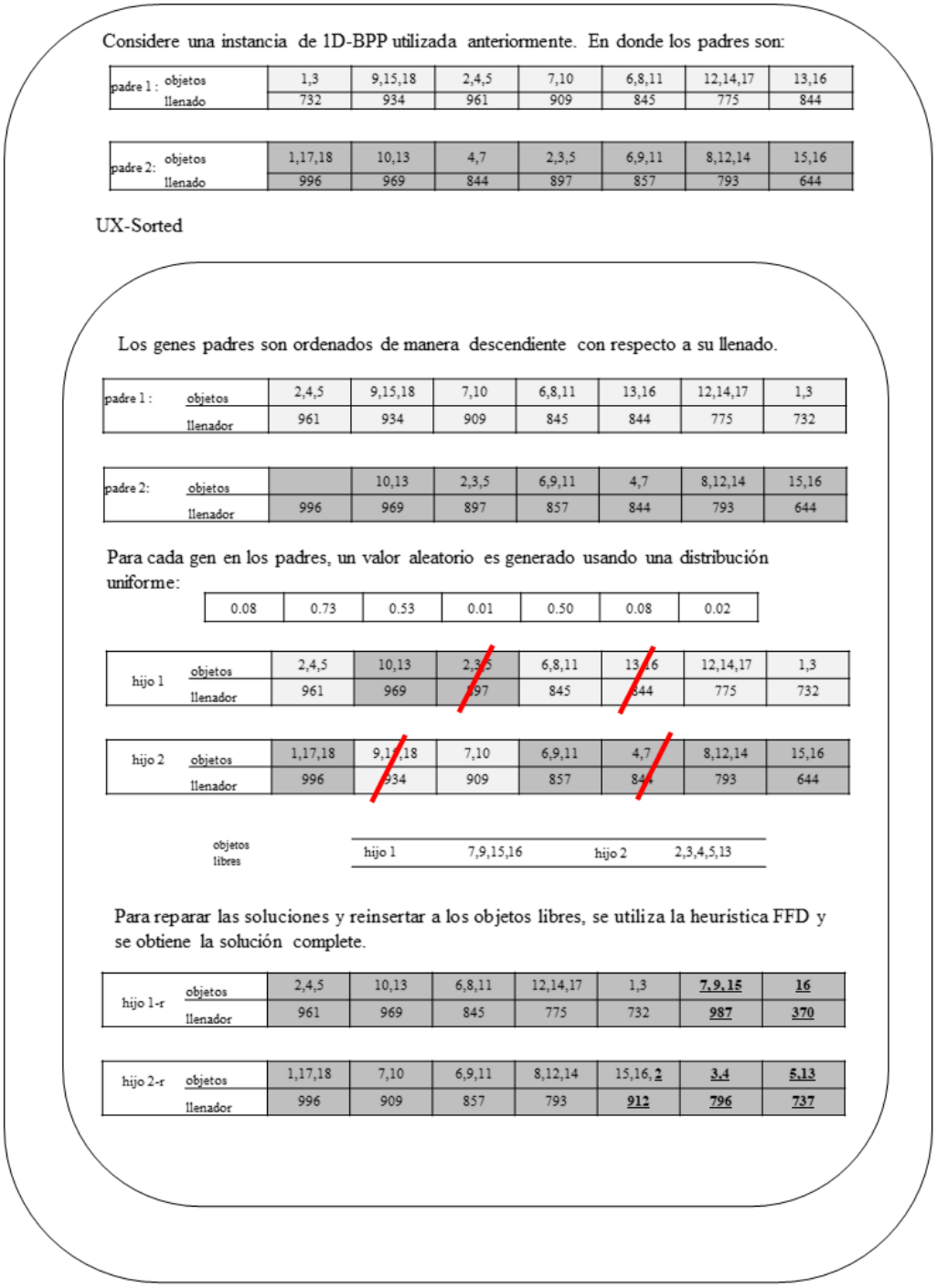


Figura 3.9: Un ejemplo del operador de cruza UX-Sorted para un problema de 1D-BPP.

Capítulo 4

Análisis de estrategias involucradas en la cruza

Como se concluyó en el capítulo anterior, las características que obtienen una mejora al utilizarlas dentro del operador de cruza en el algoritmo GGA-CGT son: el ordenamiento de los genes de los padres antes de transmitirlos a los hijos y el transmitir todo el material genético a los hijos. Es por ello que estas características son tomadas en consideración para comenzar el diseño del nuevo operador de cruza. Para el estudio de las estrategias involucradas en la cruza y el diseño de un nuevo operador se propone trabajar con tres procesos dentro de la cruza, dos de los cuales son características que se observó que muestran beneficios al desempeño del GGA-CGT: orden de los genes de los padres, transmisión de genes a los hijos y la reinserción de los objetos libres una vez que los hijos han sido creados. Con respecto al ordenamiento de los genes se proponen 7 alternativas de ordenamiento, que consideran cuatro aspectos para ordenar los genes: aleatoriedad, llenado de los contenedores, número de objetos en los contenedores y peso de los objetos. Para el caso de la transmisión de genes, se tiene la propuesta de 5 métodos, que consideran tres aspectos para elegir el orden en el que se heredarán un par de genes a un hijo: aleatoriedad, llenado de los contenedores y número de objetos en los contenedores. Finalmente, para el caso de la reinserción de los objetos libres, se utilizaron dos heurísticas bien conocidas en la literatura (Martello, 1990; Quiroz-Castellanos *et al.*, 2015): FFD y RP.

En las siguientes subsecciones se describe el funcionamiento de los métodos de ordenamiento, transmisión y reinserción, así como también se presentan los resultados experimentales obtenidos.

4.1. Ordenamiento de genes

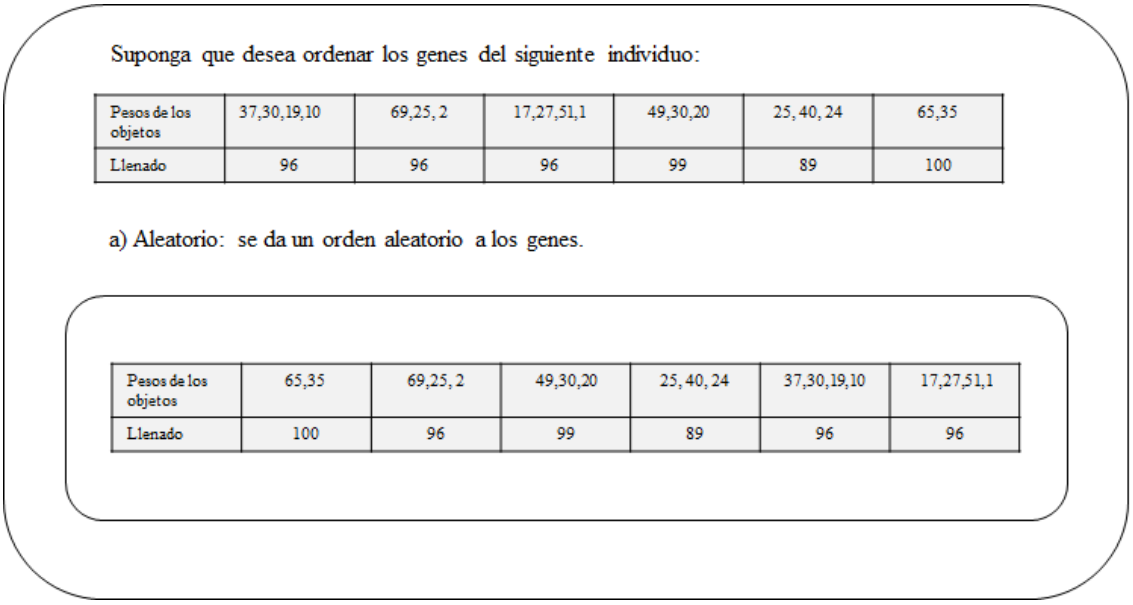
Como se concluyó en la primera fase del estudio experimental, el ordenar a los genes de los padres antes de heredarlos a los hijos, ha mostrado ser una característica que beneficia el impacto del operador de cruza en el desempeño del algoritmo genético. Derivado de lo anterior, primero se propusieron 7 tipos de ordenamiento orientados a la representación de grupos. Estos mismos fueron aplicados a los operadores de cruza: GLX-V1, ESX y UX-Sorted, debido a los buenos resultados de los dos primeros y la nueva versión propuesta para UX. A continuación se describirán las características y funcionamiento de los ordenamientos propuestos.

4.1.1. Aleatorio

Este ordenamiento, como su nombre lo dice, consiste en ordenar los genes de ambos padres de manera aleatoria. En la Figura 4.1 se muestra un ejemplo ilustrativo del ordenamiento Aleatorio.

4.1.2. Items

Para este ordenamiento de genes, se considera el número de objetos dentro de cada gen tomando en cuenta que un contenedor con menos objetos podría tener objetos grandes y utilizar mejor el espacio, los genes de los padres serán ordenados de manera ascendente con respecto al número de objetos. En caso de que un par de genes tengan el mismo número de objetos, se elegirá aleatoriamente cual va primero. En la Figura 4.2 se muestra un ejemplo ilustrativo del ordenamiento Items.



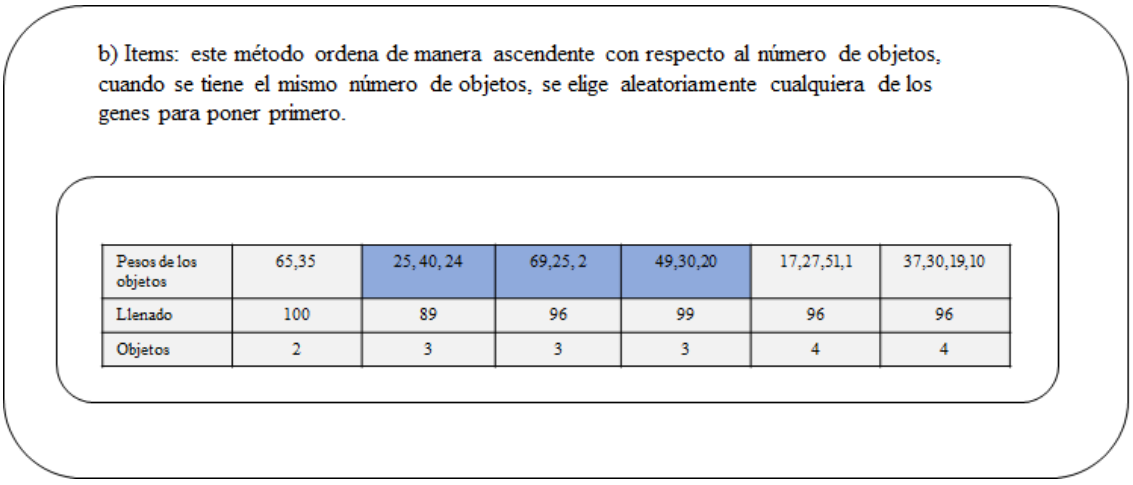
Suponga que desea ordenar los genes del siguiente individuo:

Pesos de los objetos	37,30,19,10	69,25, 2	17,27,51,1	49,30,20	25, 40, 24	65,35
Llenado	96	96	96	99	89	100

a) Aleatorio: se da un orden aleatorio a los genes.

Pesos de los objetos	65,35	69,25, 2	49,30,20	25, 40, 24	37,30,19,10	17,27,51,1
Llenado	100	96	99	89	96	96

Figura 4.1: Ejemplo ilustrativo del método de ordenamiento: Aleatorio.



b) Items: este método ordena de manera ascendente con respecto al número de objetos, cuando se tiene el mismo número de objetos, se elige aleatoriamente cualquiera de los genes para poner primero.

Pesos de los objetos	65,35	25, 40, 24	69,25, 2	49,30,20	17,27,51,1	37,30,19,10
Llenado	100	89	96	99	96	96
Objetos	2	3	3	3	4	4

Figura 4.2: Ejemplo ilustrativo del método de ordenamiento: Items.

4.1.3. Fullness

El ordenamiento Fullness es el propuesto por Quiroz-Castellanos *et al.* (2015), el cual consiste en ordenar los genes de manera descendente con respecto a su llenado. En la Figura 4.3 se muestra un ejemplo ilustrativo del ordenamiento Fullness.

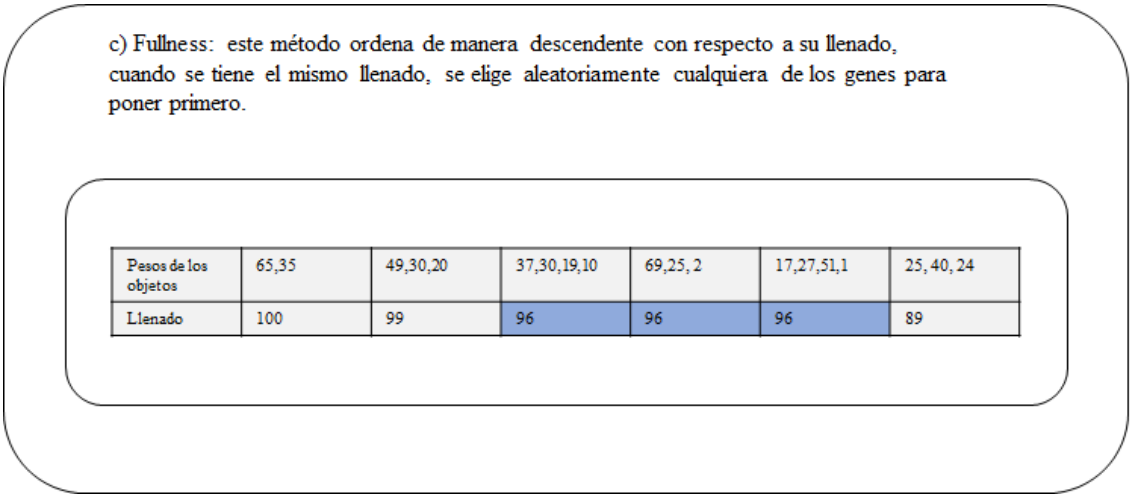


Figura 4.3: Ejemplo ilustrativo del método de ordenamiento: Fullness.

4.1.4. Items-Fullness

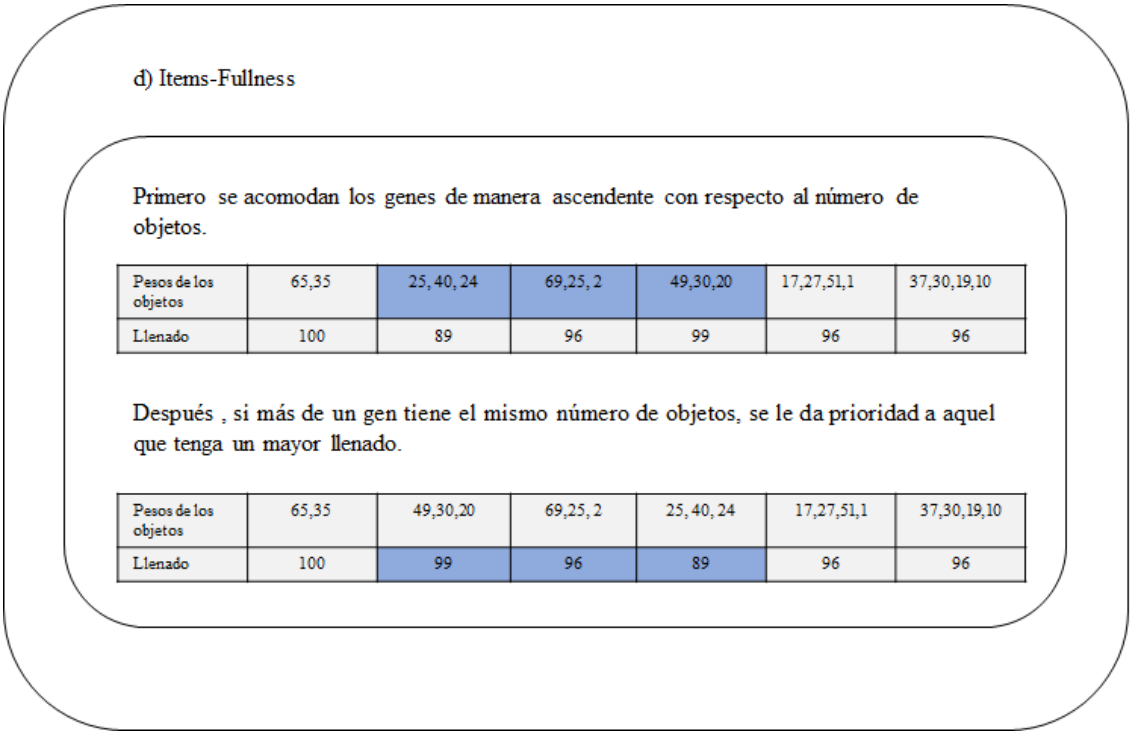
Con el ordenamiento Items-Fullness se combinan los ordenamientos Items y Fullness. Primero, se ordenan los genes de forma ascendente con respecto al número de objetos, si se da el caso que dos genes tienen el mismo número de objetos, se utiliza el criterio Fullness, es decir, se compara que tan llenos están y se le da prioridad al gen más lleno. En la Figura 4.4 se muestra un ejemplo ilustrativo del ordenamiento Items-Fullness.

4.1.5. Fullness-Items

El ordenamiento Fullness-Items, utiliza el siguiente criterio para ordenar. Primero, ordena de manera descendente con respecto al llenado, si se da el caso que dos genes tienen el mismo llenado, entonces recurre al criterio Items, es decir, compara el número de objetos y da prioridad al gen con menos objetos. En la Figura 4.5 e) se muestra un ejemplo ilustrativo del ordenamiento Fullness-Items.

4.1.6. Fullness-Weights-Items

En este ordenamiento se ordena de manera descendente con respecto al llenado, si se tiene un par de genes con el mismo llenado, se le dará prioridad al gen que tenga un objeto con un peso mayor o igual a la mitad de la capacidad c del contenedor. En caso de



d) Items-Fullness

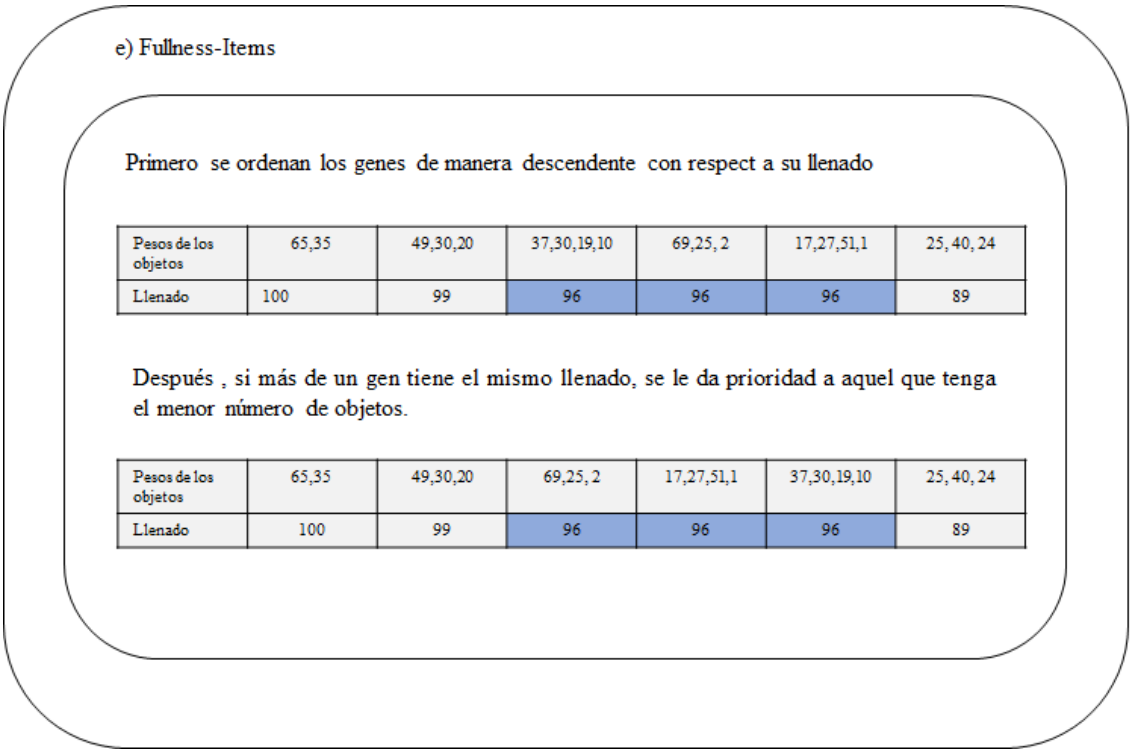
Primero se acomodan los genes de manera ascendente con respecto al número de objetos.

Pesos de los objetos	65,35	25, 40, 24	69,25, 2	49,30,20	17,27,51,1	37,30,19,10
Llenado	100	89	96	99	96	96

Después , si más de un gen tiene el mismo número de objetos, se le da prioridad a aquel que tenga un mayor llenado.

Pesos de los objetos	65,35	49,30,20	69,25, 2	25, 40, 24	17,27,51,1	37,30,19,10
Llenado	100	99	96	89	96	96

Figura 4.4: Ejemplo ilustrativo del método de ordenamiento: Items-Fullness.



e) Fullness-Items

Primero se ordenan los genes de manera descendente con respect a su llenado

Pesos de los objetos	65,35	49,30,20	37,30,19,10	69,25, 2	17,27,51,1	25, 40, 24
Llenado	100	99	96	96	96	89

Después , si más de un gen tiene el mismo llenado, se le da prioridad a aquel que tenga el menor número de objetos.

Pesos de los objetos	65,35	49,30,20	69,25, 2	17,27,51,1	37,30,19,10	25, 40, 24
Llenado	100	99	96	96	96	89

Figura 4.5: Ejemplo ilustrativo del método de ordenamiento: Fullness-Items.

que los dos genes cumplan la condición anterior, se le dará prioridad al gen con el menor número de objetos. Si también se cumple la condición anterior para ambos genes, el gen a acomodar primero se seleccionará de forma aleatoria. En la Figura 4.6 se muestra un ejemplo ilustrativo del ordenamiento Fullness-Weights-Items.

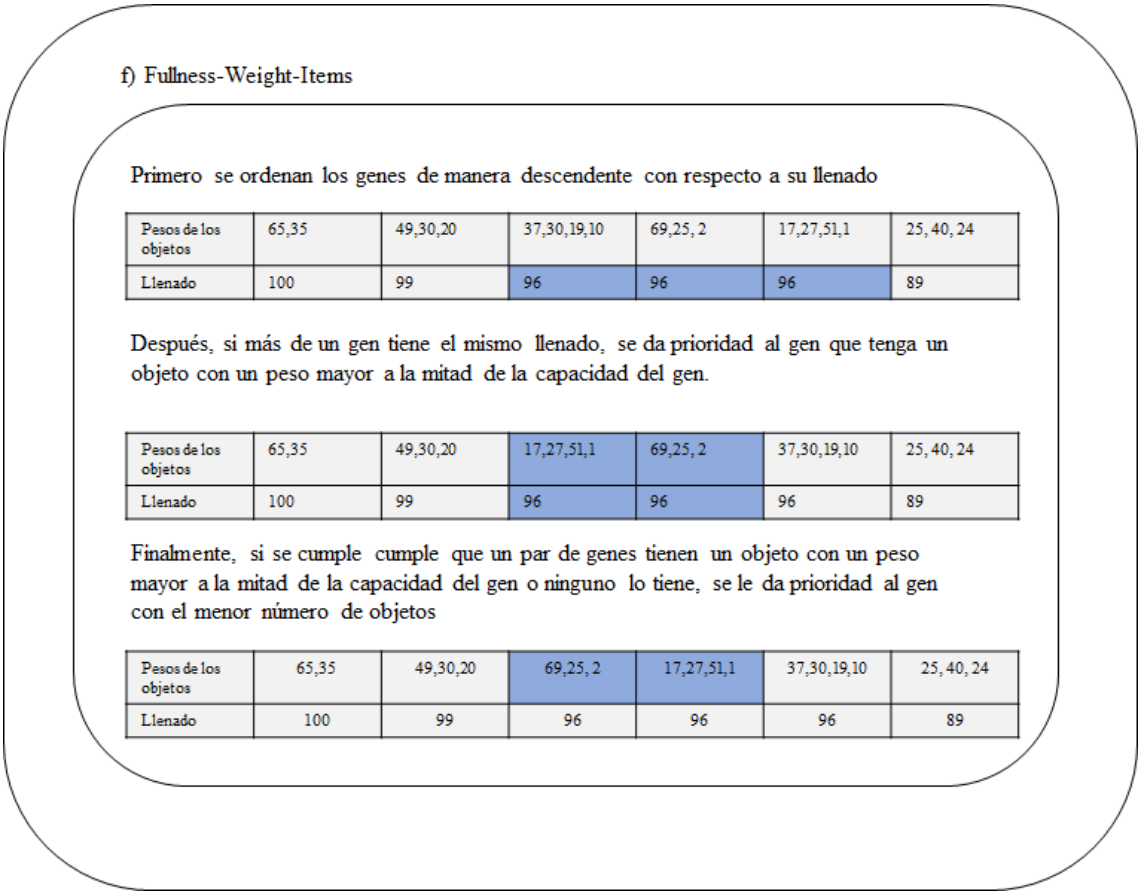
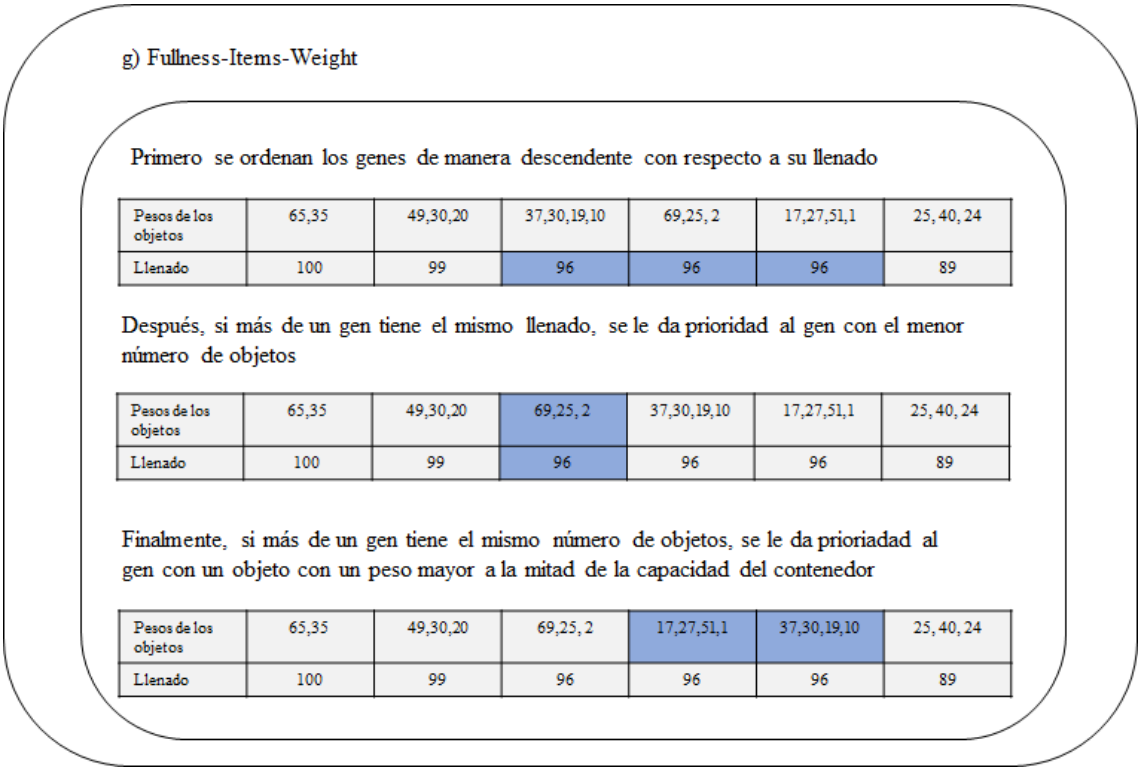


Figura 4.6: Ejemplo ilustrativo del método de ordenamiento: Fullness-Weights-Items.

4.1.7. Fullness-Items-Weights

En este ordenamiento se ordena de manera descendente con respecto al llenado, si se tiene un par de genes con el mismo llenado, se le dará prioridad al gen con el menor número de objetos. En caso de que los dos genes cumplan la condición anterior, se le dará prioridad al gen que tenga un objeto con un peso mayor o igual a la mitad del contenedor. Si aún se sigue cumpliendo lo anterior para ambos genes, el gen a acomodar primero se seleccionará de forma aleatoria. En la Figura 4.7 se muestra un ejemplo ilustrativo del ordenamiento

Fullness-Items-Weights.



g) Fullness-Items-Weight

Primero se ordenan los genes de manera descendente con respecto a su llenado

Pesos de los objetos	65,35	49,30,20	37,30,19,10	69,25, 2	17,27,51,1	25, 40, 24
Llenado	100	99	96	96	96	89

Después, si más de un gen tiene el mismo llenado, se le da prioridad al gen con el menor número de objetos

Pesos de los objetos	65,35	49,30,20	69,25, 2	37,30,19,10	17,27,51,1	25, 40, 24
Llenado	100	99	96	96	96	89

Finalmente, si más de un gen tiene el mismo número de objetos, se le da prioriadad al gen con un objeto con un peso mayor a la mitad de la capacidad del contenedor

Pesos de los objetos	65,35	49,30,20	69,25, 2	17,27,51,1	37,30,19,10	25, 40, 24
Llenado	100	99	96	96	96	89

Figura 4.7: Ejemplo ilustrativo del método de ordenamiento: Fullness-Items-Weights.

,

4.1.8. Resultados del ordenamiento de los genes

En el capítulo 2.4.2 se definió el banco de prueba BPPv_u_c. Se tiene un total de 2800 casos de prueba, divididos en cuatro clases. Cada clase de prueba tiene un total de 700 casos y estos están divididos en 7 conjuntos, en donde cada uno de estos tiene una capacidad máxima en los contenedores de sus casos de prueba. En la Tabla 4.1 se presenta una descripción del banco de prueba BPPv_u_c.

Para esta serie de experimentos, se trabajó primero con 400 casos pertenecientes al banco de prueba BPPv_u_c, para después de seleccionar el mejor método de ordenamiento de genes, medir el desempeño que se obtiene con este al evaluar con los dos bancos de prueba completos.

Se decidió trabajar primero con el banco BPPv_u_c ya que se tiene una mayor área de

Tabla 4.1: Descripción del banco de prueba BPPv_u_c.

Clase	**Casos**	**Pesos**	**Capacidades**	**Número objetos**	**Óptimo**
BBP.25	700	(0,0.25c]	$10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8$	[110,154]	15
BPP.5	700	(0,0.5c]		[124,167]	30
BPP.75	700	(0,0.75c]		[132,165]	45
BPP1	700	(0,c]		[148,188]	60

Tabla 4.2: casos de prueba resueltos óptimamente utilizando el operador de cruza UX-Sorted con los ordenamientos: Aleatorio, Items, Fullness, Items-Fullness, Fullness-Items, Fullness-Weights-Items y Fullness-Items-Weights.

Clase	Casos	Aleatorio	Items	Fullness	Items - Fullness	Fullness - Items	Fullness -Weight- Items	Fullness -Items- Weight
BPP.25-1000000	100	0	0	0	0	0	0	0
BPP.5-100000	100	0	0	0	0	0	0	0
BPP.75-100000	100	0	0	0	0	0	0	0
BPP-100000	100	0	0	0	0	0	1	0
total	400	0	0	0	0	0	1	0

oportunidad y complejidad. Los 400 casos de prueba fueron seleccionadas estratégicamente de las conjuntos más difíciles, como se muestra a continuación: clase BPP.25, conjunto: 10^6, clase BPP.5, conjunto: 10^5, clase BPP.75, conjunto: 10^5 y clase BPP1, conjunto 10^5. Las siete estrategias de ordenamiento fueron integradas en los operadores GLX-V1, ESX y UX-Sorted, que fueron implementados en el GGA-CGT. El algoritmo fue ejecutado con la configuración propuesta por Quiroz-Castellanos *et al.* (2015), en donde el único parámetro modificado con respecto a la configuración original es el porcentaje de cruza, que se estableció en 90 %, debido a los resultados obtenidos en el capítulo anterior en el conjunto BPPv_u_c. En todos los operadores se utilizó la heurística FFD para reinsertar los objetos libres.

Los resultados del algoritmo con el operador de cruza UX-Sorted se presentan en la Tabla 4.2. Como se observa, para este operador solo es posible encontrar un óptimo con el ordenamiento Fullness-Weights-Items, lo que evidencia que las estrategias incluidas en este operador para la elección de los genes que serán heredados a los hijos no son efectivas. Lo anterior se debe en mayor medida a que cuando se hace la transmisión genética se pierde

Tabla 4.3: casos de prueba resueltos óptimamente utilizando el operador de cruza ESX con los ordenamientos: Aleatorio, Items, Fullness, Items-Fullness, Fullness-Items, Fullness-Weights-Items y Fullness-Items-Weights.

Clase	Casos	Aleatorio	Items	Fullness	Items - Fullness	Fullness - Items	Fullness -Weight- Items	Fullness -Items- Weight
BPP.25-1000000	100	0	0	67	0	72	72	72
BPP.5-100000	100	0	0	8	0	7	7	7
BPP.75-100000	100	0	0	2	0	3	5	5
BPP-100000	100	10	2	67	1	64	61	61
total	400	10	2	144	1	146	145	145

Tabla 4.4: casos de prueba resueltos óptimamente utilizando el operador de cruza GLX-V1 con los ordenamientos: Aleatorio, Items, Fullness, Items-Fullness, Fullness-Items, Fullness-Weights-Items y Fullness-Items-Weights.

Clase	Casos	Aleatorio	Items	Fullness	Items - Fullness	Fullness - Items	Fullness -Weight- Items	Fullness -Items- Weight
BPP.25-1000000	100	0	0	82	1	96	77	96
BPP.5-100000	100	0	0	40	0	94	34	94
BPP.75-100000	100	0	0	6	0	13	8	14
BPP-100000	100	12	0	81	0	78	74	78
total	400	12	0	209	1	281	193	282

información ya que solo se tiene posibilidad de heredar, aproximadamente, la mitad de los genes de los padres a cada uno de los hijos.

Los resultados obtenidos con el operador ESX se muestran en la Tabla 4.3. Para este caso se obtuvo que el método de ordenamiento Items-Fullness no muestra resultados que beneficien al operador, ya que solo logra obtener 2 óptimos. Por otro lado, con el ordenamiento Fullness-Items, se obtienen 146 óptimos, mostrando ser el mejor ordenamiento para el operador, con respecto al número de óptimos encontrados.

Con respecto al último operador de cruza utilizado, es decir el GLX-V1, los resultados se presentan en la Tabla 4.4, aquí se obtiene un mínimo de 0 óptimos cuando se utiliza el ordenamiento Items, y se obtiene un máximo de 282 óptimos cuando se utiliza el ordenamiento Fullness-Items-Weights, mostrando un impacto en el desempeño del algoritmo.

Después de los experimentos anteriores, se puede concluir que el operador UX-Sorted no tiene impacto en el desempeño del algoritmo ya que únicamente logra resolver óptimamente un problema con los distintos métodos de ordenamiento propuestos. Por otro lado, se concluye también que el ordenamiento Fullness-Items y el ordenamiento Fullness-Items-Weights muestran un impacto significativo en el desempeño del algoritmo ya que con ellos se obtiene el mayor número de óptimos con los operadores ESX y GLX-V1, respectivamente. El método de ordenamiento Fullness-Items realiza un menor número de operaciones al hacer la comparación de los genes, lo que da como resultado un menor tiempo computacional, mientras que el ordenamiento Fullness-Items-Weights, con el que se obtiene un óptimo más, realiza más operaciones de comparación, siendo costoso computacionalmente, razón por la que se selecciona el ordenamiento Fullness-Items como el método de ordenamiento predeterminado en el operador. A partir de este experimento, se trabajará con el operador de cruza GLX-V1 ya que con él se obtiene el mayor número de óptimos.

El experimento final del método de ordenamiento consistió en ejecutar el algoritmo con el operador GLX-V1 y el ordenamiento Fullness-Items con las clases de pruebas del banco de prueba clásico y el banco de prueba BPPv_u_c completo.

Con respecto al banco de prueba clásico, los resultados se presentan en la Tabla 4.5. Para este conjunto, se hicieron ejecuciones del GGA-CGT variando el porcentaje de cruza desde el 0 % hasta el 90 % incrementando de 10 en 10, debido a que la cruza para los experimentos de ordenamiento estaba configurada para el banco de prueba pertenecientes al conjunto BPPv_u_c. Como se observa, se logra obtener un máximo de 1597 óptimos con un 10 % y un 70 % de cruza, lo que indica que el operador de cruza no está mostrando un impacto en el resultado del operador. El mínimo de óptimos se encuentra cuando no se cruza, siendo un total de 1584.

Con respecto al conjunto BPPv_u_c, los resultados son mostrados en la Tabla 4.6. Para este caso, se utilizó una configuración del porcentaje de cruza igual a un 90 % debido a que cuando se realizó la comparación del desempeño de los operadores de cruza para este banco de prueba, en el Capítulo 3.3, se observó un mejor desempeño por parte del algoritmo GGA-CGT con el operador GLX-V1. Como se observa en la Tabla 4.6, se logran alcanzar 69 óptimos más con la modificación en el ordenamiento de los genes, que con

Tabla 4.5: Resultados de óptimos obtenidos del banco de prueba clásico con el operador de cruza GLX-V1, con el ordenamiento Fullness-Items y porcentajes de cruza desde el 0 % hasta el 90 %.

Clase	Inst	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	712	718	718	717	719	718	719	718	718	718
Dataset 2	480	480	480	480	480	480	480	480	480	480	480
Dataset 3	10	9	10	10	10	9	9	9	10	9	9
triplets	80	80	80	79	79	77	78	79	80	79	77
uniform	80	80	80	80	80	79	80	79	80	80	79
hard28	28	8	13	13	12	13	14	13	13	13	12
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	16	16	16	16	16	16	16	16	16
Total	1615	1584	**1597**	1596	1594	1593	1595	1595	**1597**	1595	1591

respecto a la verisón original del operador GLX-V1 (los resultados también son mostrados en la Tabla 4.6).

Tabla 4.6: Óptimos encontrados mediante el algoritmo con el operador GLX-V1 en su versión original y cuando se utiliza el ordenamiento Fullness-Items.

Clase	Casos	Versión original de GLX-V1	Ordenamiento Fullness-Items
BPP.25	700	482	475
BPP.5	700	342	396
BPP.75	700	503	524
BPP1	700	623	624
Total	2800	1950	**2019**

Finalmente, después de la experimentación sobre el método de ordenamiento, se usará como base el operador GLX-V1 que muestra ser el operador que al ser implementado en el algoritmo muestra tener un mejor desempeño. Con respecto al ordenamiento, se utilizará Fullness-Items ya que con él se obtienen los mejores resultados con respecto al número de óptimos encontrados.

4.2. Transmisión de genes

Después de definir el método de ordenamiento a usar, es decir Fullness-Items, se procedió a generar las propuestas de transmisión de genes. Como se mencionó en la subsección anterior, se trabajará exclusivamente con el operador GLX-V1 como operador base.
Considerando el orden de los genes obtenido por el método Fullness-Items, contenedores individuales de ambos padres son comparados en paralelo, contenedor por contenedor. Para cada par de contenedores, se considerarán cinco estrategias para decidir cuál contenedor será el primero en ser heredado a la nueva solución (hijo), seguido de inmediato por el otro contenedor. Las estrategias se basan en tres criterios: llenado de los contenedores, número de objetos en los contenedores y aleatoriedad. Las cinco estrategias son descritas a continuación.

4.2.1. Fullness

Para el método de transmisión Fullness, los genes de las soluciones padres son comparados en paralelo, para cada par de genes, se transmite primero el gen del padre con mayor llenado y después se transmite el otro gen. Si ambos tienen el mismo llenado, le da prioridad al primer padre. Este método de transmisión es el que utiliza el algoritmo originalmente.

4.2.2. Fullness-Items

El método Fullness-Items, compara los genes en paralelo, por cada par de genes, transmite primero el gen del padre con mayor llenado y después se transmite el otro gen, si es el caso que ambos genes tienen el mismo llenado, se utiliza el criterio del número de objetos, es decir, se transmite primero aquel que tenga menos objetos, para después transmitir el otro gen.

4.2.3. Fullness-Random

Este método también compara los genes de los padres en paralelo, por cada par, compara el llenado de los genes, si es el caso que ambos tienen el mismo llenado, selecciona aleatoriamente alguno de los dos dos genes y transmite primero ese gen y después el siguiente.

4.2.4. Random

En el método de transmisión Random, los genes de los padres son tomados en paralelo, eligiendo alguno de los dos aleatoriamente para poner primero, seguido de inmediato por el otro gen.

4.2.5. Items

Este método, se comparan los genes de los padres en paralelo. Por cada par de genes, se transmite primero el gen con menos objetos y después el otro gen. Si es el caso que ambos tienen el mismo número de objetos, se toma primero el gen del primer padre.

4.2.6. Resultados de la transmisión de genes

Al igual que los experimentos sobre ordenamiento de genes descritos en la Sección 4.1.8, esta serie de experimentos se trabajó primero con 400 casos pertenecientes al banco de prueba BPPv_u_c, para después de seleccionar el mejor método de ordenamiento de genes, medir el desempeño que se obtiene con este al evaluar con los dos bancos de prueba completos.

Se decidió trabajar primero con el banco BPPv_u_c ya que se tiene una mayor área de oportunidad y complejidad. Los 400 casos de prueba fueron seleccionadas estratégicamente de las conjuntos más difíciles, como se muestra a continuación: clase BPP.25, conjunto: 10^6, clase BPP.5, conjunto: 10^5, clase BPP.75, conjunto: 10^5 y clase BPP1, conjunto 10^5.

Las cinco estrategias de transmisión fueron integradas en el operador GLX-V1 e implementadas en el GGA-CGT. El algoritmo fue ejecutado con la configuración propuesta por

Quiroz-Castellanos *et al.* (2015), en donde el único parámetro modificado con respecto a la configuración original es el porcentaje de cruza, que se estableció en 90 %, debido a los resultados obtenidos en el capítulo anterior en el conjunto BPPv_u_c. En todos los casos se utilizó la estrategia Fullness-Items para ordenar los genes de los padres y la heurística FFD para reinsertar los objetos libres.

Los resultados obtenidos se muestran en la Tabla 4.7. Como se observa, los mejores resultados son obtenidos cuando se utiliza la transmisión Fullness-Items, obteniendo 286 óptimos de los 400. Por otro lado, los peores resultados son obtenidos cuando se utiliza el criterio de transmisión Random, obteniendo únicamente 275 óptimos.

Tabla 4.7: Número de óptimos obtenidos con los métodos de transmisión propuestos.

Clase	Casos	Fullness	Fullness-Items	Fullness-Random	Random	Items
BPP.25	100	96	98	95	91	98
BPP.5	100	94	95	93	96	96
BPP.75	100	13	14	13	14	12
BPP1	100	78	79	79	74	73
Total	400	281	286	280	275	279

A partir de los resultados obtenidos, se concluyó que el mejor método de transmisión es el Fullness-Items, esto debido al número de casos de prueba que resuelve óptimamente. Es por ello que se realizó un experimento con todos los casos de prueba del banco de prueba clásico y el banco de prueba BPPv_u_c, con el objetivo de medir el desempeño del algoritmo con la propuesta de transmisión. Para esta serie de experimentos en todos los casos, dentro del operador GLX-V1, se utilizó la estrategia Fullness-Items para ordenar los genes de los padres, la transmisión de genes Fullness-Items y la heurística FFD para reinsertar los objetos libres.

Los resultados obtenidos para el banco de prueba clásico se presentan en la Tabla 4.8. Para este banco de pruebase hicieron ejecuciones con porcentajes de cruza del 0 % al 90 % incrementando de 10 en 10 el porcentaje, esto debido a que el algoritmo estaba configurado para los casos de prueba del conjunto BPPv_u_c. Como se observa, se logran resolver

Tabla 4.8: Resultados de óptimos obtenidos por el GGA-CGT con el operador de cruza GLX-V1, con el ordenamiento Fullness-Items, la transmisión Fullness-Items y porcentajes de cruza desde el 0 % hasta el 90 %.

Clase	Casos	Porcentaje de cruza									
		0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Dataset 1	720	712	719	719	719	718	719	719	718	718	718
Dataset 2	480	480	480	480	480	480	480	480	480	480	480
Dataset 3	10	9	10	10	10	10	9	9	9	9	9
triplets	80	80	80	79	80	77	77	80	79	78	79
uniform	80	80	80	80	80	80	79	79	79	79	79
hard28	28	8	15	15	16	15	15	14	11	14	11
was1	100	100	100	100	100	100	100	100	100	100	100
was 2	100	100	100	100	100	100	100	100	100	100	100
gau 1	17	15	16	16	16	16	16	16	16	16	16
Total	1615	1584	1600	1599	**1601**	1596	1595	1597	1592	1594	1592

óptimamente 1601 casos de prueba con un 30 % de cruza, siendo esta su mejor configuración, mientras que su peor configuración se obtiene con un 70 % y un 90 % obteniendo únicamente 1592 óptimos.

Se observa una mejoría utilizando el ordenamiento Fullness-Items y la transmisión de genes Fullness-Items con respecto a cuando solo se utilizaba el ordenamiento Fullness-Items (resultados presentados en Tabla 4.5), ya que se logran obtener 4 óptimos más.

Por otro lado, para el banco de prueba BPPv_u_c, los resultados se presentan en la Tabla 4.9. Como se observa, se obtienen 2048 óptimos con la propuesta de ordenamiento y transmisión, siendo un total de 29 óptimos más que cuando se utiliza únicamente el ordenamiento Fullness-Items (los resultados se muestran en la Tabla 4.6). Los cual indica que este tipo de transmisión genera una mejora al operador de cruza y por consecuente al desempeño del algoritmo GGA-CGT para el problema del 1D-BPP.

Después de esta experimentación, se concluyó que la transmisión de genes que mejores resultados genera, esto con respecto al número de óptimos encontrados, es la transmisión Fullness-Items, que al igual que la estrategia de ordenamiento, considera primero el llenado de los contenedores y en segundo lugar el número de objetos. Esta estrategia se utilizará para los siguientes experimentos.

Tabla 4.9: Óptimos encontrados mediante el algoritmo con el operador GLX-V1 cuando se utiliza el ordenamiento Fullness-Items y la transmisión de genes Fullness-Items.

Clase	Casos	Ordenamiento Fullness-Items Transmisión Fullness-Items
BPP.25	700	498
BPP.5	700	398
BPP.75	700	522
BPP1	700	630
total	2800	2048

4.3. Reinserción de objetos libres

Después de los experimentos de ordenamiento y transmisión de genes, se concluyó que los mejores métodos son los que consideran primero el llenado de los contenedores y en segundo lugar el número de objetos (Fullness-Items). Debido a lo anterior, el último experimento para reinserción de objetos libres utilizará el ordenamiento y la transmisión de genes Fullness-Items.

En el operador GLX-V1 los genes heredados al hijo con objetos duplicados son eliminados de la nueva solución, y los objetos que queden fuera de la solución son reinsertados con la heurística determinista FFD. En este último experimento, además de FFD, se explorará el desempeño de la estrategia de reinserción incluida en el operador de mutación del GGA-CGT, propuesta por Quiroz-Castellanos *et al.* (2015), denominada Rearrangement by Pairs (RP). La heurística RP se compone de dos etapas: primero, se recorre cada gen en un intento de mejorar su llenado haciendo intercambios entre pares de objetos empacados y objetos libres; segundo, los objetos libres se introducen en la solución utilizando la heurística FF.

4.3.1. Resultados de la reinserción de los objetos libres

Al igual que los experimentos anteriores, se trabajó con 400 casos de prueba pertenecientes al banco de prueba BPPv_u_c. Se seleccionaron 100 casos de prueba de cada clase, como a continuación se especifica.

- clase: BPP.25, conjunto: 1000000
- clase: BPP.5, conjunto: 100000
- clase: BPP.75, conjunto: 100000
- clase: BPP1, conjunto: 100000

La estrategia de reinserción RP fue integrada en el operador GLX-V1 e implementada en el GGA-CGT. El algoritmo fue ejecutado con la configuración propuesta por Quiroz-Castellanos *et al.* (2015), en donde el único parámetro modificado con respecto a la configuración original es el porcentaje de cruza, que se estableció en 90 %, debido a los resultados obtenidos en el capítulo anterior en el conjunto BPPv_u_c. Se utilizó la heurística Fullness-Items para ordenar los genes de los padres y la heurística Fullness-Items para la transmisión de genes.

Para el método FFD, los resultados fueron presentados en la Tabla 4.7 y discutidos en la Subsección 4.2.6. Con respecto al método RP los resultados son presentados en la Tabla 4.10. Con este tipo de reinserción, se logran encontrar 286 óptimos de los 400, dando el mismo número de óptimos que el método FFD.

Debido a que el método FFD tiene un menor costo computacional, será el método de

Tabla 4.10: Óptimos encontrados mediante el algoritmo con el operador GLX-V1 cuando se utiliza el ordenamiento Fullness-Items, la transmisión de genes Fullness-Items y la reinserción de objetos libres con RP.

Clase	Casos	Ordenamiento Fullness-Items Transmisión Fullness-Items Reinserción RP
BPP.25	100	99
BPP.5	100	96
BPP.75	100	14
BPP1	100	77
total	400	286

reinserción a utilizar. Los resultados para este método cuando se aplican a los banco de prueba clásico y BPPv_u_c fueron discutidos en la Subsección 4.2.6.

Capítulo 5

Fullness-Items Gene-Level Crossover 1 (FI-GLX-1)

En el capítulo anterior se discutieron las características que debía tener el nuevo operador de cruza. Después, a partir de las características obtenidas se comenzó el diseño del nuevo operador de cruza, el cual se enfocó en la propuesta de métodos para los tres procesos dentro la cruza: ordenamiento de genes, transmisión de genes y reinserción de objetos libres. Estos procesos fueron identificados como aquellos que podrían beneficiar los resultados obtenidos al implementarse en el operador de cruza y el algoritmo. Como resultado final, se obtuvo el nuevo operador de cruza propuesto: Fullness-Items Gene-Level Crossover 1 (FI-GLX-1).

El operador FI-GLX-1, es un operador de cruza orientado a grupos. En este operador, de dos padres p_1 y p_2 se obtiene un único hijo c. Su funcionamiento consiste en primero ordenar los genes de los padres utilizando el método Fullness-Items, después utilizar el método de transmisión Fullness-Items, para finalmente, reinsertar los objetos libres a la solución utilizando la heurística FFD. Se muestra un ejemplo ilustrativo en la Figura 5.1. Este operador a diferencia del operador GLX-V1 utiliza dos criterios en el ordenamiento y la transmisión de genes.

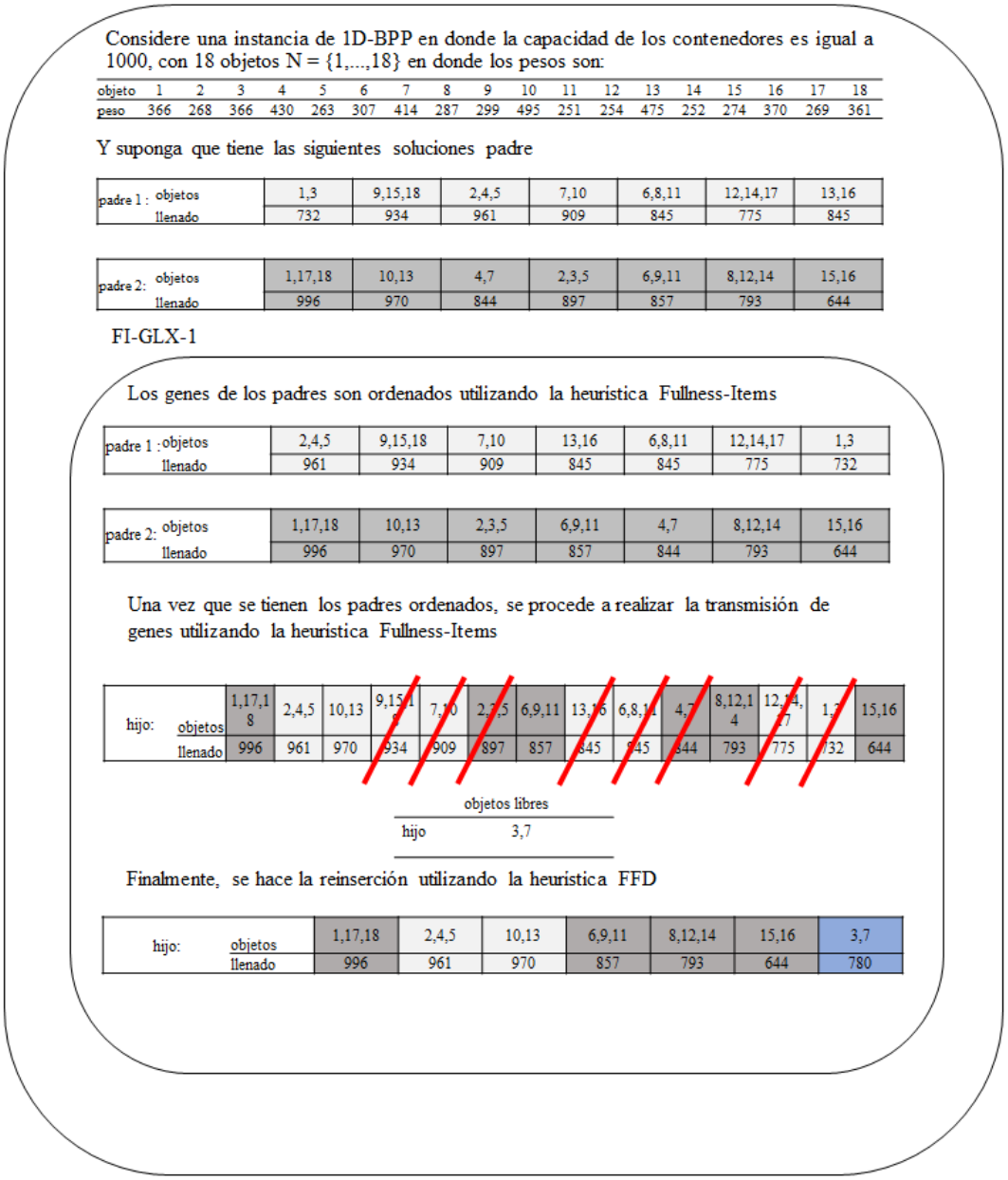


Figura 5.1: Ejemplo ilustrativo del operador de cruza FI-GLX-1.

5.1. Configuración de parámetros del GGA-CGT

Con la propuesta del nuevo operador de cruza FI-GLX-1, y su incorporación en el GGA-CGT (reemplazando el operador de cruza original), fue necesario llevar a cabo la configuración de los parámetros del algoritmo. Los parámetros que se configuraron son: número de individuos a ser cruzados n_c, número de individuos a ser mutados n_m, la tasa de cambio que se utiliza para calcular el número de genes a eliminar n_b en el operador de mutación para las soluciones no clonadas k_{ns}, el número de individuos en el grupo élite $|B|$, la tasa de cambio que se utiliza para calcular el número de genes a eliminar n_b en el operador de mutación para las soluciones clonadas k_{cs} y la edad máxima para un individuo clonado $life\_span$. Los parámetros: tamaño de la población P y el número máximo de generaciones max_gen, se mantuvieron como en la versión predeterminada del GGA-CGT.

Debido a que después de la serie de experimentos fue notorio que un mayor porcentaje de cruza beneficiaba el rendimiento del algoritmo en el banco de prueba BPPv_u_c y un menor porcentaje de cruza favorecía el desempeño del algoritmo en el banco de prueba clásico, se optó por realizar la calibración de parámetros por separado para cada banco de prueba. Los valores de los parámetros fueron configurados por medio de un enfoque experimental (Rangel-Valdez *et al.*, 2009) que se basa en covering arrays para obtener un conjunto mínimo representativo de las posibles configuraciones para evaluar el impacto de los parámetros del algoritmo. Los parámetros son presentados en la Tabla 5.1.

Tabla 5.1: Configuración de parámetros para bancos de prueba clásico y BPPv_u_c.

Parámetro	banco de prueba	
	Clásico	BPPv_u_c
P	100	100
max_gen	500	500
n_m	0.82	0.82
n_c	0.22	0.88
k_{ns}	1.334	1.74
k_{cs}	5.28	4.1
|B|	0.12	0.04
$life\_span$	20	8

Tabla 5.2: Óptimos obtenidos para el banco de prueba clásico con el algoritmo GGA-CGT cuando se utiliza operador de cruza FI-GLX-1.

Clase	**Casos**	**Óptimos**
Dataset 1	720	720
Dataset 2	480	480
Dataset 3	10	10
triplets	80	80
uniform	80	80
hard28	28	17
was1	100	100
was 2	100	100
gau 1	17	16
Total	1615	**1603**

Con respecto al banco de prueba clásico, utilizando el operador FI-GLX-1 en el algoritmo GGA-CGT, con la configuración de parámetros nueva, se logran obtener un total de 1603 óptimos, los resultados se presentan en la Tabla 5.2.

Para el caso del conjunto BPPv_u_c, los resultados son presentados en la Tabla 5.3. Con la nueva configuración de parámetros, se logra obtener un total de 2128 óptimos de 2800.

Tabla 5.3: Óptimos obtenidos para el banco de prueba BPPv_u_c con el algoritmo GGA-CGT cuando se utiliza operador de cruza FI-GLX-1.

Clase	**Casos**	**Óptimos**
BPP.25	700	507
BPP.05	700	414
BPP.75	700	566
BPP1	700	641
Total	2800	2128

5.2. Prueba de Robustez

Se realizó una prueba de robustez realizando la ejecución del algoritmo GGA-CGT con el operador FI-GLX-1. El experimento consistió en ejecutar, para cada uno de los conjuntos, cinco corridas del algoritmo con diferentes semillas de números aleatorios. Esta prueba

se realizó para las clases de de prueba por separado.

Para el banco de prueba clásico, el algoritmo encontró las soluciones óptimas en 8000 ejecuciones, y en 75 ejecuciones no se encontró la solución óptima. El algoritmo GGA-CGT en su versión original (Quiroz-Castellanos *et al.*, 2015), no consigue obtener soluciones óptimas en un total de 78 ejecuciones.

Con respecto al banco de prueba BPPv_u_c, la ejecución se realizó para la versión original del algoritmo y para el algoritmo con el operador propuesto. Para el operador propuesto, se obtuvo la solución óptima en un total de 10,597 ejecuciones y no se obtuvo la solución óptima en 3,403 ejecuciones. Para el algoritmo GGA-CGT en su versión original, no consigue obtener soluciones óptimas en un total de 6,921 ejecuciones.

Los resultados obtenidos, muestran que la propuesta de operador, genera buenos resultados y robustos en clases de pruebas difíciles. Se puede concluir que el algoritmo GGA-CGT con el operador FI-GLX-1 muestra una alta precisión y robustez, superando la versión original del algoritmo que incluía el operador GLX.

5.3. Ejecución a largo plazo de GGA-CGT

Al igual que con el GGA-CGT con el operador de cruza original (Quiroz-Castellanos *et al.*, 2015), se investigó cómo los valores de max_gen influyen en el desempeño del GGA-CGT con el operador FI-GLX-1. En la Tabla 5.4 se muestra la probabilidad de encontrar una solución óptima para el banco de prueba clásico, utilizando el algoritmo GGA-CGT con el operador FI-GLX-1, con respecto al valor máximo de generaciones (max_gen), esto para cada clase del banco de prueba. Para el banco de prueba se observa que la probabilidad de encontrar una solución óptima aumenta de 99.25 a 99.56 cuando el número de generaciones pasa de 500 a 5,000,000.

Para el banco de prueba BPPv_u_c, los resultados de la probabilidad de encontrar una solución óptima, utilizando el algoritmo GGA-CGT con el operador FI-GLX-1, con respecto al valor máximo de generaciones (max_gen), esto para cada clase del banco de prueba, se muestran en la Tabla 5.5. Para el banco de prueba se observa que la probabilidad de encontrar una solución óptima aumenta de 76 a 86 cuando el número de generaciones

Tabla 5.4: Probabilidad de encontrar una solución óptima por clase de prueba para el banco de prueba clásico.

max_gen	**Porcentaje de soluciones óptimas por clase**									
	U	**T**	**Set 1**	**Set 2**	**Set 3**	**Was 1**	**Was 2**	**Was 3**	**Gau1**	**Hard 28**
500	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	94.11	60.71
5000									94.11	67.85
10000									94.11	67.85
100000									94.11	67.85
1000000									94.11	71.42
5000000									94.11	78.57

Tabla 5.5: Probabilidad de encontrar una solución óptima por clase de prueba para el banco de prueba BPPv_u_c.

max_gen	Porcentaje de soluciones óptimas por clase			
	BPP.25	BPP.5	BPP.75	BPP1
500	72.43	59.14	80.86	91.57
5000	86.14	74.86	87.71	95.00

pasa de 500 a 5,000. Mostrando que el algoritmo con el operador de cruza FI-GLX-1 alcanza un alto grado de precisión y robustez y es capaz de encontrar la solución óptima de una gran variedad de casos de prueba de 1D-BPP.

5.4. Comparación de GGG-CGT con GLX y FI-GLX-1

Después de la fase experimental, se realizó una comparación entre el algoritmo GGA-CGT con su operador predeterminado (GLX) y con el operador FI-GLX-1, en ambos bancos de prueba, para comparar el número de casos en los que se obtiene una solución óptima y el número promedio de generaciones para cada clase de pruebas.

En la Tabla 5.6 se muestran los resultados obtenidos para el banco de prueba clásico. Con el algoritmo original, se logra obtener un total de 1602 óptimos y tarda 90.23 generaciones en promedio. Respecto al algoritmo con la propuesta de operador de cruza, FI-GLX-1, se obtiene un total de 1603 óptimos y tarda 89.90 generaciones en promedio. La propuesta muestra una mejora con respecto al número de generaciones promedio y el número de óptimos encontrados.

Tabla 5.6: Comparación de óptimos encontrados y generaciones promedio realizadas, al resolver el banco de prueba clásico, utilizando los operadores de cruza GLX y FI-GLX-1 en el algoritmo GGA-CGT.

Clase	**Casos**	**GLX**		**FI-GLX-1**	
		Gen. Promedio	**Óptimos**	**Gen. Promedio**	**Óptimos**
Dataset 1	720	131.958333	720	131.459722	720
Dataset 2	480	24.58125	480	24.7333333	480
Dataset 3	10	361.8	10	382.7	10
triplets	80	40.475	80	36.35	80
uniform	80	10.9375	80	10.975	80
hard28	28	385.464286	16	379.035714	17
was1	100	0.67	100	0.86	100
was 2	100	185.24	100	185.34	100
gau 1	17	105.705882	16	107.411765	16
Total	**1615**	**90.23**	**1602**	**89.90**	**1603**

Con respecto al banco de prueba BPPv_u_c, los resultados se muestran en la Tabla 5.7. Para el algoritmo original, con el operador de cruza predeterminado, se obtiene un total de 1442 óptimos y un promedio de 277.96 generaciones. Con respecto al operador FI-GLX-1, se logran obtener 2128 óptimos en un total de 169.80 generaciones promedio. Se obtienen mejores resultados cuando se utiliza el operador propuesto, teniendo un menor número de generaciones promedio y un mayor número de óptimos obtenidos.

Tabla 5.7: Comparación de óptimos encontrados y generaciones promedio realizadas, al resolver el banco de prueba BPPv_u_c, utilizando los operadores de cruza GLX y FI-GLX-1 en el algoritmo GGA-CGT.

Clase	**Casos**	**GLX**		**FI-GLX-1**	
		Gen. promedio	**Óptimos**	**Gen. promedio**	**Óptimos**
BPP.25	700	232.71286	415	180.62286	507
BPP.5	700	298.14857	312	241.85286	414
BPP.75	700	345.49571	247	169.78000	566
BPP1	700	235.47000	468	86.98000	641
		277.96	**1442**	**169.80**	**2128**

5.5. Prueba estadística

Con el objetivo de observar las diferencias entre el algoritmo original GGA-CGT y el algoritmo con el operador de cruza FI-GLX-1, se realizó la prueba estadística Wilcoxon Rank-sum. La prueba fue realizada a una muestra de 30 ejecuciones del algoritmo original y el algoritmo con FI-GLX-1. Primero se calculó el error de los resultados para cada problema en cada ejecución. El error está dado por la diferencia relativa, definida como $(y-x)/x$, donde x corresponde al número óptimo de contenedores a utilizar y y al número de contenedores obtenidos por el algoritmo.

Se calculó el promedio de errores de las 30 ejecuciones para cada problema. Después, se realizó la prueba estadística a cada clase de pruebas. También se aplicó la prueba estadística al número promedio de contenedores obtenidos en las 30 ejecuciones.

Para el banco de prueba clásico, los p-values se encuentran en la Tabla 5.8. Debido a todos los valores de p son mayores que 0.05, se puede afirmar sin pérdida de la generalidad que no existe una diferencia significativa en el desempeño del algoritmo GGA-CGT en su versión original, es decir, cuando utiliza el operador de cruza GLX y cuando utiliza el operador de cruza FI-GLX-1. Se puede concluir que ninguno de los operadores GLX y FI-GLC-1 en el algoritmo GGA-CGT es mejor para la solución de los casos de prueba pertenecientes al banco de prueba clásico.

Tabla 5.8: p-value obtenido por clase de la prueba estadística Wilcoxon con respecto al promedio del error y al promedio de contenedores utilizados, para el banco de prueba clásico.

Clase	**GLX vs FI-GLX-1**	
	error promedio	**Contenedores promedio**
Dataset 1	0.99994944	0.99989887
Dataset 2	1	1
Dataset 3	0.96984998	0.96984998
Triplets	0.05599608	0.65730262
Uniform	0.89142229	0.99455429
Hard 28	0.6463552	0.92167738
was 1	1	1
was 2	1	1
gau	0.78289506	0.9862597

Con respecto al banco de prueba BPPv_u_c los resultados se muestran en la Tabla 5.9. Se presentan los p-values por clase de pruebas, para el error promedio y los contenedores promedio. En este banco de prueba, se muestra una diferencia significativa entre el algoritmo GGA-CGT con el operador GLX y con la propuesta de operador de cruza FI-GLX-1, mostrando ser mejor con el operador propuesto, esto con un 95 % de confianza.

Tabla 5.9: p-value obtenido por clase de la prueba estadística Wilcoxon con respecto al promedio del error y al promedio de contenedores utilizado, para el banco de prueba BPPv_u_c.

Clase	**GLX vs FI-GLX-1**	
	error promedio	**Contenedores promedio**
BPP.25	3.47×10^{-6}	3.47×10^{-6}
BPP.5	4.06×10^{-6}	4.06×10^{-6}
BPP.75	2.01×10^{-58}	2.01×10^{-58}
BPP1	8.81×10^{-54}	8.81×10^{-54}

5.6. Comparación del GGA-CGT con el estado del arte

Con el objetivo de medir el desempeño del algoritmo GGA-CGT con la propuesta del operador FI-GLX-1, los resultados de la solución para el banco de prueba BPPv_u_c se compararon con obtenidos por los mejores algoritmos para la solución del 1D-BPP: Arc-Flow Formulation with Graph Compression (Brandão & Pedroso, 2016), CNS_BP (Buljubašić & Vásquez, 2016) y la propuesta GGA-CGT/D (González-San-Martín, 2021). Los resultados del algoritmo Arc-Flow Formulation with Graph Compression, CNS_BP y GGA-CGT/D son presentados en la tesis de González-San-Martín (2021). Las características del equipo de cómputo que utilizaron son las mencionadas a continuación: sistema operativo CentOS release 6.7, versión 5.1.0 de gcc y procesador Intel Xeon E5-2650 2.30GHz.

Para el caso del algoritmo Arc-Flow Formulation, se utilizó un tiempo de 10,000 segundos por problema, con respecto al algoritmo CNS_BP, se le dieron en total 100 millones de iteraciones por caso de prueba, los parámetros anteriores fueron calibrados con el objetivo de estar en condiciones similares al tiempo que tarda el algoritmo GGA-CGT/D. Los parámetros utilizados para GGA-CGT con el operador FI-GLX-1 y GGA-CGT/D se muestran

Tabla 5.10: Parámetros utilizados por los algoritmos GGA-CGT/D y GGA-CGT con el operador de cruza FI-GLX-1 para resolver el banco de prueba BPPv_u_c.

Parámetro	**Parámetros de los algoritmos**	
	GGA-CGT/D	**GGA-CGT con FI-GLX-1**
P	100	100
max_gen	1000000000	5000
n_m	0.20	0.88
n_c	0.83	0.82
k_{ns}	1.3	1.74
k_{cs}	4	4.1
$\|B\|$	10	0.04
$life\_span$	10	8

en la Tabla 5.10. Adicional a los parámetros principales, el algoritmo GGA-CGT/D, utiliza un porcentaje de Diversificación de 10 % y 15 % de P a $\frac{1}{2}$ de max_gen.

Para efectos de una comparación justa y dado que los algoritmos entre los que se realiza utilizan un gran número de iteraciones, generaciones o un tiempo alto, se usa la versión del algoritmo GGA-CGT con FI-GLX-1 cuando se le dan cinco mil generaciones.

Los resultados mostrados, son la comparación del número de óptimos obtenidos por clase de pruebas, los cuales son mostrados en la Tabla 5.11. Para la clase BPP.25, se observa que se obtienen mejores resultados utilizando los otros tres algoritmos que con la propuesta FI-GLX-1, ya que se logran obtener 603 óptimos, 84 óptimos menos (-12 % del banco de prueba total) que el máximo número de óptimos encontrados por el mejor algoritmos para esta clase, es decir, el algoritmo General-Arc-Flow Formulation, sin embargo, es importante recalcar que la propuesta utiliza un número menor de generaciones y por consecuente el tiempo de ejecución es menor que el realizado por el algoritmo con mejor desempeño para esta clase.

Para la clase BPP.5, se observa, que con la propuesta FI-GLX-1 se obtienen 524 óptimos quedando en tercer lugar con respecto al número de óptimos obtenidos. Para esta clase, el algoritmo GGA-CGT/D logra obtener los mejores resultados, resolviendo un total de 542 casos de prueba, 19 más (2.71 % más del banco de prueba total) que la propuesta FI-GLX-1. Es importante recalcar que el algoritmo GGA-CGT/D utiliza 999,995,000 generaciones más que la propuesta GGA-CGT con FI-GLX-1 para cada problema.

Con respecto a la clase BPP.75, se observa para esta clase de pruebas un mejor desempeño del algoritmo GGA-CGT con el operador FI-GLX-1 que, con los otros cuatro métodos mencionados, esto con respecto al número de óptimos obtenidos, siendo 614 óptimos, lo que corresponde a 212 (30.29 % más del banco de prueba total) óptimos más que el algoritmo GGA-CGT/D (segundo mejor algoritmo para esta clase de pruebas).

Finalmente, para la clase BPP1, en donde se observa un buen resultado con el operador de cruza FI-GLX-1 en el algoritmo GGA-CGT, obteniendo 665 óptimos, siendo el segundo mejor algoritmo para esta clase, con solo 10 óptimos menos (1.43 % menos que el banco de prueba total) que el GGA-CGT/D (mejor algoritmo para esta clase de pruebas), recordando que utiliza 999,995,000 generaciones más que la propuesta con FI-GLX-1.

El algoritmo GGA-CGT con la propuesta del operador FI-GLX-1 obtiene 2406 óptimos, teniendo el mejor desempeño con respecto al banco de prueba total. En segundo lugar, el algoritmo GGA-CGT/D, logra obtener 2306 óptimos, en tercer lugar, el algoritmo Arc-Flow Formulation with Graph Compression obtiene 2260 óptimos y en cuarto y último lugar, el algoritmo CNS_BP obteniendo 2072 óptimos.

Como conclusión de este experimento, el algoritmo GGA-CGT con FI-GLX-1 muestra un mejor desempeño para el banco de prueba BPPv_u_c, esto con respecto al número de óptimos, obteniendo un total de 2406, con una efectividad del 0.85929, siendo un 0.03572 superior al segundo algoritmo con mejor desempeño, el GGA-CGT/D. Además, es importante hacer énfasis en que el GGA-CGT con FI-GLX-1 únicamente utiliza un 0.0005 % de las generaciones que utiliza el algoritmo GGA-CGT/D para obtener ese número de óptimos.

En la Figura 5.2 se muestra un diagrama de la secuencia de experimentos realizados.

Tabla 5.11: Comparación de óptimos encontrados del banco de prueba BPPv_u_$c$con los algoritmos: Arc-Flow Formulation with Graph Compression, CNS_BP, GGA-CGT/D y GGA-CGT con operador de cruza FI-GLX-1.

Clase	Casos	General Arc-Flow Formulation	CNS_BP	GGA-CGT/D	GGA-CGT con FI-GLX-1
BPP.25	700	687	661	686	603
BPP.5	700	526	477	543	524
BPP.75	700	396	308	402	614
BPP1	700	651	626	675	665
Total	2800	2260	2072	2306	**2406**
Efectividad	1	0.80714	0.74	0.82357	**0.85929**

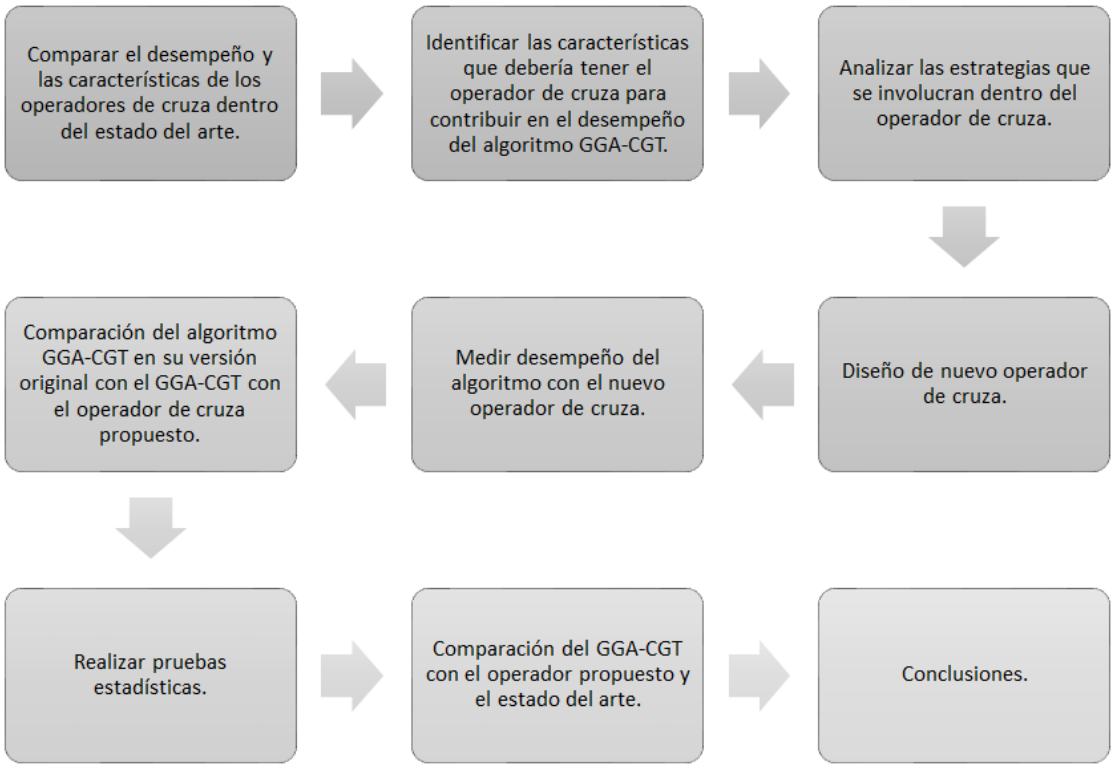


Figura 5.2: Diagrama de la secuencia experimental.

Capítulo 6

Conclusiones y trabajo futuro

En esta sección se presentan las conclusiones, los resultados obtenidos al realizar este proyecto de investigación, así como posibles propuestas para desarrollar el trabajo futuro.

6.1. Conclusiones

El trabajo de investigación que es presentado comprendía como objetivo inicial realizar un estudio de operadores de cruza para el problema del empacado de objetos en contenedores (1D-BPP) cuando es utilizado como estrategia de solución el algoritmo metaheurístico GGA-CGT, uno de los algoritmos del estado del arte para resolver el 1D-BPP. Como objetivo principal del trabajo se tenía que mejorar el desempeño del algoritmo GGA-CGT con respecto al número de casos de prueba resueltos óptimamente realizando una modificación al operador de cruza, lo que se logró a partir de la propuesta de un análisis experimental que dio como resultado un nuevo operador de cruza. El operador propuesto logró mejorar el desempeño del algoritmo, lo cual prueba la hipótesis de esta investigación.

La investigación realizada mostró que, en el operador de cruza, el generar solo un hijo es beneficioso debido a que se conserva la diversidad de la población, explorando de mejor manera en el espacio de búsqueda. También, el orden de los genes de los padres antes de comenzar la transmisión es de gran importancia, específicamente, ordenar poniendo primero a los genes más llenos primero y en caso de tener más de un gen con el mismo llenado, darle prioridad al que tiene menos objetos. Con respecto a la transmisión, también se mostró

que darles prioridad a los genes más llenos para heredar primero y en caso de estar igual de llenos, heredar primero el que tiene menos objetos, muestra ser favorecedor ya que se preservan los mejores genes de los padres. También es importante tener la consideración anterior ya que, si se eliminan los genes más llenos durante el proceso, existe la posibilidad de que el gen contenga objetos muy pesados y al reinsertarlos en la solución, la perjudique; lo mismo sucede para el caso de los genes con menos objetos, se puede suponer que la mayoría son objetos pesados y al eliminar esos genes y reinsertar a la solución, podría perjudicar a la misma.

El GGA-CGT con la propuesta de operador tiene un buen desempeño para resolver la gran mayoría de los casos de prueba de los bancos de prueba, sin embargo, existe un pequeño grupo de casos de prueba, que resultan difíciles de resolver óptimamente, debido a sus características como son la capacidad de los contenedores, que tan dispersos están los pesos y la diferencia de pesos entre el objeto más pequeño y el más grande, por lo que encontrar el óptimo no es tarea sencilla y se debe realizar un análisis profundo de características.

También se encontró que los resultados mostraron que los bancos de prueba requieren parámetros distintos para obtener buenos resultados, ya que el banco de prueba clásico requería de cruzar a una pequeña parte de la población y el banco nuevo requería cruzar una gran parte de ésta. Estos resultados se utilizaron para realizar un ajuste de todos los parámetros del algoritmo para el nuevo operador de cruza, resolviendo un mayor número de casos de prueba. Se observó que los parámetros dependen en gran medida de las características de los casos de prueba y su dificultad.

La ejecución a largo plazo realizada mostró que, para el banco de prueba nuevo, el GGA-CGT con el operador de cruza propuesto supera a todos los algoritmos dentro del estado del arte con los que se compara.

Se logró cumplir con todos los objetivos propuesto a lo largo de esta investigación. A continuación, se presentan las principales contribuciones de esta investigación:

1. Se realizó un diseño experimental para analizar y entender el comportamiento del algoritmo a partir del operador de cruza, con distintos bancos de prueba.

2. Se realizó un estudio de operadores de cruza para problemas de agrupamiento.

3. Se adaptaron e implementaron los mejores operadores de cruza para problemas de agrupamiento para el 1D-BPP y el algoritmo GGA-CGT.

4. Se presenta el operador de cruza FI-GLX-1.

5. Se realizó un análisis del desempeño del operador de cruza propuesto.

6. El operador propuesto permitió mejorar el desempeño del algoritmo GGA-CGT cuando resuelve el banco de prueba BPPv_u_c, utilizando 500 generaciones, logró incrementar la efectividad de un 51.5 % a un 76 %.

7. El operador propuesto permitió mejorar el desempeño del algoritmo GGA-CGT cuando resuelve el banco de prueba clásico para 1D-BPP, obteniendo un óptimo más.

6.2. Trabajo Futuro

Con el objetivo de dar continuidad al trabajo de investigación presentado, se identifican áreas de oportunidad para trabajo futuro:

1. Analizar el desempeño de otros GGAs para la solución del 1D-BPP, y así poder realizar modificaciones a operadores de variación que mejoren el desempeño.

2. Realizar un análisis a las características particulares de los casos de prueba en las cuales no se encuentra el óptimo con el GGA-CGT.

3. Realizar un análisis de los parámetros del GGA-CGT para encontrar la relación que existe entre éstos y el comportamiento algorítmico.

4. Realizar métodos adaptativos para los parámetros del GGA-CGT con respecto a las características de los casos de prueba.

5. Aplicar otros operadores de cruza y muta dentro del GGA-CGT, los cuales sean adaptativos dependiendo del comportamiento evolutivo.

6. Analizar las características del problema para encontrar la relación existente entre éstas y las características de los casos de prueba.

7. Generar una nueva heurística de reparación de soluciones para los operadores de cruza y muta.

Bibliografía

Alvim, A. C. F., Ribeiro, C. C., Glover, F. & Aloise, D. J. (2004). *A hybrid improvement heuristic for the one-dimensional bin packing problem.*

Basse, S. (1998). *Computer algorithms: Introduction to design and analysis.* Addison-Wesley Longman Publishing Co., Inc.

Bayraktar, T., Aydin, M. E., Düğenci, M. & Dugenci, M. (2014). *A memory-integrated artificial bee algorithm for 1-d bin packing problems.* https://www.researchgate.net/publication/282655741

Borgulya, I. (2020). A hybrid evolutionary algorithm for the offline bin packing problem. *Central European Journal of Operations Research*, *29*(2), 425-445. https://doi.org/10.1007/s10100-020-00695-5

Brandão, F. & Pedroso, J. P. (2016). Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, *69*, 56-67. https://doi.org/https://doi.org/10.1016/j.cor.2015.11.009

Buljubašić, M. & Vásquez, M. (2016). Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing. *Computers & Operations Research*, *76*, 12-21. https://doi.org/https://doi.org/10.1016/j.cor.2016.06.009

Carmona-Arroyo, G., Vázquez-Aguirre, J. B. & Quiroz-Castellanos, M. (2021). *One-dimensional bin packing problem: An experimental study of instances difficulty and algorithms performance* (Vol. 940). Springer, Cham.

Cruz-Reyes, L., Quiroz-Castellanos, M., Alvim, A. C. F., Huacuja, H. J. F., Gómez, C. & Torres-Jiménez, J. (2012). Heurísticas de agrupación híbridas eficientes para el problema de empacado de objetos en contenedores. *16*, 349-360.

Dokeroglu, T. & Cosar, A. (2014). Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms. *Computers & Industrial Engineering*, *75*, 176-186. https://doi.org/https://doi.org/10.1016/j.cie.2014.06.002

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, *2(1)*, 5-30. https://doi.org/https://doi.org/10.1007/BF00226291

Falkenauer, E. & Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. *Proceedings 1992 IEEE International Conference on Robotics and Automation*, 1186-1192. https://doi.org/10.1109/ROBOT.1992.220088

Fleszar, K. & Charalambous, C. (2011). Taverage-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. *European Journal of Operational Research*, *210(2)*, 176-184. https://doi.org/10.1016/j.ejor.2010.11.004

Fleszar, K. & Hindi, K. S. (2002). New heuristics for one-dimensional bin-packing. *Computers & operations research*, *29(7)*, 821-839. https://doi.org/hhttps://doi.org/10.1016/S0305-0548(00)00082-4

Fukunaga, A. S. (2008). A new grouping genetic algorithm for the multiple knapsack problem. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2225-2232. https://doi.org/10.1109/CEC.2008.4631094

González-San-Martín, J. E. (2021). *Un estudio formal de heurísticas para el problema de empacado de objetos de una dimensión.*

Kaaouache, M. A. & Bouamama, S. (2015). Solving bin packing problem with a hybrid genetic algorithm for vm placement in cloud [Knowledge-Based and Intelligent Information & Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings]. *Procedia Computer Science*, *60*, 1061-1069. https://doi.org/https://doi.org/10.1016/j.procs.2015.08.151

Kämpke, T. (1988). Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research*, *16*(1), 327-332. https://doi.org/10.1007/bf02283751

Kucukyilmaz, T. & Kiziloz, H. E. (2018). Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. *Computers & Industrial Engineering*, *125*, 157-170. https://doi.org/https://doi.org/10.1016/j.cie.2018.08.021

Loh, K. H., Golden, B. & Wasil, E. (2008). Solving the onedimensional bin packing problem with a weight annealing heuristic. *Computers & operations research*, *35(7)*, 2283-2291. https://doi.org/https://doi.org/10.1016/j.cor.2006.10.021

Martello, S. (1990). *Knapsack problems : Algorithms and computer implementations*. J. Wiley & Sons.

Ozcan, S. O., Dokeroglu, T., Cosar, A. & Yazici, A. (2016a). A novel grouping genetic algorithm for the one-dimensional bin packing problem on gpu. En T. Czachórski, E. Gelenbe, K. Grochla & R. Lent (Eds.), *Computer and information sciences* (pp. 52-60). Springer International Publishing.

Ozcan, S. O., Dokeroglu, T., Cosar, A. & Yazici, A. (2016b). A novel grouping genetic algorithm for the one-dimensional bin packing problem on gpu. En T. Czachórski, E. Gelenbe, K. Grochla & R. Lent (Eds.), *Computer and information sciences* (pp. 52-60). Springer International Publishing.

Quiroz-Castellanos, M. (2014). *Caracterización del proceso de optimización de algoritmos heurísticos aplicados al problema de empacado de objetos en contenedores*. Instituto Tecnológico de Tijuana.

Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez S., C., Huacuja, H. J. F. & Alvim, A. C. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Comput. Oper. Res.*, *55*(100), 52-64. https://doi.org/10.1016/j.cor.2014.10.010

Ramos-Figueroa, O., Quiroz-Castellanos, M., Mezura-Montes, E. & Kharel, R. (2021). Variation operators for grouping genetic algorithms: A review. *Swarm and Evolutionary Computation*, *60*, 100796. https://doi.org/https://doi.org/10.1016/j.swevo.2020.100796

Rangel-Valdez, N., Torres-Jiménez, J., Bracho-Ríos, J. & Quiz-Ramos, P. (2009). Problem and algorithm fine-tuning: A case of study using bridge club and simulated annealing. *IJCCI 2009 - International Joint Conference on Computational Intelligence, Proceedings*, 302-305. https://doi.org/10.5220/0002321303020305

Rivera, G., Cisneros, L., Sánchez-Solís, P., Rangel-Valdez, N. & Rodas-Osollo, J. (2020). Genetic algorithm for scheduling optimization considering heterogeneous containers: A real-world case study. *Axioms*, *9*(1). https://doi.org/10.3390/axioms9010027

Schoenfield, J. E. (2002). Fast, exact solution of open bin packing problems without linear programming. *Draft, US Army Space and Missile Defense Command, Huntsville, Alabama, USA*.

Scholl, A., Klein, R. & Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, *24*(7), 627-645. https://doi.org/https://doi.org/10.1016/S0305-0548(96)00082-2

Schwerin, P. & Wäscher, G. (1997). The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, *4*(5), 377-389. https://doi.org/https://doi.org/10.1016/S0969-6016(97)00025-7

Sim, K., Hart, E. & Paechter, B. (2012). A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. En C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia & M. Pavone (Eds.), *Parallel problem solving from nature - ppsn xii* (pp. 348-357). Springer Berlin Heidelberg.

Singh, A. & Gupta, A. K. (2007). Two heuristics for the one-dimensional bin-packing problem. *OR Spectr.*, *29*(4), 765-781. https://doi.org/https://doi.org/10.1007/s00291-006-0071-2

Sotelo-Figueroa, M. A., Soberanes, H. J. P., Carpio, J. M., Fraire Huacuja, H. J., Reyes, L. C. & Soria Alcaraz, J. A. (2013). Evolving bin packing heuristic using micro-differential evolution with indirect representation. En O. Castillo, P. Melin & J. Kacprzyk (Eds.), *Recent advances on hybrid intelligent systems* (pp. 349-359). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33021-6_28

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*. Wiley Publishing.

Tan, B., Ma, H. & Mei, Y. (2020). A group genetic algorithm for resource allocation in container-based clouds. En L. Paquete & C. Zarges (Eds.), *Evolutionary computation in combinatorial optimization* (pp. 180-196). Springer International Publishing.

Wiischer, G. & Gau, T. (1996). *Orspe m heuristics for the integer one-dimensional cutting stock problem: A computational study.*

Wilcox, D., McNabb, A. & Seppi, K. (2011). Solving virtual machine packing with a reordering grouping genetic algorithm. *2011 IEEE Congress of Evolutionary Computation (CEC)*, 362-369. https://doi.org/10.1109/CEC.2011.5949641