AUTORIZACIÓN DE USO DE DERECHOS DE AUTOR OTORGADO POR

JOSE FERNANDO PADRÓN TRISTÁN, mayor de edad, con domicilio ubicado en **CALLE 5 DE MAYO # 900 SUR, COL 1 DE MAYO, CIUDAD MADERO, TAMPS.**, en mi calidad de titular y autor de la tesis denominada **ALGORITMO DE VIRTUAL SAVANT BASADO EN LOGICA DIFUSA COMPENSATORIA PARA PROBLEMAS DE EMPACADO DE OBJETOS** quien para todos los fines del presente documento se denominará **EL AUTOR Y/O TITULAR**, suscribo el presente documento de autorización de uso de derechos patrimoniales de autor a favor del Instituto Tecnológico de Ciudad Madero el cual se regirá por clausulas siguientes:

PRIMERA – AUTORIZACIÓN: EL AUTOR Y/O TITULAR, mediante el presente documento autoriza la utilización de los derechos patrimoniales de autor al Instituto Tecnológico de Ciudad Madero, de la tesis denominada **ALGORITMO DE VIRTUAL SAVANT BASADO EN LOGICA DIFUSA COMPENSATORIA PARA PROBLEMAS DE EMPACADO DE OBJETOS**, a través del Repositorio Institucional del Tecnológico Nacional de México (en lo sucesivo TecNM) y en el Repositorio Nacional, que puede ser consultado en la liga electrónica: (https://www.repositorionacionalcti.mx/).

SEGUNDA - OBJETO: Por medio del presente escrito, **EL AUTOR Y/O TITULAR** Autoriza al Instituto Tecnológico de Ciudad Madero, a través del Repositorio Institucional del Tecnológico Nacional de México (en lo sucesivo TecNM) y en el Repositorio Nacional para que de conformidad con la Ley Federal del Derecho de Autor y la Ley de la Propiedad Industrial, use los derechos del documento antes referido, con fines exclusivamente académicos.

TERCERA - TERRITORIO: Los derechos aquí Autorizados se dan sin limitación geográfica o territorial alguna.

CUARTA – ALCANCE: La presente autorización se da tanto para formato o soporte material, y se extiende a la utilización en medio óptico, magnético, electrónico, en red, mensajes de datos o similar conocido o por conocer, del ejemplar o número respectivo de la publicación.

QUINTA – EXCLUSIVIDAD: La autorización de uso aquí establecida no implica exclusividad en favor del Instituto Tecnológico de Ciudad Madero. Por lo tanto **EL AUTOR Y/O TITULAR** en su carácter de autor de la obra objeto del presente documento se reserva el derecho de publicar directamente, u otorgar a cualquier tercero, autorizaciones de uso similares o en los mismos términos aquí acordados.

SEXTA - DERECHOS MORALES (Créditos y mención): La Autorización de los derechos antes mencionados no implica la cesión de los derechos morales sobre los mismos por cuanto en conformidad con lo establecido en los artículos 18, 19, 20, 21, 22 y 23 de la Ley Federal de Derechos de Autor, dada la cuenta que estos derechos son inalienables, imprescriptibles, irrenunciables e inembargables. Por lo tanto, los mencionados derechos seguirán radicados en cabeza de EL AUTOR Y/O TITULAR, y siempre deberá mencionarse su nombre cuando se utilice la obra.

SÉPTIMA - AUTORIA: EL AUTOR Y/O TITULAR, declara y ratifica que el material objeto de la presente y fue realizada por él (o ella) sin violar o usurpar derechos de Propiedad Intelectual de terceros.

Ciudad Madero, Tamps. a los 3 del mes de junio de 2020.

Autor de la Tesis

Ing. José Fernando Padrón Tristán
CURP PATF760818HTSDRR00

Asesor de la Tesis

Dra. Laura Cruz Reyes
CURP CURL590403MTSRYR02

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MAESTRÍA EN CIENCIAS COMPUTACIONALES

TESIS

ALGORITMO DE VIRTUAL SAVANT BASADO EN LÓGICA DIFUSA COMPENSATORIA PARA PROBLEMAS DE EMPACADO DE OBJETOS

Que para obtener el grado de
Maestro en Ciencias Computacionales
Presenta
Ing. José Fernando Padrón Tristán
G96071173

Director de Tesis
Dra. Laura Cruz Reyes
Co-director de Tesis
Dr. Bernabé Dorronsoro

Ciudad Madero, Tamaulipas — Junio 2020

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Ciudad Madero
División de Estudios de Posgrado e Investigación

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Cd. Madero, Tams., a 22 de Mayo de 2020

OFICIO No.: U.024/20
ÁREA: DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

ING. JOSÉ FERNANDO PADRÓN TRISTAN
No. DE CONTROL G96071173
P R E S E N T E

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

"ALGORITMO DE VIRTUAL SAVANT BASADO EN LÓGICA DIFUSA COMPENSATORIA PARA PROBLEMAS DE EMPACADO DE OBJETOS"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTE :	DR.	NELSON RANGEL VALDEZ
SECRETARIO:	DR.	HÉCTOR JOAQUÍN FRAIRE HUACUJA
VOCAL:	DRA.	LAURA CRUZ REYES
SUPLENTE:	DR.	JUAN FRAUSTRO SOLÍS
DIRECTORA DE TESIS :	DRA.	LAURA CRUZ REYES
CO-DIRECTOR DE TESIS:	DR.	BERNABE DORRONSORO

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

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Por mi patria y por mi bien"®

DR. JOSÉ AARÓN MELO BANDA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p.- Archivo

JAMB 'MCC 'COB'

Contenido

Índice de figuras

Índice de tablas

Resumen

Existen problemas de optimización que no son fáciles de resolver, por lo que se categorizan como NP-duro. Para la solución de estos problemas se han utilizado métodos exactos, que son de alta complejidad algorítmica. Por otro lado, existen los métodos heurísticos que obtienen soluciones aproximadas en tiempo de cómputo razonable, pero no garantizan encontrar la solución óptima. Debido a sus limitaciones, ambos tipos algoritmos están en constante evolución.

En esta tesis se presenta un nuevo método de solución heurística del problema de empacado de objetos en contenedores (BPP, por su sigla en inglés) que es NP-duro. El método propuesto está basado en el paradigma Virtual Savant cuyo objetivo es inferir el comportamiento de un algoritmo mediante aprendizaje automático, para reproducirlo en arquitecturas paralelas. El método propuesto incorpora nuevas estrategias de transformación, clasificación y refinación.

La estrategia de transformación consiste en obtener instancias de clasificación a partir de instancias de optimización resueltas de BPP. En la mejor alternativa de transformación propuesta, un par de objetos pertenece a la clase 1 si estos están empacados en el mismo contenedor, en caso contrario pertenece a la clase 0. Además de la clase, se toma en cuenta los pesos de los objetos y la capacidad residual causada por el primer objeto.

La clasificación está basada en lógica difusa compensatoria y para nuevas instancias no resueltas de BPP se producen probabilidades de asignación a las clases. Usando el vector de probabilidades de clase se construyen soluciones. Estas soluciones se refinan con una heurística propuestas para mejorar las soluciones en corto tiempo y finalmente seleccionar la mejor de acuerdo a su valor de aptitud.

La ventaja de la propuesta de solución de BPP que se presentan en este trabajo fue respaldada por un conjunto de experimentos que consideran calidad y desempeño. Además de ofrecer la posibilidad de resolver otros problemas de agrupación como lo es BPP.

Abstract

There are optimization problems that are not easy to solve, so they are categorized as NP-hard. Exact methods have been used to solve these problems, which are of high algorithmic complexity. On the other hand, there are heuristic methods that require approximate solutions in reasonable computation time, but do not find the optimal solution. Due to their limitations, both types of algorithms are constantly evolving.

In this thesis, a new method of heuristic solution of the problem of packaging objects in containers (BPP) is presented, which is NP-hard. The proposed method is based on the Virtual Savant paradigm whose objective is to infer the behavior of an algorithm through machine learning, to reproduce it in parallel architectures. The proposed method incorporates new transformation, classification to refine strategies.

The transformation strategy consists in obtaining classification instances from solved optimization instances of BPP. In the best transformation alternative proposed, a pair of objects belongs to class 1 if they are packed in the same container; otherwise, it belongs to class 0. In addition to the class, it takes into account the weights of the objects and the residual capacity caused by the first object.

The classification is based on fuzzy compensatory logic and for new unresolved instances of BPP the class assignment factors occur. Using the vector of class probabilities, solutions are constructed. These solutions are refined with heuristic proposals to improve the solutions in a short time and finally select the best one according to its fitness value.

The advantage of the BPP solution proposal presented in this work was supported by a set of experiments that determines quality and performance. Besides offering the possibility of solving other grouping problems such as BPP.

Agradecimientos

A mi esposa, Janice Astrid, por su apoyo incondicional y su paciencia, no sólo en este proyecto sino en todas las metas que nos proponemos, es mi motivación.

A mi madre, María de Lourdes, quien me ha dado un ejemplo de dedicación y de valores.

A mis hermanos, Juan Carlos, Mary y Fátima.

A mis directores y comité de tesis, los doctores Laura Cruz Reyes, Bernabé Dorronsoro, Héctor Fraire, Nelson Rangel y Juan Frausto por su guía y apoyo en este proyecto.

Y sobre todo, gracias a Dios, por permitirme terminar esta maestría, para poder ir por la siguiente meta.

Capítulo 1

1. Introducción

Un nuevo escenario que está tomando mucha importancia para mejorar la gestión tradicional de las organizaciones, y de la sociedad en general, es el cómputo en la nube. En este contexto, un aspecto clave es encontrar estrategias de planeación que permitan utilizar los recursos de manera eficiente.

Entre estas estrategias destaca la colocación de recursos en la nube que puede ser modelada como un problema de acomodo de objetos (Gao, 2013). El problema clásico de acomodo de objetos en contenedores (Bin Packing Problem, BPP) pertenece a la clase NP-duro, por lo que se considera irresoluble en tiempo polinomial (Békési, 2000). Para los problemas de esta clase, la búsqueda de algoritmos eficientes es un área de constante evolución.

Para resolver problemas de optimización complejos existe en la literatura una gran cantidad de modelos de optimización genéricos. El uso de modelos genéricos implica que el desarrollador debe dominar el problema y las nuevas arquitecturas de cómputo paralelo; de esta manera podría obtener soluciones que satisfagan las necesidades de los consumidores de tecnologías de información disponibles en la nube (Keckler, 2009; Koziel, S, 2011).

Existen muchos esfuerzos de la comunidad científica para la generación automática de programas, en donde se busca que las computadoras pueden aprender por sí mismas las tareas requeridas para producir los resultados deseados, sin intervención de un experto programador.

Con este fin, un trabajo reciente propone el Virtual Savant (VS), que utiliza técnicas de aprendizaje automático para inferir el comportamiento de un algoritmo dado, y reproducirlo en arquitecturas paralelas (Pinel, 2014; Pinel, 2018). VS ha sido aplicado exitosamente en la solución de problemas de calendarización de tareas.

En este trabajo se busca extender VS adicionando lógica difusa compensatoria (Espin, 2016), así como estrategias específicas para problemas de agrupación como lo es BPP, así como estrategias específicas para problemas de agrupación como es BPP con la finalidad de potenciar el proceso de inferencia. Para validar la calidad de la propuesta, se usará como caso de estudio instancias retadoras de BPP y algoritmos del estado del arte que lo resuelven (Quiroz, 2015).

1.1. Antecedentes

Este trabajo forma parte de un proyecto mayor que realizan investigadores de tres universidades: la universidad de Málaga, el Instituto Tecnológico de Ciudad Madero y la Universidad autónoma de Coahuila.

El trabajo conjunto ha permitido obtener importantes productos de investigación en las áreas de cómputo paralelo e inteligente y en sus aplicaciones a la logística y a la toma de decisiones.

Se espera que el proyecto de tesis contribuya a las áreas mencionadas, particularmente al desarrollo de algoritmos metaheurísticos paralelos basados en aprendizaje con lógica difusa.

1.2. Justificación

La contribución de la aplicación va encaminada a la automatización de procesos de optimización, como la selección de algoritmos, la configuración o el diseño de nuevas heurísticas para problemas nuevos.

El trabajo del Dr. Dorronsoro (Pinel, 2014) es el primero, hasta nuestro conocimiento, que aborda la optimización automática con técnicas de aprendizaje automático para aprender el proceso de generar soluciones mediante un nuevo paradigma que denominó Virtual Savant.

El trabajo propuesto en esta tesis aborda el problema de empacado de objetos, que no ha sido visto por el VS. Así mismo, se busca extender las capacidades del generador automático, al agregarle características de interpretabilidad a las soluciones generadas.

1.3 Hipótesis

Para el problema de empacado de objetos de una dimensión se plantean las siguientes cuestiones:

¿Es posible superar el rendimiento del algoritmo genético GGA-CGT, con una extensión del paradigma Virtual Savant que use la lógica difusa compensatoria como clasificador, sustituyendo a la máquina de soporte vectorial que actualmente usa?

¿Qué procesos de Virtual Savant deben ser modificados para su aplicación al problema de empacado?

1.4. Objetivos

1.4.1 Objetivo general

Desarrollar un método que genere optimizadores paralelos que resuelvan problemas de empacado usando técnicas de aprendizaje automático para modelar la capacidad de solución de un algoritmo de referencia visto como caja negra.

1.4.2 Objetivos específicos

Para lograr el objetivo general de este proyecto, se han planteado los siguientes objetivos específicos:

1) Evaluar la influencia de diferentes clasificadores.
2) Usar un clasificador basado en lógica difusa compensatoria y comparar contra el clasificador usado actualmente por el Virtual Savant: la máquina de soporte vectorial.

3) Evaluar ventajas y desventajas de la lógica difusa compensatoria como clasificador.
4) Desarrollar una estrategia de solución de problemas NP-duros limitados por las capacidades del hardware para problemas de agrupamiento.
5) Analizar de manera teórica y práctica el método que genera optimizadores paralelos VS que se basa en máquinas de soporte vectorial.
6) Adaptar un módulo de lógica difusa compensatoria como clasificador para Virtual Savant.
7) Adaptar heurísticas de búsqueda local del problema de empacado de objetos para su incorporación en Virtual Savant.
8) Comparar experimentalmente la propuesta basada en lógica difusa compensatoria contra la versión original de Virtual Savant.
9) Aplicar la metodología propuesta en instancias no vistas y de mayor tamaño o en instancias difíciles de resolver.

1.5. Alcances y limitaciones

1.5.1 Alcances

1) Experimentación con VS para el problema de la mochila para mostrar que se llegan a los resultados obtenidos en la literatura.
2) Incorporación de LDC a las fases de entrenamiento y predicción del VS.
3) Experimentación con instancias de BPP para el VS con LDC incorporada.

1.5.2 Limitaciones

1) Para el problema de la mochila, sólo se resolverán instancias reportadas en la literatura, específicamente conjuntos de datos del problema de la siguiente liberación de software (Next Release Problem, NRP, Harman, 2014).
2) Sólo se resolverán instancias de BPP de una dimensión reportadas en la literatura, en especial, instancias que son muy difíciles de resolver reportadas por Quiroz (2019).

1.6 Problema de investigación

Este trabajo resuelve el problema de empacado de objetos, el cual es de alta complejidad algorítmica ya que pertenece a la clase NP-duro (Falkenauer, 1996), por lo que requiere de una alta cantidad de recursos computacionales para su solución.

Se parte del supuesto que es posible llegar a la solución óptima de este problema a través del Virtual Savant (Pinel, 2014, Massobrio, 2018), que es un paradigma que ha sido usado para resolver el problema de la mochila y el de asignación de tareas.

El Virtual Savant aprende el comportamiento de un algoritmo de referencia para en la ejecución tratar de encontrar la solución óptima en instancias no vistas. Durante el aprendizaje utiliza instancias pequeñas relacionando el valor de las variables de cada observación en el conjunto de datos con la respectiva clase a la que pertenece. El resultado del proceso de aprendizaje es un modelo de clasificación que en la operación se ejecuta de forma paralela para encontrar la solución con instancias más grandes.

La ejecución del Virtual Savant se realiza en dos pasos: predicción y refinamiento de soluciones. En el primer paso se genera una población de soluciones de acuerdo a las probabilidades calculadas por el modelo de clasificación. En el segundo paso se ejecuta un operador de refinamiento en las soluciones, para de acuerdo a los criterios del problema seleccionar la mejor solución como la salida del Virtual Savant.

En el grupo de trabajo al que se pertenece, se han generado nuevas instancias muy difíciles de resolver. En un estudio preliminar se encontró que el algoritmo del estado del arte GGA-CGT tarda hasta días en encontrar una solución que no es garantizada como la óptima. Son estos conjuntos de datos que serán tomados como entrenamiento y prueba del presente proyecto de tesis.

1.7 Organización del documento

El documento prosigue como se muestra a continuación:

El capítulo 2 es el marco teórico, en el cual se reseñan los problemas de investigación a tratar, los problemas de la mochila y el de empacado de objetos, el paradigma Virtual Savant, la programación paralela, la lógica difusa compensatoria, entre otros.

El capítulo 3 muestra el estado del arte, que incluye el algoritmo GGA-CGT que será utilizado como algoritmo de referencia.

El capítulo 4 detalla la metodología de solución utilizada.

El capítulo 5 muestra la experimentación realizada y los resultados obtenidos comparando el algoritmo de referencia con el Virtual Savant utilizando tanto la máquina de soporte vectorial como la lógica difusa compensatoria como clasificadores.

El capítulo 6 muestra las conclusiones, las aportaciones y el trabajo futuro de este proyecto de tesis.

Capítulo 2

2. Marco teórico

La optimización consiste en la selección de la mejor alternativa, en algún sentido, busca minimizar o maximizar el valor de una variable o el valor máximo o mínimo de una función. Los problemas de optimización constan de función objetivo, variables y restricciones y resolverlos consiste en encontrar los valores que deben tomar las variables para hacer óptima la función objetivo satisfaciendo el conjunto de restricciones (Ramos, 2010).

2.1 Problema de la mochila

El problema de la mochila (KP) puede ser definido con un conjunto de *n* artículos donde cada artículo es identificado por x_i, con un valor entero p_j, y un peso w_j. El problema consiste en elegir un subconjunto de n artículos maximizando el beneficio obtenido considerando el peso total de los artículos seleccionados, sin exceder la capacidad c de la mochila (Bruno, 2013), como se muestra en la figura 2.1.

Una definición del problema se presenta a continuación: “Se tiene una determinada instancia de KP con un conjunto de objetos N, que consiste de *n* objetos *j* con ganancia *p* y peso *w*, y una capacidad c. (Usualmente, los valores toman números enteros positivos). El objetivo es seleccionar un subconjunto de N tal que la ganancia total de esos objetos seleccionados es maximizado y el total de los pesos no excede a *c*”.

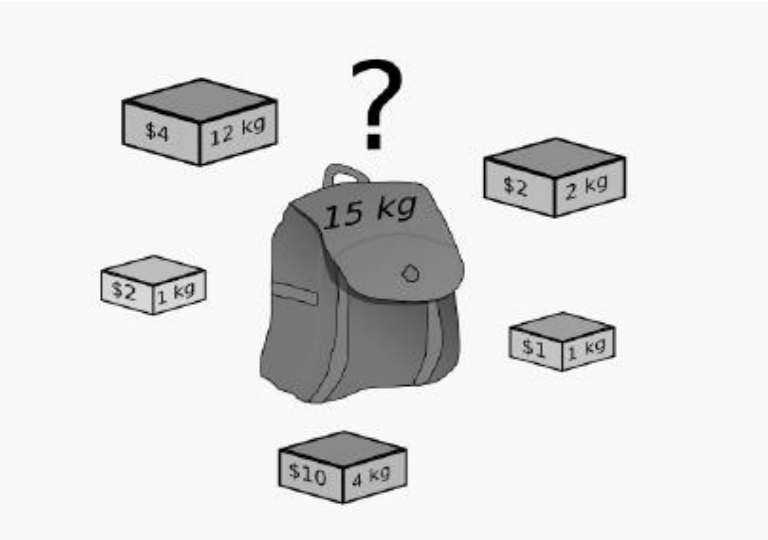

Figura 2.1 Problema de la mochila (Massobrio, 2018).

El modelo básico del KP es el siguiente:

$$\boldsymbol{maximizar\ z = \sum_{j=1}^{n} p_j x_j} \qquad (2.1)$$

$$sujeto\ a\ \sum_{j=1}^{n} w_j x_j \leq c$$

$$con\ x_j \in \{0,1\}\ j = 1, \ldots, n$$

Donde:

x_j son variables de decisión

w_j es el peso *w* del elemento *j*

c es la capacidad total del contenedor (mochila)

n es el número de elementos

El modelo se puede expresar como:

1. maximizar los resultados del proyecto a partir de la integración de múltiples variables que pertenecen al proyecto actual identificado por el subíndice *j*,
2. siendo necesario estimar el peso total de los artículos que serán guardados en un contenedor cuya capacidad es determinada por la variable *c*.

2.2 Bin Packing Problem

El problema de empacado de objetos en contenedores de una dimensión, (one-dimensional Bin Packing Problem), de aquí en adelante llamado BPP, consiste en que, dado un número ilimitado de contenedores, también llamados bins, con una capacidad fija $c > 0$ y un conjunto de n elementos, con un peso específico $0 < w_i \leq c$, consiste en almacenar el total de objetos en el menor número de contenedores sin sobrepasar la capacidad de ningún contenedor (figura 2.2).

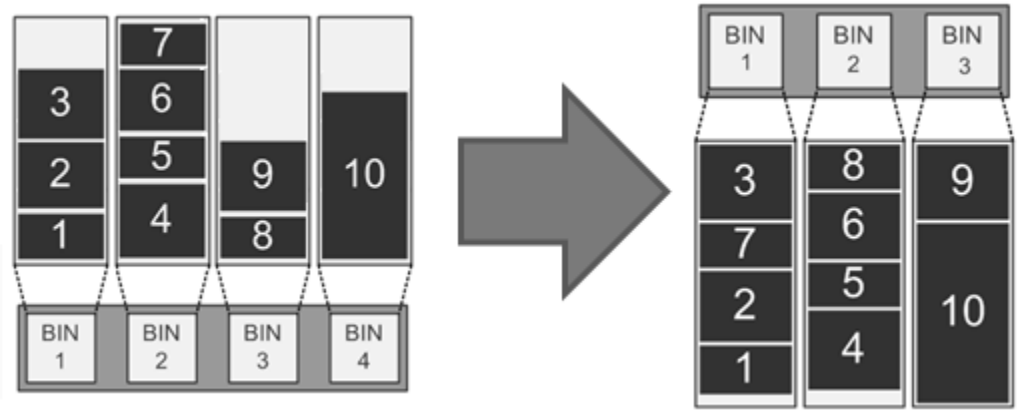

Figura 2.2 Ejemplo de empacado de objetos (Jankovic 2013).

Éste es un problema clásico de optimización combinatoria NP-duro, considerado intratable pues demanda una gran cantidad de recursos para su solución (Quiroz, 2014).

Dado un conjunto $N = \{1,\ldots, n\}$ de objetos a distribuir en contenedores de la misma capacidad.

BPP consiste en asignar cada objeto a un contenedor de tal forma que la suma de los pesos de los objetos en cada contenedor no exceda c y el número de contenedores m utilizado sea mínimo (Martello, 1990). Es decir, se busca encontrar el menor número de subconjuntos m para una partición del conjunto N.

$$\boldsymbol{minimizar\ m}, \bigcup_{\boldsymbol{j=1}}^{\boldsymbol{m}} \boldsymbol{B_j} = \boldsymbol{N} \qquad (2.2)$$

$$sujeto\ a \sum_{\forall i \in B_j}^{n} w_i \leq c \quad 1 \leq j \leq m, i \in N = \{1, \ldots, n\}$$

Donde:

c la capacidad de cada contenedor
w_i el peso del objeto i, tal que $0 < w_i \leq c$ para i=1 hasta n

El BPP es un problema en constante evolución, donde aparecen nuevas instancias y se mantiene vigente debido a que se utiliza en logística, cómputo en la nube, entre otras actividades.

El mejor acomodo de objetos en contenedores supone un ahorro de recursos o costos. En logística, el utilizar una menor cantidad de contenedores o bins, se traduce, por ejemplo, en transporte en el empleo de menos vehículos reduciendo además recursos humanos y costos como el combustible, entre otros.

Vector BPP

Para el cómputo en la nube el vector BPP (Abdel-Basset et al., 2019) es un ejemplo del uso de este modelo, en el cual se trata de usar la menor cantidad de equipos en un clúster para la asignación de trabajos o Jobs de manera estática.

Un clúster está conformado por equipos que tienen diferentes características, entre las cuales se encuentran la velocidad del procesador, la memoria RAM, la velocidad de la tarjeta de red y la capacidad en disco duro.

De estas características se consideran restricciones suaves el procesador, RAM y tarjeta de red, considerando como única restricción dura la capacidad del disco duro permitiendo modelar el problema como BPP.

El acomodo óptimo de los trabajos en los equipos del clúster permite un menor número de dispositivos utilizados, apagando los que no son empleados traduciéndose en un ahorro de energía.

2.3 Algoritmos exactos y aproximados

Guerrero-Treviño (2007) menciona que para la resolución de problemas de optimización se puede hacer uso de algoritmos exactos o algoritmos aproximados.

Los algoritmos exactos garantizan obtener la solución óptima de un problema en estudio ya que se basan en generar todas las soluciones que cumplen con todas las restricciones, llamadas soluciones factibles, calculando su valor de aptitud, es decir, el costo ocasionado por cada solución y seleccionando de ellas la mejor.

El inconveniente de este tipo de algoritmos es que, al resolver un problema con instancias grandes, puede resultar impráctico evaluar todas las soluciones generadas ya que implica un gran consumo de recursos, ya sea la memoria utilizada o el tiempo necesario de procesamiento.

Por esta razón los algoritmos exactos más populares son los de enumeración parcial, ya que reducen el espacio de búsqueda en el espacio de soluciones, sin embargo, el consumo de recursos continúa siendo alto.

Por otra parte, los algoritmos aproximados son una buena opción cuando se desea resolver casos de instancias grandes de un problema de optimización que es considerado NP-duro. Estos algoritmos generan soluciones aproximadas sin el costo excesivo de recursos como tiempo o memoria, pero sin la garantía de obtener la solución óptima.

Además, otra limitación de los algoritmos aproximados es que no es posible determinar qué tan cerca se queda la solución encontrada de la óptima.

Díaz et al (1996) señalan otras situaciones además de la mencionada donde se pueden aplicar algoritmos aproximados:

- no existe un algoritmo exacto para resolver el problema,
- no se requiere una solución óptima,
- los datos con los que se cuenta no son fiables,
- se requiere una rápida respuesta, a costa de la precisión y,
- las soluciones generadas son punto de partida de algún otro algoritmo.

2.4 Algoritmos heurísticos de propósito específico o deterministas

Guerrero-Treviño (2007) describe que los algoritmos heurísticos de propósito específico o deterministas tienen la característica de obtener la misma solución en diferentes ejecuciones.

Un análisis teórico se presenta en Coffman et al (1997) con los algoritmos para acomodo de objetos de acuerdo al primer, mejor y peor ajuste.

Algoritmo primer ajuste

Como su nombre lo indica, en el algoritmo primer ajuste (FF, first fit) la regla a seguir es colocar el objeto actual en el primer contenedor que lo pueda almacenar, es decir, el de más bajo índice tomando en cuenta el peso del objeto y la capacidad del contenedor. Inicialmente, los contenedores se consideran parcialmente llenos.

Algoritmo peor ajuste

Para el algoritmo peor ajuste (WF, worst fit), cada objeto será empacado en el contenedor con más bajo nivel de almacenaje, en el caso de que no pueda ser colocado en ninguno de los contenedores por rebasar su capacidad, entonces será almacenado en un nuevo contenedor

Algoritmo mejor ajuste

En el algoritmo mejor ajuste (BF, best fit), cada objeto, de acuerdo a su peso, será empacado en el contenedor con mayor nivel de llenado que lo pueda almacenar, en caso de que no pueda ser empacado, se colocará en un nuevo contenedor.

Para los empacados WF y BF, en caso de que un objeto pueda almacenarse en más de un contenedor, entonces se elegirá el de más bajo índice.

Versiones de algoritmos de ajuste decrecientes

Para los tres algoritmos (FF, WF y BF) existen las versiones decrecientes (FFD, WFD y BFD), donde inicialmente, el conjunto de objetos a empacar es ordenado de acuerdo a sus pesos de mayor a menor, permitiendo almacenar primero los objetos más grandes los cuales son más difíciles de acomodar.

Reacomodo por Pares

El reacomodo por pares (RPP) se compone de dos etapas: primero, se recorre cada contenedor en un intento de mejorar su llenado haciendo intercambios entre pares de

objetos empacados y objeto libres; segundo, los objetos libres se introducen en la solución utilizando la heurística FF (Quiroz, 2014).

2.5 Algoritmos meta heurísticos

Los algoritmos meta heurísticos son algoritmos aproximados de optimización y búsqueda de propósito general.

Son procedimientos iterativos que guían una heurística subordinada combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda (Herrera, 2006).

Algoritmo GGA-CGT

El Algoritmo Genético de Agrupación con Transmisión Genética Controlada (GGA-CGT, Quiroz, 2014) para el BPP promueve la transmisión de los mejores genes en los cromosomas sin perder el equilibrio entre la presión selectiva y la diversidad de la población.

La transmisión de los mejores genes se logra mediante un nuevo conjunto de operadores genéticos de agrupación, mientras que la evolución se equilibra con una nueva técnica de reproducción que controla la exploración del espacio de búsqueda y evita la convergencia prematura del algoritmo.

2.6 Programación paralela

La velocidad de los computadores secuenciales convencionales se ha incrementado continuamente para adaptarse a las necesidades de las aplicaciones. Además, el rendimiento de los computadores secuenciales está comenzando a saturarse, siendo limitados por memoria disponible o procesadores (Barney, 2018).

Algunos problemas conllevan costosos cálculos iterativos sobre grandes cantidades de datos con fuertes restricciones temporales: Además son sistemas cada vez más complejos que requieren mayor tiempo de cómputo, donde:

La solución es usar varios procesadores y la forma en la que se pueden aprovechar estos recursos es mediante sistemas paralelos.

En el sentido más simple, la computación paralela es el uso simultáneo de múltiples recursos informáticos para resolver un problema computacional donde:

1. Un problema se divide en partes discretas que pueden resolverse simultáneamente.
2. Cada parte se desglosa en una serie de instrucciones.
3. Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores.
4. Se emplea un mecanismo general de control / coordinación.

Las técnicas de programación paralela aplican estrategias de descomposición o particionamiento de datos y de cómputo, para dividir un problema en subproblemas de menor complejidad (Nesmachnow, 2010).

El objetivo primario de la descomposición será dividir en forma equitativa tanto los cálculos asociados con el problema como los datos sobre los cuales opera el algoritmo.

Según se enfoque principalmente en la descomposición de datos o de tareas, resulta una técnica diferente de programación paralela.

En el paradigma MapReduce, basada en descomposición de datos, la función map divide el conjunto de datos en partes que contienen una clave de identificación y colecciones de registros. Cada porción de datos es procesada por separado para obtener valores intermedios. La función reduce toma la salida del map y agrupa los resultados de acuerdo a su clave. Posteriormente aplica un procesamiento final a los resultados reducidos (Hernández & Hernández, 2015).

En implementaciones de este paradigma (Dean, 2008), por empresas que se dedican al desarrollo de software como google, los usuarios especifican el cálculo en términos de un mapa y una función de reducción, y el sistema de tiempo de ejecución subyacente paraleliza automáticamente el cálculo a través de grupos de máquinas a gran escala, maneja fallas de la máquina y programa la comunicación entre máquinas para hacer un uso eficiente de la red y los discos.

Muchos programadores encuentran que el framework MapReduce de google es fácil de usar: más de diez mil programas MapReduce distintos se han implementado internamente en Google en un periodo de cuatro años de 2004 a 2008, y un promedio de cien mil trabajos MapReduce se ejecutan en los clústeres de Google todos los días, procesando un total de más de veinte peta bytes de datos por día.

2.7 Minería de datos

El concepto de minería de datos (DM, Data Mining) está aplicado a la extracción de patrones o características útiles de un conjunto de datos. DM es un proceso que consiste en la aplicación del análisis de datos y algoritmos de descubrimiento que producen una particular enumeración de patrones (o modelos) sobre los datos (Mannila, 1996).

Otra definición de DM es "el proceso iterativo de extraer patrones predictivos que están escondidos en bases de datos grandes, usando técnicas tanto tecnológicas como estadísticas" (Mena, 1999). Esta definición involucra a la estadística y el aprendizaje automático, como una rama de la inteligencia artificial (IA) en las cuales está sustentada la DM (Aluja, 2001).

DM hace referencia a la minería, que es la actividad de extracción de minerales que yacen en el suelo o subsuelo, tales como el oro, cobre, plata, entre otros, comparando estos minerales con los datos y la minería con las técnicas utilizadas en DM.

DM es una etapa del descubrimiento de conocimiento en bases de datos (KDD, Knowledge Discovery in Databases), el cual abarca todo el proceso de descubrimiento de conocimiento, en el que además se encuentran las etapas previas de preparación, selección y limpieza de los datos y la posterior interpretación de los resultados (Riquelme, 2006).

Mediante DM se pueden realizar tareas descriptivas o prescriptivas, con las cuales se descubren patrones interesantes en los datos y, se clasifican nuevos datos basándose en la información previa, respectivamente (Riquelme, 2006).

Entre otras, DM se compone de las siguientes funciones: regresión, clasificación, clustering, agrupamiento, extracción de reglas.

2.7.1 Clasificación

La clasificación es el proceso de asignar una categoría o clase a un objeto o elemento de acuerdo a sus características o propiedades (Gorunescu, 2011), agrupa todas las herramientas que permiten establecer esta pertenencia a la clase correspondiente (figura 2.3).

Este proceso es logrado analizando un conjunto de elementos de los cuales se conoce la clase a la que pertenecen, descubriendo reglas que toman en cuenta el valor de ciertas variables, que serán utilizadas para discriminar cada elemento y relacionarlo con su respectiva clase. Las reglas descubiertas sirven para evaluar las características de nuevos elementos prediciendo la categoría a la que pertenecen (Martínez, 2001).

La clasificación utiliza herramientas tales como algoritmos matemáticos, análisis discriminantes, sistemas de conocimientos, sistemas expertos e inducción de reglas, entre otros (Martínez, 2001).

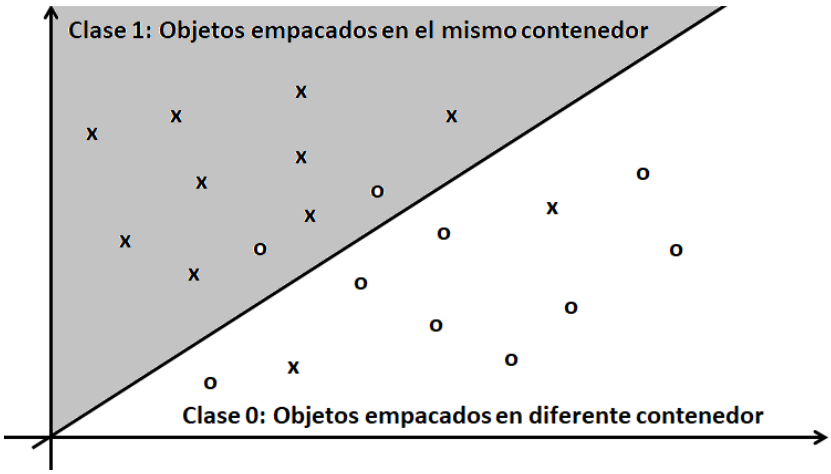

Figura 2.3. Ejemplo de clasificación de datos.

2.7.2 Máquinas de soporte vectorial

La máquina de soporte vectorial (support vector machine, SVM) es una técnica que utiliza algoritmos de aprendizaje automático y es utilizada propiamente para problemas de clasificación y regresión (Hsu, 2016).

SVM Encuentra el hiperplano que separe con el mayor margen las clases incluidas en los conjuntos de datos, donde (figura 2.4):

a) El parámetro coste (C) penaliza los errores de clasificación, a mayor valor de C, permite menor cantidad de errores o vectores de soporte
b) La función kernel replantea el problema de tal forma que los datos se mapean a un espacio de dimensión mayor.
c) Gama (γ) puede considerarse como la "propagación" del kernel (Albon, 2017).

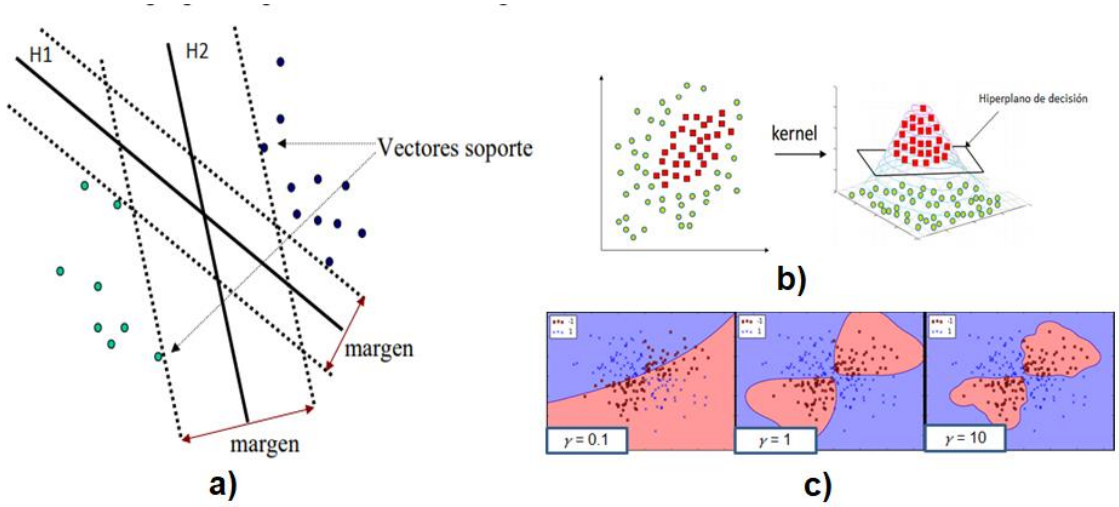

Figura 2.4 Funcionamiento SVM (Hsu 2003, Albon, 2017)

2.8 Virtual Savant

2.8.1 El síndrome de Savant

(Pinel, 2014) La propuesta del Virtual Savant trata de emular el cerebro humano, el cual es capaz de resolver problemas secuenciales en poco tiempo, asegurando que el método empleado se basa en el paralelismo, para adaptarse a la tendencia actual de las arquitecturas de computación. Las características esenciales del síndrome de Savant se traducen en un enfoque algorítmico general para la paralelización automática.

Las personas que presentan síntomas del síndrome de Savant (o síndrome del sabio) pueden calcular pequeñas tareas secuenciales, como el cálculo del calendario (dada una fecha, radica en encontrar qué día de la semana es), casi instantáneamente y que el aumento

del tiempo de respuesta de los sabios para proporcionar una solución con respecto al tamaño del problema sea inferior al de cualquier algoritmo conocido.

Aunque no se entiende completamente, se cree que los sabios extraen reglas que capturan las regularidades percibidas en los datos, como las que se encuentran en los calendarios y números primos, y luego aplican estos patrones (en paralelo) en los datos de entrada de problemas, para resolver sus problemas. Las descripciones de la percepción interna de los números de Savant muestran sinestesia.

Estos hallazgos podrían ayudar a explicar cómo pueden realizar cálculos complejos de calendario y enumeración de números primos, sin siquiera conocer los detalles complicados de los calendarios o incluso qué son los números primos.

2.8.2 El framework de virtual Savant

Basados en el síndrome de Savant, Virtual Savant (VS) (Pinel, 2014, 2018), es capaz de generar automáticamente un programa paralelo masivo a partir de un conjunto de observaciones, que no requieren ningún tipo de código fuente como entrada.

Para eso, VS confía en el aprendizaje automático para aprender el comportamiento de un algoritmo de referencia dado para resolver algún problema de optimización, utilizando como observaciones varias instancias de problemas y sus soluciones, proporcionadas por el algoritmo de referencia (como se muestra en el paso de aprendizaje en la Figura 2.5.

Por lo tanto, dado un algoritmo, VS puede generar automáticamente un programa nuevo y completamente diferente que reproduce su comportamiento, pero de una manera masivamente paralela y altamente eficiente, gracias a su motor de reconocimiento de patrones paralelos. En la Figura 2.5 se muestra una vista general de la plantilla de VS. Básicamente, VS se compone de dos pasos.

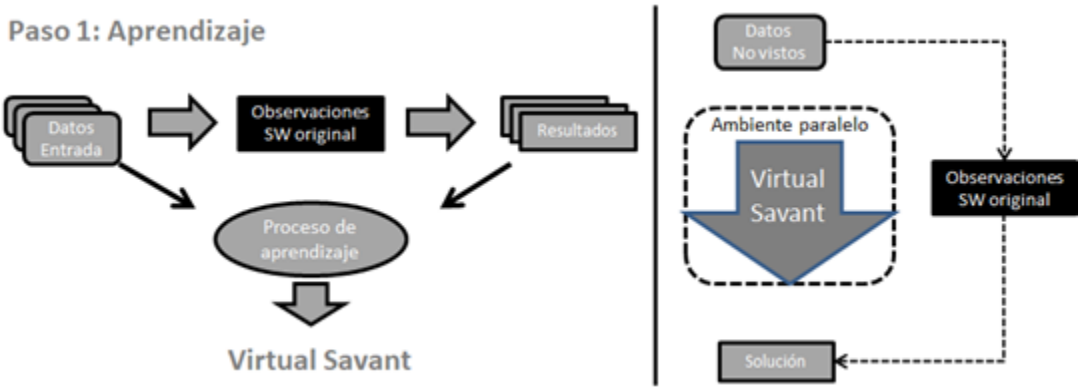

Figura 2.5. Aprendizaje y ejecución de VS (Massobrio, 2018).

El primero es el paso de predicción. Por analogía con el síndrome de Savant, VS implementa un predictor paralelo para generar una solución, basado en reconocimientos de patrones paralelos.

Cada predictor está compuesto por un motor de reconocimiento de patrones para la asignación de un valor a la variable correspondiente (representada en la figura 2.6 como cuadros grises, etiquetados como "P"). Solo requieren como entrada la información relevante para tomar la decisión sobre una variable, y no toda la instancia del problema (por lo tanto, la instancia del problema debe estar particionada).

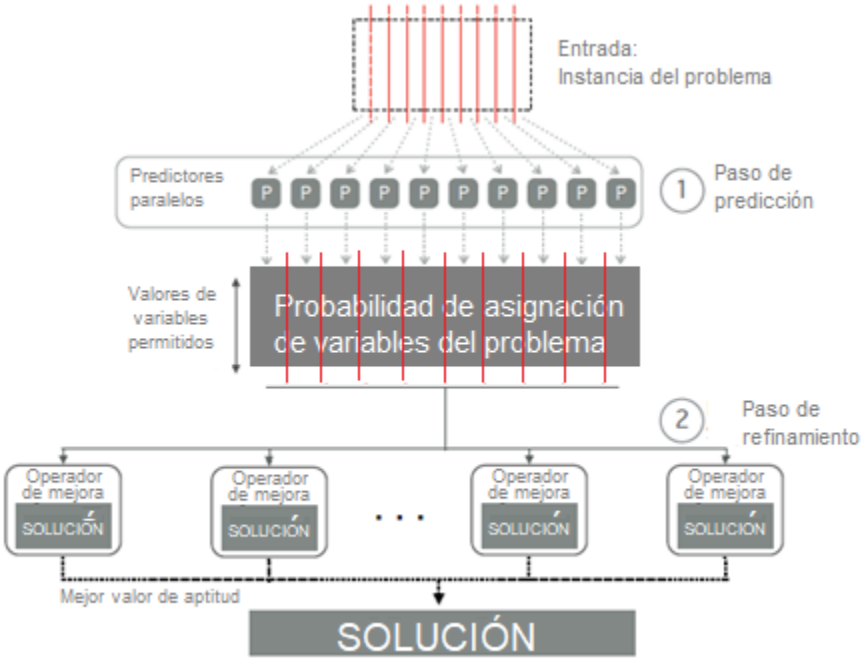

Figura 2.6. Visión general del algoritmo paralelo de Virtual Savant (Massobrio, 2018).

Una solución completa al problema se crea ejecutando un predictor para cada variable, en paralelo. Tenga en cuenta que los predictores son procesos independientes que no comparten ninguna información, por lo que el modelo construye la solución de forma masiva y paralela, y puede utilizar tantos procesos paralelos como la cantidad de variables que tenga el problema. Este número se puede multiplicar en caso de que los predictores sean paralelos también.

Los predictores proporcionan las probabilidades de asignar cada valor permitido a cada variable. Esto proporciona a VS más información para corregir posibles imprecisiones de predicción en el segundo paso: el paso de refinamiento. Se encuentra en la ejecución de una serie de algoritmos de búsqueda en paralelo, con el fin de encontrar una solución precisa para el problema. Las soluciones de inicio se generan de acuerdo con las probabilidades de asignación de valores calculadas en el paso de predicción. La mejor solución encontrada en el paso de refinamiento se reporta como la salida de VS.

2.8.3 El paso de aprendizaje de VS

En la implementación del VS se utiliza la SVM. La SVM se entrena con casos de problemas resueltos, obtenidos de un solucionador existente, que debe ser paralelizado.

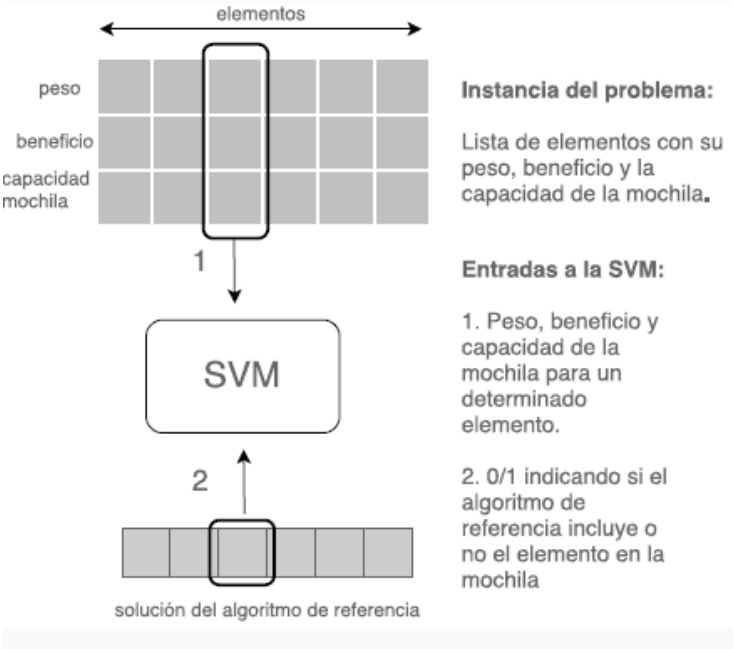

Figura 2.7. Proceso de entrenamiento del VS para el problema de la mochila (Massobrio, 2018).

Las características utilizadas para entrenar el SVM se destacan remarcándolos con el número 1 en la Figura 2.7 y son: información de la instancia del problema e información del algoritmo de referencia.

Por lo tanto, los VS capacitados pueden aplicarse a instancias de cualquier tamaño. Además, este modelo acelera el proceso de aprendizaje, porque solo se necesita capacitar a una SVM.

2.8.4 El paso de predicción de VS

Una vez que los SVM se entrenan en el paso de aprendizaje, se pueden usar para predecir el comportamiento del algoritmo original en instancias de problemas no vistos. La fase de predicción crea una solución tentativa de una manera masivamente paralela, utilizando tantos procesos concurrentes independientes (los predictores). Los predictores son clasificadores SVM.

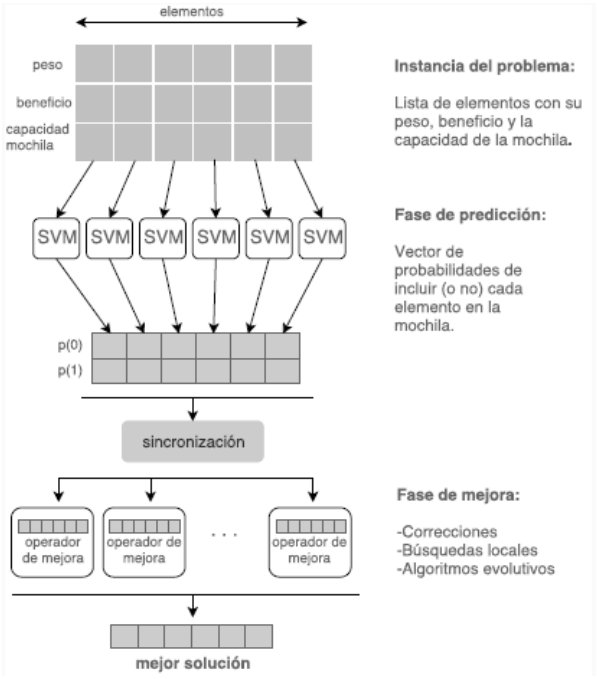

Figura 2.8. Fase de predicción del VS para el problema de la mochila (Massobrio, 2018).

La salida de cada SVM no es una asignación definida para una tarea, sino un vector de probabilidades de asignación (Figura 2.8). El vector de probabilidades contiene las probabilidades para cada elemento de la solución, de acuerdo con el comportamiento aprendido del solucionador de referencias. Tomamos esta decisión para tener más información para el siguiente paso, llamada la fase de refinamiento o mejora, con el fin de hacer frente con eficacia y eficacia a las posibles inexactitudes en las predicciones.

En el diseño presentado, las comunicaciones son casi insignificantes, ya que cada proceso requiere como entrada las instancias en todas las máquinas, es decir, un vector de valores, y su salida es un vector con las probabilidades. Además, los predictores podrían ser paralelos entre sí, aumentando el paralelismo del modelo.

2.8.5 El paso de refinamiento de VS

Los resultados de los predictores no se pueden utilizar como están. Existe la necesidad de un método para reparar posibles imprecisiones dado que en los problemas de optimización combinatoria (como en el problema de programación que se aborda en este trabajo) existe una fuerte dependencia entre los valores de las diferentes variables en una solución,

mientras que la descomposición propuesta asigna cada elemento de la solución de forma independiente.

El paso de predicción construye el vector de probabilidades utilizando una serie de procesos completamente independientes (que también pueden ser paralelos). El paso de refinamiento (los cuadros de "Algoritmo de refinamiento" en la Figura 2.8) cierra la brecha entre las tareas independientes y la naturaleza de la optimización combinatoria del problema, ya que funciona con soluciones completas haciendo uso de la función de aptitud.

Cada refinador comienza generando una solución completa al problema de optimización, con un método aleatorio que hace uso de la matriz de probabilidades generada en el paso de predicción. Los refinadores no se limitan al ensamblaje de la solución, sino que también buscan soluciones de ajuste.

Cada refinador realiza una búsqueda aleatoria en el espacio de la solución, almacenando la mejor solución encontrada, que se envía al final del proceso. Esto se hace con el propósito de generalidad. En esencia, debido a que la búsqueda aleatoria no es eficiente, la búsqueda del refinador se guía por la salida de los predictores.

La búsqueda realizada por los refinadores es maleable, lo que significa que el esfuerzo de cálculo necesario está en función de la cantidad de iteraciones realizadas (evaluaciones de la solución). Sus dos componentes, la generación aleatoria y la evaluación de la condición física, se pueden ejecutar en cualquier número de nodos, porque son sin estado.

2.9 Lógica difusa compensatoria

Básicamente, la Lógica Difusa (Fuzzy Logic, en inglés) o borrosa es una lógica multivalente que permite representar matemáticamente la incertidumbre y la vaguedad, proporcionando herramientas formales para su tratamiento (Morcillo, 2011).

En la lógica convencional un elemento pertenece o no a un conjunto, dando valores de verdadero o falso o de cero y uno para indicar esa pertenencia, mientras que en la LD se define un grado de pertenencia a ese conjunto, es decir qué tanto es cierto o falso que ese

elemento pertenece a un grupo tomando valores entre el falso y verdadero o entre el 0 y 1 convencionales

En la figura 2.9 se ilustra como en la Lógica Clásica, a partir de la altura 1.8 m se da pertenencia de 1 y para alturas inferiores queda en cero. Para la Lógica Difusa, mientras más se acerca a la altura de 1.8, mayor grado de pertenencia tiene al conjunto difuso de alto (Cejas, 2011).

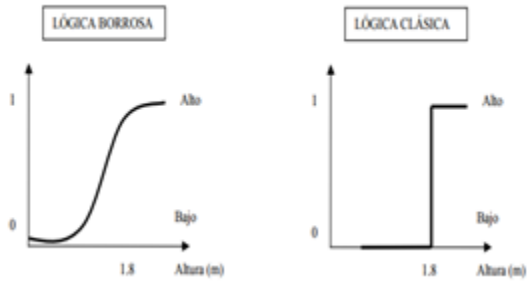

Figura 2.9. Diferencia en la definición de pertenencia entre un conjunto difuso y uno clásico (Cejas, 2011).

Las lógicas multivalentes se definen en general como aquellas que permiten valores intermedios entre la verdad absoluta y la falsedad total de una expresión. Entonces el 0 y el 1 están asociados ambos a la certidumbre y la exactitud de lo que se afirma o se niega y el 0,5 a la vaguedad y la incertidumbre máximas (Cejas, 2011).

Se definen entonces grados de certidumbre, donde si un valor de verdad está más cercano a 1, se considera más cierto que falso, caso contrario para los valores cercanos al 0.

2.9.1 La lógica difusa compensatoria

La LDC es un modelo lógico multivalente que permite la modelación simultánea de los procesos deductivos y de toma de decisiones. Sus características más importantes son: la flexibilidad, la tolerancia con la imprecisión, la capacidad para modelar problemas no-lineales y su fundamento en el lenguaje de sentido común (Cejas, 2011).

La LDC debe de cumplir con axiomas, entre los cuales se encuentran el de compensación (que define esta lógica) y el de veto.

El axioma de compensación indica que, si una variable en un predicado impacta de manera significativa su evaluación, puede compensarse con el cálculo de otra variable.

Por otra parte, el axioma del veto advierte que, bajo ciertas circunstancias, el axioma de compensación no podrá ser aplicado.

El objetivo de la LDC es el descubrimiento de conocimiento, que pueda ser interpretable mediante tareas de evaluación descubrimiento o inferencia de predicados, entre otras.

2.9.2 Estados lingüísticos

Los estados lingüísticos son aquellas variables dentro de la lógica difusa cuyos valores son palabras o etiquetas en vez de números, que son definidas por medio de conjuntos difusos, y por tanto, un estado es más específico que una etiqueta (Galindo, 2007).

Ejemplos de estados lingüísticos son, entre otros:

- lento, moderado o rápido para velocidad en un automóvil,
- baja o alta para temperatura,
- económico o costoso, para el precio de algún artículo y,
- pequeño o grande, para el tamaño de objetos.

Su definición formal está dada por la quinteta (X, T(X), U, G, M), (Galindo, 2007) donde:

- X es la variable lingüística,
- T(X) es el conjunto de etiquetas que puede tomar X,
- U es el dominio subyacente, los valores que son representados por las etiquetas lingüísticas,
- G es la gramática para generar las etiquetas lingüísticas y,
- M es la regla semántica que asocia cada etiqueta de T(X) al conjunto de valores en U.

2.9.3 Funciones de pertenencia.

Así mismo, los estados lingüísticos se definen mediante funciones de pertenencia o de membresía. Una función de membresía o de pertenencia fA(x) en un espacio X caracteriza a un conjunto difuso A, donde cada valor x es asociado a un número real dentro del intervalo [0,1] (De Vito, 2006).

Las funciones de pertenencia pueden ser continuas o discretas y, para calcular el resultado de las funciones de pertenencia se deben definir sus parámetros. Por otra parte, las funciones continuas se caracterizan porque tiene un número de soluciones infinito, cualquier variación en *x* causa un valor diferente en el cálculo de la función (Espin, 2009).

Las funciones singleton y map-nominal son ejemplos de funciones discretas mientras que las funciones sigmoidal, sigmoidal negativa y gausiana son ejemplos de funciones continuas (figuras 2.10 y 2.11).

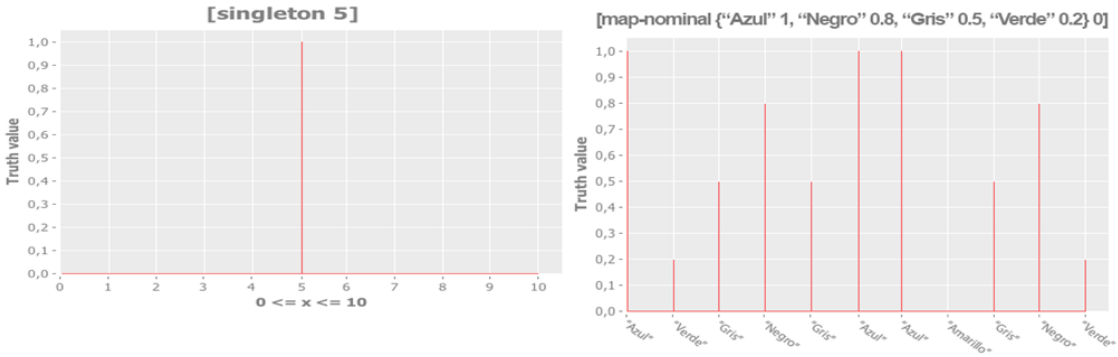

Figura 2.10. Funciones discretas: singleton y map-nominal (Espin, 2009).

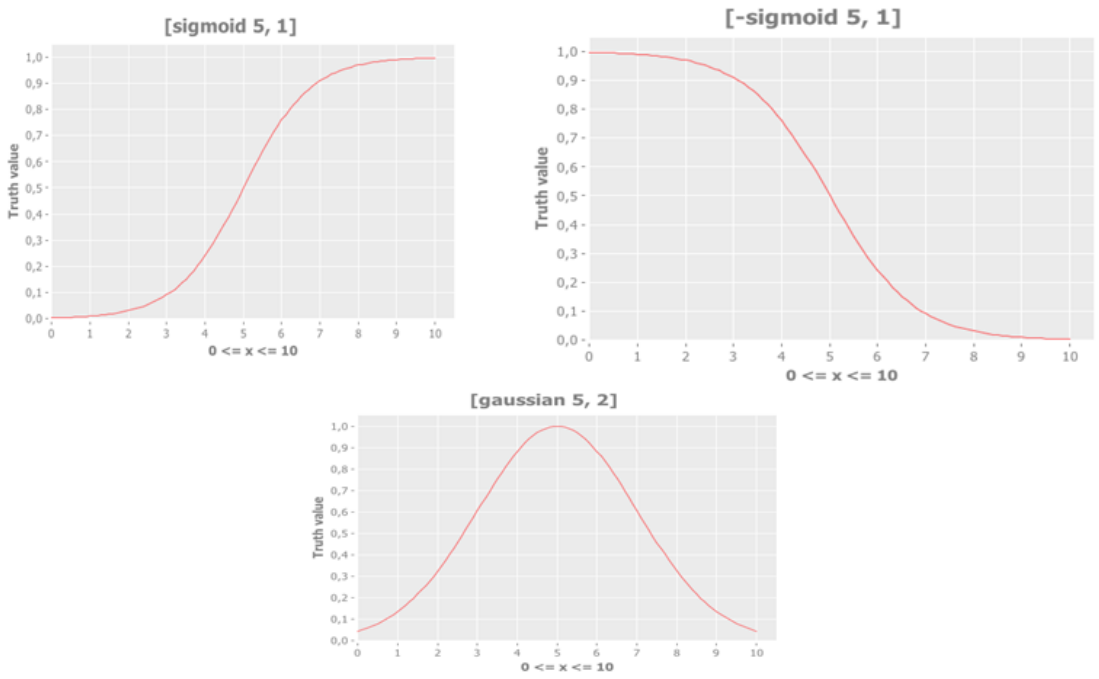

Figura 2.11. Funciones continuas: Sigmoidal, sigmoidal negativa y gausiana (Espin, 2009).

Dentro de las funciones continuas, está la función de pertenencia generalizada (FPG, Espin, 2009), que es muy útil en las tareas de descubrimiento de conocimiento, ya que se puede moldear tomando la forma de las funciones sigmoidales positiva o negativa o de una gausiana, dependiendo de los valores de sus parámetros.

Esta función toma 3 valores, con $\alpha > 0$ $\gamma \in \mathbb{R}$, toma la definición de la función sigmoidal con:

$$\text{sigm}(x, [\alpha, \gamma, m]) = \frac{1}{1 + e^{\alpha(x-\gamma)}} \tag{2.3}$$

$$FPG(x[\alpha, \gamma, m]) = \frac{\text{sigm}(x, [\alpha, \gamma, m])^{m}(1 - \text{sigm}(x, [\alpha, \gamma, m]))^{1-m}}{M} \tag{2.4}$$

Con $M = m^m(1 - m^{1-m})$ donde $0^0 = 1$

2.9.4 Operadores

En la LDC, además de los operadores conjunción, disyunción y negación, que están basados en los análogos de la lógica clásica, existen los operadores implicación y equivalencia, que calculan las inferencias para la toma de decisiones, los operadores de orden y los cuantificadores universal y existencial (López, 2004).

2.9.5 Tipos de lógica.

Se han definido diferentes tipos de lógica para el cálculo de los valores de estos operadores lógicos, en este proyecto se utilizó la Lógica Difusa Compensatoria basada en la Media Geométrica (LDCMG) para la evaluación y descubrimiento de predicados (Espin, 2009).

El cálculo de los operadores conjunción (C), disyunción (D), orden (O) y negación (N) por medio de la LDCMG se muestra a continuación:

$$C(x_1, x_2, \dots, x_n) = (x_1\ x_2\ \dots\ x_n)^{\frac{1}{n}} \tag{2.5}$$

$$D(x_1, x_2, \dots, x_n) = 1 - [(1 - x_1)\ (1 - x_2) \dots (1 - x_n)]^{\frac{1}{n}} \tag{2.6}$$

$$O[x, y] = 0.5[C(x) - C(y)] + 0.5 \tag{2.7}$$

$$N(x) = 1 - x \tag{2.8}$$

Otros ejemplos de tipo de lógica son:

- Lógica de Zadeh.
- Lógica probabilística.
- Lógica Difusa Compensatoria basada en la Media Aritmética (LDCMA).
- Lógica Arquimediana

2.9.6 Predicados

Un predicado es una proposición lógica que afirma o niega algo de un objeto dependiendo de alguna característica en particular, por ejemplo, en una persona se puede calificar la estatura, para evaluar si es alta, o la edad para valorar si es joven o adulto; en cuanto a características, se puede estimar la velocidad de un auto como lenta o rápida.

Los predicados compuestos pueden calificar más de un atributo uniendo los estados lingüísticos, que son definidos a través de las funciones de pertenencia y sus respectivos parámetros, a través de operadores lógicos, que a su vez se determinan por medio de los tipos de lógica.

Los predicados clásicos dan lugar a una clara división del universo A de acuerdo con el cumplimiento del predicado P(x), separando los elementos que cumplen con P(x)

otorgándoles un valor verdadero; y dando a P(x) un valor de falso a los que no cumplen con el predicado. (Olmo, 2008).

En la LDC esta división no es absoluta y los predicados pueden ser evaluados para calcular su valor de verdad de acuerdo a las funciones de pertenencia de los estados lingüísticos y los operadores lógicos. Los resultados obtenidos de los valores de verdad en la LDC están en el rango continuo de [0, 1].

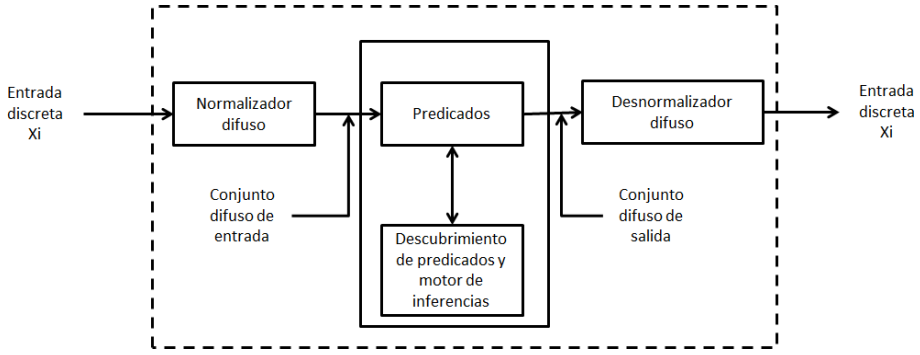

Figura 2.12. Sistema de inferencia basado en lógica difusa compensatoria.

Un sistema basado de inferencia basado en lógica difusa se muestra en la figura 2.12:

- la entrada de datos,
- un módulo normalizador lógico,
- un núcleo que incluye un módulo de descubrimiento de predicados y de inferencia, así como la base de conocimiento generado por este módulo,
- un módulo desnormalizador lógico y,
- la salida de datos en un valor numérico.

En este modelo, para cada variable en la entrada discreta se ejecuta una normalización lógica, llevando los valores de este conjunto a rangos de [0, 1] de acuerdo estados lingüísticos de las reglas difusas almacenadas en el sistema.

Los valores normalizados son evaluados en los predicados o reglas para obtener su valor de verdad. Finalmente, los valores de verdad obtenidos en la evaluación de predicados son desnormalizados para deducir, por parte de un experto, la interpretación de esta salida de acuerdo a su experiencia.

2.9.8 Tareas de descubrimiento de conocimiento

Las tareas de descubrimiento de conocimiento a desarrollar en este proyecto son los siguientes (Espin, 2009), estas tareas son realizadas por el software Eureka-Universe, el cual está soportado por un algoritmo genético encargado de buscar predicados de alto valor de verdad.

Evaluación

La tarea de evaluación calcula los valores de verdad de un predicado difuso para un set de datos. El predicado se construye seleccionando las variables lingüísticas y los operadores lógicos.

El valor de verdad de un predicado difuso se calcula tomando en cuenta las columnas de un conjunto de datos que corresponden a las variables en el predicado.

Las variables se evalúan de acuerdo a los estados lingüísticos por medio de sus funciones de pertenencia. Con los valores resultantes se calcula el valor de verdad realizando las operaciones lógicas de acuerdo al tipo de lógica seleccionada.

Este proceso se efectúa para cada registro en el conjunto de datos y para cada uno de ellos se obtiene el valor de verdad. Además, se calcula tomando en cuenta todos los valores de verdad del conjunto el operador existencial y universal.

Descubrimiento

La tarea de descubrimiento busca relaciones entre los estados lingüísticos de un conjunto de datos (predicados difusos) que cumplan con las especificaciones del usuario.

En esta tarea, teniendo un conjunto de datos, son seleccionados:

- las variables,
- los estados lingüísticos y su configuración,
- los operadores lógicos,
- el tipo de lógica y,
- una estructura genérica, llamado predicado simbólico, para delimitar la forma que tomarán los predicados descubiertos.

Además, se establece una configuración de parámetros para el algoritmo de búsqueda.

Inferencia

La tarea de inferencia realiza primeramente una tarea de descubrimiento en un conjunto de datos en el que se han definido los estados lingüísticos de variables de condición y decisión.

Los predicados difusos obtenidos son utilizados para inferir los valores de las variables de decisión a partir de otro set de datos en el que sólo se conocen las variables de condición.

Capítulo 3

3. Estado del arte

En esta tesis se aborda la solución de BPP mediante el aprendizaje de la conducta de un algoritmo de referencia. Por esta razón se presenta el análisis de dos algoritmos del estado del arte que se basan en este enfoque (VS y C-LDC). También se revisa GGA-CGT, el algoritmo del estado del arte de BPP considerado como referencia. La propuesta de esta tesis se contrasta con los tres algoritmos revisados.

3.1 VS: Generación automática de optimizadores

Por medio de la generación automática de solucionadores paralelos propuesto por el paradigma VS se han abordado problemas como el problema de la siguiente versión de software (Next release problem, NRP, Bagnall, 2001) que es una generalización del KP (descrito en la sección 2.1) y el problema de asignación de tareas (Pinel, 2014), en el cual se asignan tareas a máquinas donde cada máquina realiza una tarea en un tiempo distinto a las demás; el objetivo es minimizar el tiempo de ejecución para todas las tareas.

3.1.1 Problema de asignación de tareas

Entre los algoritmos de solución del problema de asignación de tareas destacan las heurísticas secuenciales Min-Min (Ibarra, 1977), su variante Max-Min (Maheswaran, 1999) y Sufferage (Tabak, 2014). La heurística Min-Min comienza con el conjunto de tareas completo e iterativamente selecciona la tarea que puede ser completada más rápido

tomando en cuenta la actual selección de máquinas y removiéndola del set de tareas candidatas sin asignar.

Debido a que la complejidad de estas heurísticas crece con el tamaño de las instancias, incrementó la tendencia de diseño de solucionadores paralelos para resolver este problema (Pinel, 2017).

La versión paralela de la heurística Min-Min incluyó un algoritmo genético celular y reportó ser 538 veces más veloz que la versión secuencial de esta heurística (Pinel, 2013). Por otra parte, Abraham et al. (Abraham, 2015) propone usar múltiples hilos combinado con Min-Min dividiendo el número de tareas y de procesadores en diferentes grupos, ejecutando el Min-Min de forma independiente paralelamente en cada grupo, incrementando la velocidad de solución de manera significativa, sin reportar la calidad de las soluciones. Por otra parte, Pinel (Dorronsoro, 2010) diseñó un CGA paralelo asíncrono (PA-CGA) multi-hilo.

En el diseño de la construcción automatizada de programas desde el código fuente el paralelismo fue tomado en cuenta como un paso más. La descomposición del programa en tareas que pueden ser ejecutadas en forma paralela, respetando las dependencias entre las tareas, es una investigación de Fonseca et al. (Fonseca, 2016), logrando acelerar el proceso alrededor de 11 veces. Este método aplica transformaciones que no alteran la semántica del programa original, pero se vio limitado ya que preserva el código fuente. Otros trabajos siguieron esta misma línea.

La meta de VS, por otra parte, no es modificar la implementación preservando el algoritmo, sino el producir un nuevo algoritmo completamente.

VS es un método que resuelve problemas de automatización donde anteriormente era necesario conocer la técnica e incluso el problema. Para ello busca aprovechar las arquitecturas de cómputo paralelas que existen actualmente, Utiliza aprendizaje automático para reconocer patrones y resolver problemas en forma masiva y paralela (Pinel, 2017).

Durante el aprendizaje, el VS utiliza un algoritmo de referencia para aprender, entrenando un clasificador como la SVM proporcionando la instancia particionada y la

salida esperada. El entrenamiento se realiza con instancias pequeñas o poco complejas para que en la ejecución se pueda realizar con instancias de mayor tamaño o mayor dificultad.

Durante la ejecución, el método requiere dos pasos, en el primero realiza la predicción utilizando el clasificador ya entrenado para generar una población de soluciones que son mejoradas durante el refinamiento.

Tanto para el aprendizaje como la ejecución del VS, se realizan transformaciones de las instancias, que son originalmente de un problema de optimización combinatoria, para convertirlas a instancias de clasificación.

En el problema de asignación de tareas, la instancia se representa con una matriz de tiempo esperado de ejecución (expected time to compute, ETC), en esta matriz las columnas representan cada una de las tareas y las filas representan las máquinas disponibles (figura 3.1). Cada elemento *i*, *j* en la matriz contiene el tiempo que se tarda en ejecutar la tarea *j* en la máquina *i*. En la conversión del problema, cada identificador de máquina de la instancia de optimización se toma como una clase en la instancia de clasificación.

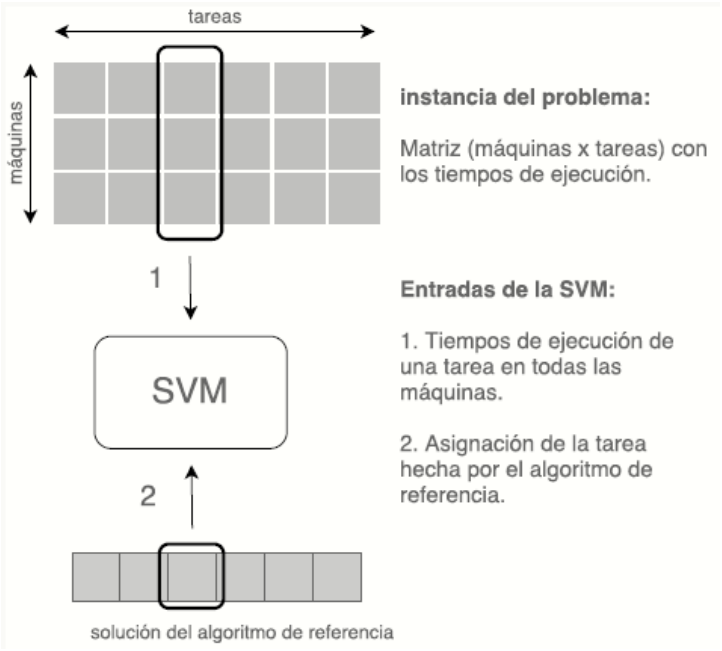

Figura 3.1. Entrenamiento de la SVM para el problema de asignación de tareas (Massobrio).

Cada elemento del vector solución en la figura 3.1 (solución del algoritmo de referencia) representa una tarea y el valor que contiene representa la máquina a la que fue asignada dicha tarea.

3.1.2 Problema de la mochila

Para el KP, el algoritmo de referencia es el Nemhauser-Ullmann, un algoritmo exacto que no impone restricciones adicionales sobre su entrada. Este algoritmo utiliza funciones crecientes con un número finito de peldaños, que se pueden representar como listas de pares que contienen las coordenadas de cada peldaño (Palomo-Lozano, 2016).
Aunque las funciones crecientes reducen el tiempo de ejecución de este algoritmo, sigue siendo un no-polinomial en el peor caso (Röglin, 2010).

Previamente se ha mencionado que tanto para el aprendizaje como la ejecución del VS, se requiere realizar transformaciones de las instancias, pasando de un problema de optimización combinatoria a un problema de clasificación. En el caso del KP, la variable de decisión es tomada como la clase, indicando si el objeto es o no incluido en la mochila.

3.2 Optimización basada en clasificación con LDC

Como un estudio de factibilidad de la presente tesis, se desarrolló un primer clasificador basado en LDC (C-LDC), para su aplicación en problemas de optimización y clasificación en temas como el de la recolección de pedidos en un almacén (Padron-Tristan et al., 2019) y el de diabetes (Padron-Tristan et al., 2020), respectivamente.

El problema de recolección de pedidos en un almacén es un problema de optimización NP-Duro donde, dato una cantidad de órdenes de pedido, se debe optimizar la distancia recorrida dentro del almacén para recolectar todas las unidades incluidas en todas las órdenes, donde existen diferentes formas de recorrer el almacén (Fuentes, 2011).

En este problema se plantea el recorrido por lotes, donde dado un conjunto de órdenes, éstas se recolectan completamente y se debe obtener el conjunto de órdenes que son atendidas en un mismo recorrido. Para este procedimiento se utilizan soluciones conocidas mediante el algoritmo *First Fit-Enveloped Based Batching (*FF-EBB, Rubens, 1999).

Este problema de optimización se resuelve en Fuentes (2011) mediante la técnica de clasificación con el algoritmo C4.5 para, previa combinación de todas las órdenes de pedido, determinar cuáles de ellas son recolectadas en el mismo recorrido.

En la figura 3.2 se muestra:

a) La instancia original, que es generada aleatoriamente con las órdenes de pedidos solicitadas en el lapso de una hora, por cada orden se indican los artículos incluidos, el volumen total de la orden y el identificador más lejano al mostrador del almacén;
b) los lotes generados por medio de la heurística FF-EBB, indicando las órdenes de pedidos en cada lote o recorrido por el recolector, así como la distancia y los pasillos recorridos y la utilización del recolector, y;
c) un fragmento de la instancia de clasificación final, de acuerdo a los lotes generados, donde se muestran las órdenes, las variables correspondientes a cada orden y la variable de decisión *batched*, que indica si las órdenes son recolectadas en el mismo lote.

Llegada	ID Orden	Elementos solicitados				Vol	Max Elem.
08:08	O01	156				1	156
08:14	O02	89				1	89
08:16	O03	15	57	139		3	139
08:18	O04	11	41			2	41
08:21	O05	24	32			2	32
08:23	O06	12	101	217	227	4	227
08:26	O07	165				1	165
08:35	O08	179				1	179
08:52	O09	132				1	132
08:59	O10	27	89	188		3	188

a)

Número de lotes	Ordenes en lote	Distancia de recolección	Pasillos recorridos	Pasillos cruzados	Utilización recolector
1	1, 2, 4, 5, 9	137.2 m	3	2	100%
2	3, 7, 8	117.2 m	2	2	71%
3	6, 10	182.4 m	3	3	100%
Total		436.8 m	9	7	90%

b)

Par Ordenes	Pasillos O1	Vol O1	Dist O1	Pasillos O2	Vol O2	Dist O2	Pasillos en común	Pasillos extra	Pasillos cruzado	Dist. lote	enlotado ?
O1 - O2	1	1	37.18	1	1	105.17	0	1	2	117.20	1
O1 - O3	1	1	37.18	2	3	117.2	1	0	2	37.18	0
O1 - O4	1	1	37.18	1	2	36.63	0	1	2	117.20	1
O1 - O5	1	1	37.18	1	2	6.66	0	1	2	117.20	1
O1 - O6	1	1	37.18	3	4	182.4	0	0	2	37.18	0
O1 - O7	1	1	37.18	1	1	67.15	1	0	2	37.18	0
O1 - O8	1	1	37.18	1	1	113.77	1	0	2	37.18	0
O1 - O9	1	1	37.18	1	1	57.16	1	0	2	37.20	1
O1 - O10	1	1	37.18	3	3	152.44	0	0	2	37.18	0
O2 - O3	1	1	105.17	2	3	117.2	0	0	1	105.17	0
O2 - O4	1	1	105.17	1	2	36.63	0	1	1	108.60	1
O2 - O5	1	1	105.17	1	2	6.66	0	1	1	108.60	1
O2 - O6	1	1	105.17	3	4	182.4	1	0	1	105.17	0
O2 - O7	1	1	105.17	1	1	67.15	0	0	1	105.17	0
O2 - O8	1	1	105.17	1	1	113.77	0	0	1	105.17	0

c)

Figura 3.2. Transformación de una instancia de optimización a una de clasificación para el problema de recolección de órdenes en un almacén (Padron-Tristan et al., 2019).

En este problema se compara el comportamiento del algoritmo C4.5 con el clasificador basado en LDC, obteniendo una mejora con el 86 % de efectividad descubriendo predicados difusos que son utilizados como reglas de clasificación.

Por otra parte, el clasificador basado en LDC se utilizó también en el problema de clasificación de diabetes, donde tomando el conjunto de datos PIMA Indian dataset (PID, PIMA, 2016), que incluye registros de mujeres de herencia Pima en Estados Unidos con alta estadística en problemas de diabetes.

Cada registro en el PID contiene datos de una paciente como el nivel de glucosa, las veces que ha estado embarazada, la edad, presión diastólica, índice de masa corporal, entre otras. La variable de decisión indica si la paciente tiene diabetes o no.

Se comparó el comportamiento y precisión con tres clasificadores como son Self-Organized Maps based Fuzzy Predicate Clustering (SFPC, Meschino, 2015), el Spider Monkey Mine-Ruler (SM-RuleMiner, Cheruku, 2017) y el Recursive-Rule Extraction (RE-RX, Hayashi, 2016). El clasificador basado en LDC obtuvo una precisión del 77 % contra el 72 % del SFPC, el 89 % del SM-RuleMiner y el 84 % del Re-RX.

El sistema Eureka-Universe (Padron-Tristan, 2017) se utilizó para la tarea de descubrimiento de conocimiento en los proyectos del problema de order picking y el relacionado con diabetes.

Eureka-Universe está basado en la metodología de la LDC, que apoya a la toma de decisiones mediante la interacción con un núcleo científico, que está orientado para su empleo tanto a usuarios especializados en un área determinada pero que no son expertos en herramientas científicas, como a usuarios investigadores (Espin-Andrade, 2018).

Implementa una arquitectura fundamentada en el modelo general de los sistemas basados en la lógica difusa compensatoria, con los módulos necesarios para la administración y procesamiento de los datos de entrada y de salida, de las configuraciones de las consultas de entrada y la interacción con el núcleo científico.

3.3 Algoritmo evolutivo GGA-CGT

El estudio de la literatura relacionado con el BPP abordado por Quiroz (2015) para el GGA-CGT presentó los siguientes trabajos: el algoritmo genético de agrupamiento híbrido (hybrid grouping genetic algorithm, HGGA, Alvim, 2004), la heurística de mejora híbrida (hybrid improvement heuristic, HI_BP, Falkenauer, 1996) y Perturbation-SAWMBS (Fleszar, 2011) como los mejores hasta el momento, los cuales se describen a continuación.

Para el HGGA, Falkenauer (1996) propuso un algoritmo genético de agrupamiento híbrido que usa un esquema de codificación especial para grupos de elementos y heurísticas inspiradas por el criterio de dominación de Martello y Toth (1990), en el que se establece que es posible hacer facilitar la mejora de una solución extrayendo un conjunto de elementos pequeños de un contenedor y sustituirlos por un elemento libre con el mismo peso.

Por otra parte, Alvim et al. (Alvim, 2004) presentó una heurística de mejora híbrida, en la que se reintroduce la estrategia dual usada por Scholl et al. (Scholl, 1997) en adición a la técnica de reducción del espacio de búsqueda utilizado por Martello y Toth (1990), además de utilizar estrategias de límite inferior.

Gutpa y Ho (1999) propusieron la heurística holgura mínima del contenedor (Minimum bin slack, MBS) que realiza una búsqueda lexicográfica para llenar cada contenedor, minimizando el espacio sobrante en cada uno de ellos, parte de la idea de que a menor espacio sobrante en cada bin, menor cantidad de contenedores.

Fleszar e Hindi (2002) mediante el algoritmo Perturbation-MBS modificaron el algoritmo MBS construyendo nuevas soluciones con mejor calidad mediante la perturbación de las soluciones encontradas. Posteriormente, Fleszar y Charalambous (2011) modificaron el algoritmo Perturbation-MBS usando el nuevo principio peso promedio suficiente (Sufficient average weight, SAW) para controlar el peso promedio de los elementos empacados en cada contenedor.

3.4 Propuesta de esta tesis: VS basado en LDC

Actualmente, el VS ha resuelto problemas como el KP y el de asignación de tareas, utilizando la SVM como clasificador. En esta tesis se busca analizar el comportamiento de la LDC como clasificador para reemplazar la SVM y realizar una comparación.

Además, este análisis se realiza con un problema no abordado por el VS, como lo es el BPP, que agrega la complejidad de no saber inicialmente la cantidad de contenedores que son la solución al problema de optimización y que es variable en cada instancia de un mismo tipo.

Para el KP, la cantidad de clases es conocida, tomando sólo dos valores que representan si el objeto es o no incluido o no en la mochila (0, 1). Mientras que para el problema de asignación de tareas, la cantidad de clases representa el total de máquinas que están disponibles en la instancia y también se conoce de manera inicial.

Este proyecto resuelve el 1-DBPP basado en la generación de optimizadores paralelos utilizada por el VS incorporando el clasificador basado en LDC, que reemplaza la SVM. Además compara la precisión de VS que utiliza la SVM con el que incorpora la LDC.

Tabla 3.1. Comparación de este proyecto con el estado del arte.

Enfoque	**Problema de optimización**	**Entrenamiento y Prueba**	**Refinamiento**	**Modelo paralelización**	**Comparación**
VS (Pinel, 2014)	1. Mochila 2. Asignación de tareas	SVM	Búsqueda local	Map-reduce	Heurístico de referencia
Clasificación LDC (Padrón et al, 2019, 2020)	1. Order-picking 2. Diabetes Problem	Descubrimiento y evaluación	Selección de predicados	NA	Clasificador C4.5
GGACGT (Quiroz, 2014)	1D-BPP	NA	Reacomodo por pares con Primer ajuste	NA	Heurístico de referencia
Este trabajo (Padrón-Tristán)	1D-BPP	LDC para clasificación	Reacomodo por pares con mejor ajuste	Map-reduce	VS-SVM

Además, este proyecto aprovecha la metodología revisada en el trabajo de Fuentes (2011), en el cual la instancia de optimización se transforma a una de clasificación, integrando una combinación de las órdenes de pedidos para determinar cuáles son recolectadas en el mismo recorrido o lote. Un análisis de los trabajos relacionados se muestra en la tabla 3.1.

Capítulo 4

4. Metodología de solución

Para contribuir a la solución de BPP se propone el método VS-LDC, el cual aprende el comportamiento de un algoritmo de referencia aplicado a la solución de instancias de BPP con solución conocida. Posteriormente, VS-LDC aplica el modelo aprendido a la solución de nuevas instancias de mayor dimensión. El enfoque seguido, conocido como Virtual Savant, fue concebido para el cómputo paralelo.

En este capítulo de describe la metodología seguida en el desarrollo de VS-LDC, particularmente en el diseño de sus componentes para BPP: un convertidor de instancias de optimización a instancias de clasificación y viceversa, un clasificador probabilístico basado en lógica difusa compensatoria, y heurísticas de refinamiento de la solución.

4.1 Propuesta de solución

La propuesta de solución de BPP se basa en la arquitectura VS presentada en la figura 4.1. El primer paso consiste en convertir un problema de optimización, en este caso BPP, a un problema de aprendizaje y clasificación, tomando en cuenta las variables de la instancia original: el peso del objeto, la capacidad del contenedor y el identificador del almacén donde es almacenado el objeto.

En el problema de clasificación, los identificadores de los contenedores a los que se asignan los objetos son tomados como las clases. En el paso de predicción, las instancias transformadas son la entrada al VS-LDC, que procesa las observaciones con los clasificadores basados en LDC que son ejecutados paralelamente para generar un vector de valores de verdad, que será tomado como el vector de probabilidades.

Se genera aleatoriamente una población de soluciones candidatas con la distribución dada por el vector de probabilidades. Posteriormente, estas soluciones son decodificadas para obtener soluciones del problema original de optimización.

En el paso de refinamiento, a las soluciones decodificadas se les aplican heurísticas que son propias de BPP como operadores de mejora.

Finalmente, los valores de aptitud de todas las soluciones mejoradas son comparados para tomar la mejor solución como la salida del VS-LDC.

4.2 Arquitectura de Virtual Savant para BPP

A continuación se describe la arquitectura de VS especializada para la solución de BPP, en la que aparece un módulo para codificar las instancias que originalmente son de un problema de optimización para transformarlas a instancias de clasificación (Figura 4.1).

Durante el paso de predicción, cada observación en la instancia codificada de clasificación es procesada por clasificadores basados en LDC de forma paralela e independiente para generar un vector con los valores de verdad que son tomados como el valores de probabilidad para generar una población de soluciones que son decodificadas.

Durante el paso de refinamiento, a las soluciones decodificadas se les aplica un operador de mejora, seleccionando de ellas la mejor de acuerdo a su valor de aptitud para ser la salida de VS-LDC.

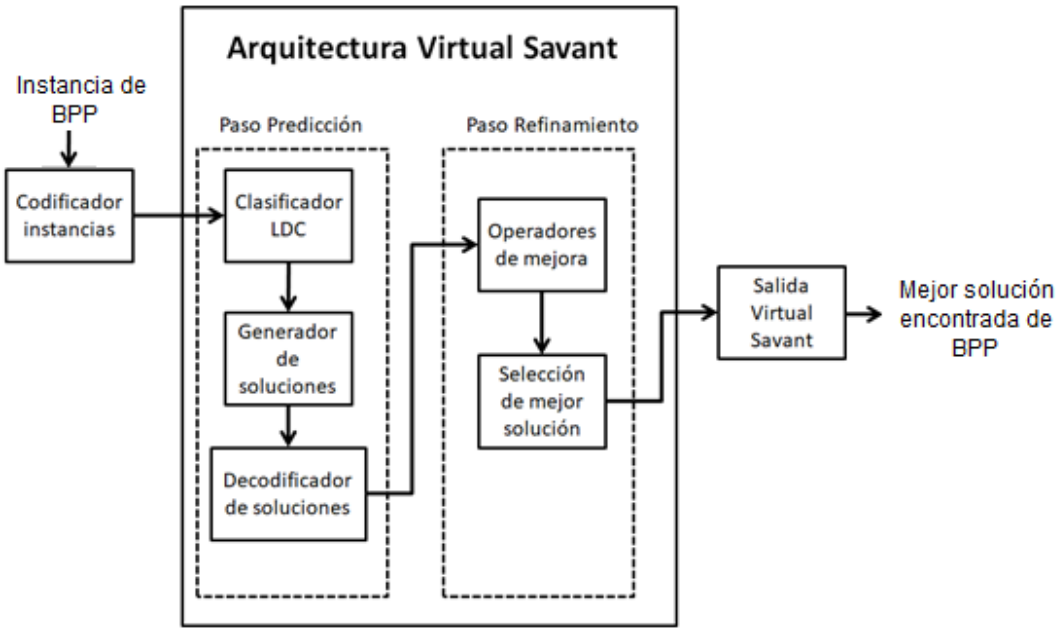

Figura 4.1. Arquitectura VS-LDC aplicada a BPP.

4.3 Transformación de instancias

Originalmente las observaciones de las instancias de optimización están conformadas por el peso del objeto, la capacidad del contenedor y la variable de decisión, que indica el identificador del contenedor en el que será almacenado el objeto.

Las instancias serán transformadas para ser la entrada a un problema de clasificación, donde son tomados en cuenta los identificadores de los contenedores como clases a los cuales corresponde cada observación en la instancia.

4.3.1. Propuesta 1

La primera propuesta consiste en contar con la instancia tal cual, tomando en cuenta el peso del objeto, la capacidad de los contenedores y la variable de decisión, que toma el valor del contenedor al cual es asignado el objeto, así, cada observación de la instancia está conformada por (figura 4.2):

a. el peso w_i del objeto,
b. la capacidad c del contenedor y
c. el identificador x del contenedor al cual es asignado el objeto.

```
w,c,x
99,100,0
99,100,1
96,100,2
96,100,3
92,100,4
92,100,5
91,100,6
88,100,7
87,100,8
86,100,9
85,100,10
76,100,11
```

Figura 4.2. Ejemplo de instancia de clasificación con la variable de decisión x tomando el valor de la clase.

De esta forma cada identificador de los contenedores será una clase en el problema de clasificación.

La desventaja de esta propuesta es que las soluciones para cada instancia cuentan con un número desconocido de contenedores, por lo que no se puede tener la cantidad de clases inicialmente para el proceso de aprendizaje para el VS-LDC.

.3.2. Propuesta 2

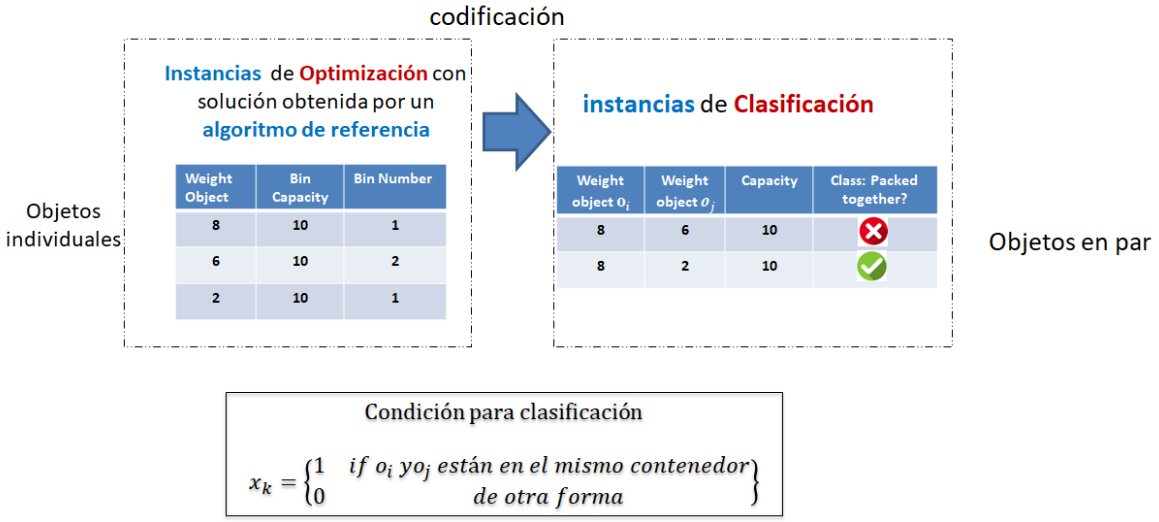

Figura 4.3. Codificación de instancias de optimización a clasificación.

Para la segunda propuesta, cada una de las observaciones es transformada para convertir las instancias a un problema de clasificación, realizando combinaciones de objetos y de acuerdo con la instancia original indicar si son almacenados en el mismo contenedor, teniendo en cuenta los pesos y la capacidad residual causada por el primer objeto (figura 4.3).

Se analizó una relación de pesos de dos objetos y la capacidad residual ocasionada por el primer objeto, de tal forma que la variable de decisión toma el valor de 1 o 0, significando que dichos objetos, *i, j,* se puedan almacenar en el mismo contenedor. En cada observación de la instancia se incluye:

1. peso w_i del objeto *i*,
2. peso w_j del objeto *j*,
3. capacidad *c* residual del contenedor y ocasionada por el primer objeto,
4. la variable de decisión *x*, indicando si los objetos i y j se almacenan en el mismo contenedor.

La cantidad de observaciones generadas para cada instancia transformada se eleva a:

$$n = \frac{n'!}{2!\,(n'-2)!}$$

Siendo *n'* la cantidad de observaciones en la instancia de optimización.

4.3.3. Propuesta 3

Para la tercera propuesta, la instancia transformada de clasificación contiene las probabilidades de que un objeto sea almacenado con cada uno de los objetos restantes, las instancias estarán conformadas por una matriz de $n \times m$ con $m = n + 1$ donde cada elemento $m_{i,j}$, con $i, j = 1 \ldots n$, contiene la relación del objeto $o_i$con el objeto o_j mediante la siguiente fórmula:

$$\boldsymbol{m_{i,j} = \begin{cases} \dfrac{o_j}{c - o_i} & si\ c > o_i \\ 0 & de\ lo\ contrario \end{cases}} \qquad (4.1)$$

con c = capacidad del contenedor.

La columna m contiene el identificador del objeto j, aquél con el que el objeto i fue almacenado en el mismo contenedor.

Tanto la segunda como la tercera propuesta proporcionan la relación de objetos que se almacenan en un mismo contenedor, para ambas se debe hacer un post procesamiento que genere la asignación de objetos a los contenedores en la población de soluciones como la salida del paso de predicción y la entrada en el paso de refinamiento.

De las tres propuestas se seleccionó la segunda para la transformación de instancias de optimización a instancias de clasificación.

En esta tesis se propone un clasificador basado en LDC, y para evaluar su desempeño, en el capítulo de experimentación se contrasta con el muy conocido clasificador SVM. Por esta razón, a partir de esta sección se hace referencia a ambos clasificadores.

4.4 Aprendizaje de clasificadores de VS

El entrenamiento se realiza con las instancias ya codificadas, y durante esta etapa se crea un modelo de clasificador, tanto con la SVM como con la LDC.

Para el aprendizaje además se cuenta con instancias de entrenamiento, en las que se cuenta con observaciones con la clase 1, donde dos objetos se almacenan en un mismo contenedor, y la clase 0, donde esto no sucede.

Debido a que las instancias son creadas a partir de combinaciones de dos objetos, el número de observaciones para la clase 0 es mucho mayor que el número de observaciones para la clase 1, por lo que se pierde representatividad de esta clase en las instancias. Esto puede afectar el proceso de aprendizaje.

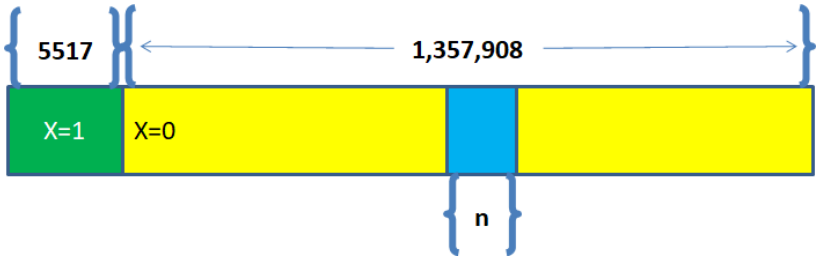

Figura 4.4. Ejemplo de balanceo de instancias para las clases X=0 y X=1.

Es por ello que para esta etapa se balancean las instancias, tomando una muestra de las observaciones de la clase 0 (ejemplo con la figura 4.4).

La segunda tarea de preprocesamiento es obtener una muestra de la instancia balanceada, comparando el proceso de aprendizaje del clasificador con varios tamaños de muestra (figura 4.5).

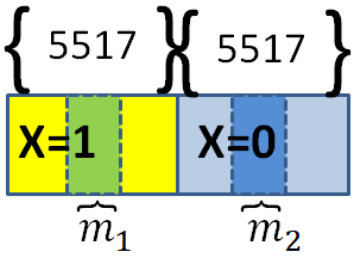

Figura 4.5. Ejemplo de muestreo para las clases X=0 y X=1.

Finalmente, los valores que toman las variables en las instancias tienen diferentes rangos para los pesos de los objetos y la capacidad del contenedor.

Para tener un proceso de aprendizaje y ejecución que sea general para todas las instancias, se escalaron los valores de las variables, tomando en cuenta la capacidad del contenedor, así, los pesos de los objetos *i*, *j* y la capacidad residual del contenedor, se dividieron por la capacidad total del bin.

4.4.1 Entrenamiento SVM

En una primera etapa, se seleccionan los mejores parámetros, el de la región de decisión para la función kernel RBF y el coste (γ y c) mediante una búsqueda de rejilla, o *gridsearch*.

Se generan modelos de la SVM tomando en cuenta los parámetros γ y c, tamaño de instancia de entrada, la proporción de clases en la variable de decisión, eligiendo el mejor modelo por medio de validación cruzada de 10 bloques. El mejor modelo de SVM se selecciona como el clasificador de VS (figura 4.6, algoritmo 4.1).

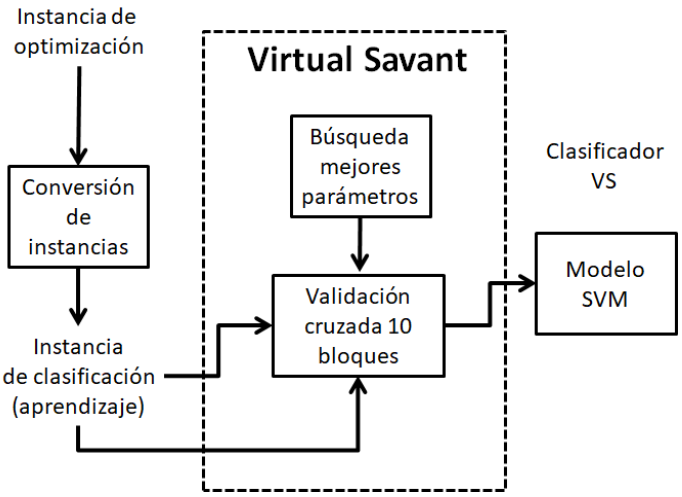

Figura 4.6. Entrenamiento SVM como clasificador VS.

Algoritmo 4.1: Ajuste de parámetros de VS con SVM APVSSVM (dataset)

Busca y selecciona los mejores parámetros costo y gama para el entrenamiento de la SVM mediante gridsearch (búsqueda de rejilla) y validación cruzada de 10 bloques.

Entrada: *dataset*, conjunto de datos de entrada transformado en instancia de clasificación

Salida: Mejores parámetros *costo*, *gama* y *precisión* del modelo para entrenamiento SVM

1. *mejor_precision* = 0
2. *tipoSVM* = "clasificacion"
3. *kernel* = "RBF"
4. Crear *k*=10 bloques de *dataset* para la validación cruzada
5. Crear *grid*, con permutaciones de *costo* y *gama* = 2^{10} hasta 2^{-10} con saltos de potencia de 10
6. *precision_parametros = 0*
7. **para cada** *costo* y *gama* en *grid*
8. *precisión_bloque* = 0
9. **para** j=1 **hasta** *k*
10. $tabla_E$= *dataset* menos bloque *j*
11. $tabla_P$= bloque *j* de *dataset*
12. x_E= variable de decisión *x* de *dataset* menos bloque *j*
13. x_P= bloque *j* de variable de decisión *x* de *dataset*
14. *SVM* = crearModelo($tabla_E$, x_E, *costo*, *gama*, *tipoSVM*, *kernel*)
15. *vector_probabilidades* = ejecutarSVM(*SVM*, $tabla_P$, x_P)
16. *precisión_bloque = precisión_bloque +* calcularPrecision(*vector_probabilidades*, x_P)
17. *precisión_parametros = precisión_bloque / k*
18. si *precisión_parametros > mejor_precision*
19. *mejor_costo = costo*
20. *mejor_gama = gama*
21. **retornar** *mejor_costo, mejor_gama, mejor_precision*

4.4.2 Entrenamiento C-LDC

El modelo del clasificador basado en LDC obtiene un conjunto de predicados como se muestra en el algoritmo 4.2. Toma como entrada un predicado que se selecciona de la siguiente manera: tomando como entrada la instancia de clasificación que, siguiendo la técnica de validación cruzada, será particionada en bloques con los cuales se realizará la tarea de descubrimiento, generando un listado de predicados, de los cuales se tomará el valor de verdad de cada observación como el vector de probabilidades en la SVM.

Posteriormente, los predicados descubiertos serán evaluados con las instancias particionadas, tomando el valor de verdad de cada observación y tomando el predicado con mayor cantidad de aciertos en la clasificación como el modelo del VS (figura 4.7).

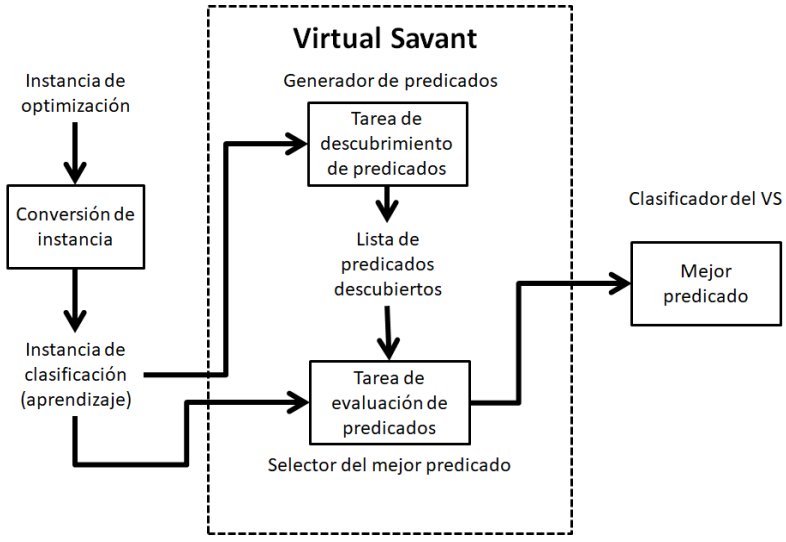

Figura 4.7. Entrenamiento LDC como clasificador VS.

Durante la etapa de aprendizaje se proporciona la instancia y su solución, además de un predicado simbólico que indica la estructura de las reglas que serán obtenidas.

Un predicado simbólico incluye, en este caso, el operador de implicación y la clase, además de uno o varios estados y sus respectivas configuraciones de funciones de pertenencia y parámetros. También existe un predicado comodín, representado con un asterisco (*), el cuál representará cualquier predicado descubierto que cumpla con las condiciones del algoritmo de búsqueda. Ejemplos de predicados simbólicos se muestran en la figura 4.8:

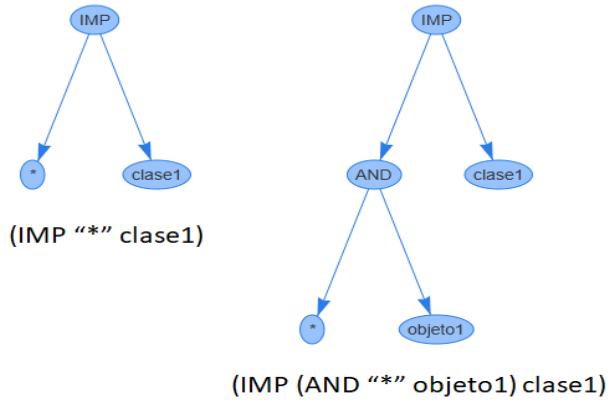

Figura 4.8. Ejemplos de predicados simbólicos

El predicado simbólico está sujeto a la configuración del algoritmo de búsqueda utilizado en la tarea de descubrimiento de predicados e incluye:

- población de predicados a generar,
- nivel de profundidad de los predicados,
- la cantidad de resultados a desplegar,
- los estados lingüísticos,
- operadores lógicos y
- valor de verdad mínimo, entre otros.

La tarea de descubrimiento trae como resultado todos los predicados que cumplan con la estructura del predicado simbólico y con la configuración del algoritmo de búsqueda (tabla 4.1). Además, en su caso, puede calcular los parámetros de la función de pertenencia especificada para los estados lingüísticos.

Además, la tarea de descubrimiento despliega los mejores predicados con base en el valor de verdad calculado, que cumplan con el valor de verdad mínimo que fue especificado en la configuración del algoritmo de búsqueda.

De esta lista, para cada predicado se realiza la tarea de evaluación sobre la misma instancia de entrenamiento, se establece un umbral de 0.5 para la separación de clases y se calcula el valor de verdad para cada registro de la instancia.

Tabla 4.1. Ejemplos de predicados descubiertos.

Valor de verdad	Predicado
0.780822	(IMP (AND "objeto1" "objeto2") "clase1")
0.910035	(IMP (OR "CapacidadResidual" " objeto1" " objeto2") "clase1")
0.890376	(IMP (AND "objeto1" "CapacidadResidual" "objeto2") "clase0")

Si el valor de verdad del predicado para el registro es mayor que 0.5, se asigna como resultado la clase 1, donde los objetos son empacados en el mismo contenedor.

Tabla 4.2. Ejemplo de verificación de aciertos en la clasificación de la instancia de entrenamiento.

objeto 1	Objeto 2	Capacidad residual	Clase original	valor de verdad	Clase inferida	Acierto
99	96	1	1	0.57342936	1	VERDADERO
99	95	1	1	0.56734305	1	VERDADERO
99	95	1	0	0.56734305	1	FALSO
99	91	1	1	0.56734305	1	VERDADERO
99	91	1	1	0.55381806	1	VERDADERO
99	91	1	0	0.55381806	1	FALSO
99	90	1	1	0.55015346	1	VERDADERO
99	89	1	1	0.55015346	1	VERDADERO
99	86	1	0	0.53437681	1	FALSO

De lo contrario, si el valor de verdad para el registro es menor que 0.5, se asigna el registro a la clase 0, donde los objetos no se almacenan en el mismo contenedor.

Se compara el resultado de la evaluación del predicado con los registros contra la solución dada en la instancia y se cuentan los aciertos (tabla 4.2).

El predicado que tenga más aciertos en comparación con la solución de la instancia de entrenamiento se utiliza como el clasificador del VS para la etapa de predicción.

El algoritmo 4.2 muestra el proceso de selección del predicado que opera como el modelo LDC clasificador del VS.

Algoritmo 4.2: Generador de C-LDC (dataset, config_algoritmo)

Selecciona el mejor predicado **para** ser el clasificador LDC para el VS. Extiende el algoritmo de validación cruzada con la tarea de descubrimiento

Entrada: *dataset*, conjunto de datos de entrada transformado en instancia de clasificación y la configuración del algoritmo de búsqueda de la tarea de descubrimiento de predicados.

Salida: Mejor predicado para el clasificador basado en LDC.

1. EmpacarJuntos = 1
2. predicado_simbolico = '(* IMP 1)' todos los predicados que impliquen empacar dos objetos en el mismo contenedor
3. Crear *k*=10 bloques de *dataset* para la validación cruzada
4. **para** *f*=1 **hasta** *k*
5. $tabla_E$= *dataset* menos bloque *j*
6. $tabla_P$= bloque *j* de *dataset*
7. m_f = *total de observaciones en* $tabla_P$
8. *lista_predicados = tareaDescubrimiento(*$tabla_E$*, config_algoritmo); realiza la tarea de descubrimiento con la tabla de entrenamiento y la configuración del algoritmo, retorna una lista de* n_f *predicados descubiertos*
9. $lista\_premisas_f$ = *obtenerPremisas(lista_predicados)*
10. n_f = *total de premisas en* $lista\_premisas_f$
11. **para** p = 1 **hasta** n_f
12. $premisa_{fp}$ = *premisa p en* $lista\_premisas_f$
13. *VectorVV = tareaEvaluacion(*$premisa_{fp}$*,* $tabla_P$*); realiza la tarea de evaluación con la premisa del predicado descubierto con la tabla de prueba; retorna el vector de valores de verdad*
14. $precision_{fp} = \frac{\sum_{i=1}^{mf} z_i}{m_f}$*; calcula la precisión de la premisa p del bloque f*
15. ***retornar*** *premisa con mejor precisión, precisión*

donde

$$z_i = \begin{cases} 1 & si\ observacion_{i,clase} = 1\ Y\ vectorVV_i \geq 0.5 \\ 1 & si\ observacion_{i,clase} = 0\ Y\ vectorVV_i < 0.5 \\ 0 & de\ otra\ forma \end{cases}$$

4.5 Etapa de ejecución del VS para generar soluciones

En la ejecución del VS, se proporciona como entrada una instancia no vista y codificada pero sin la solución correspondiente.

Las instancias se codifican formando combinaciones de dos objetos en los que se toma en cuenta el peso de cada uno de ellos, la capacidad residual con respecto al primer objeto para determinar si son almacenados en el mismo contenedor.

Como se muestra en la figura 4.2 la ejecución del VSLDC se realiza a través de los pasos de predicción y el de refinamiento.

4.5.1 Paso de predicción

Todos los registros u observaciones de la instancia son evaluados por los clasificadores VLC y C-LDC, que consiste en el predicado seleccionado en la etapa de aprendizaje. Esta evaluación se realiza paralela e independientemente.

Los clasificadores calculan el valor de verdad para cada observación, que actúa de manera similar al vector de probabilidades que genera la SVM (figura 4.9).

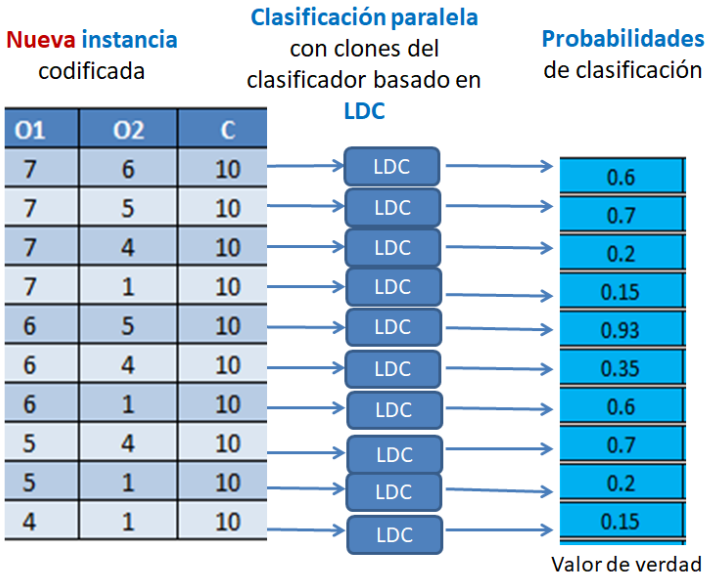

Figura 4.9. Cálculo del vector de valores de verdad generados por el clasificador.

El vector de valores de verdad es usado para generar una población de soluciones, donde cada solución es generada de la siguiente forma (figura 4.10):

Para cada observación de la instancia, se genera aleatoriamente un número fraccionario, si éste menor que el valor de verdad correspondiente en el vector, entonces se asigna un 1 a la posición del registro en la solución, en caso contrario, se asigna un cero.

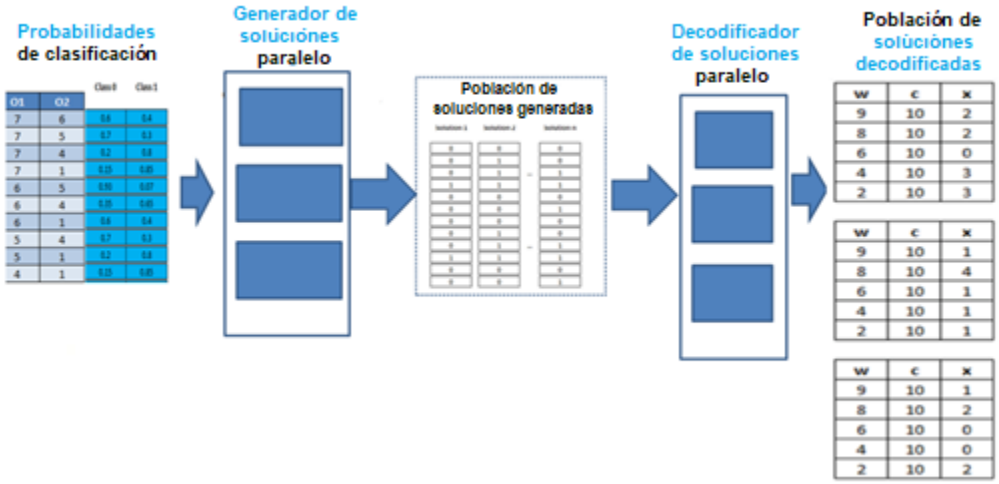

Figura 4.10. Generación de población de soluciones decodificadas.

Una vez generada la población de soluciones, éstas son decodificadas, obteniendo las instancias iniciales de optimización, que son la salida del paso de predicción.

4.5.2. Paso de refinamiento

En el paso de refinamiento tiene como entrada las soluciones decodificadas obtenidas en la etapa anterior, a las cuales se les aplica el operador de mejora para obtener una población de soluciones refinadas (Figura 4.11).

El refinamiento consiste en la aplicación de heurísticas propias del problema de empacado de objetos, tomando en cuenta las características y restricciones propias, construyendo soluciones factibles que son evaluadas calculando su aptitud y, seleccionando la que tenga la menor cantidad de contenedores como salida del VS.

Las heurísticas consideradas para el paso de refinamiento fueron el acomodo de objetos de acuerdo al primer ajuste, mejor ajuste, peor ajuste y el reacomodo por pares.

También se aplicó una modificación a la heurística reacomodo por pares, en la etapa donde ya no es posible acomodar objetos libres, y se almacenan los objetos restantes mediante la técnica FFD, reemplazando esta heurística por la BFD en esta parte del algoritmo.

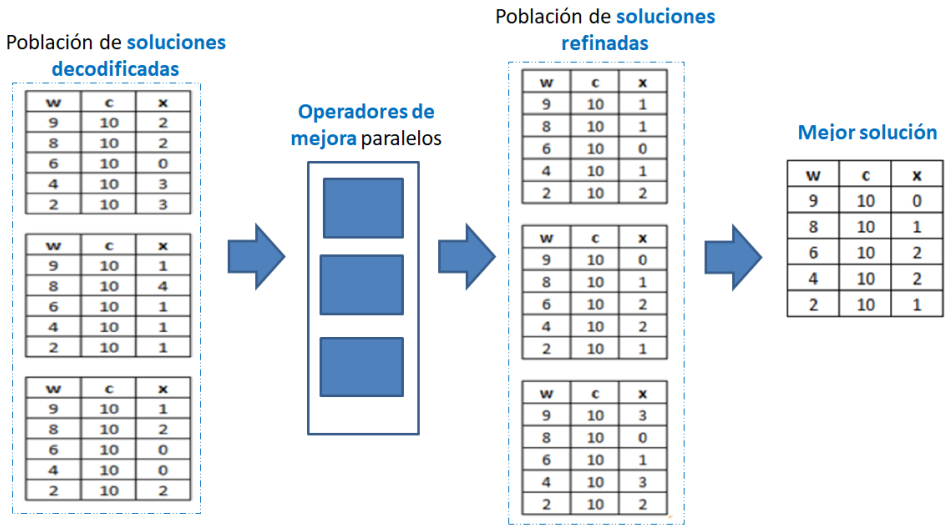

Figura 4.11. Selección de la mejor solución de la población de soluciones refinadas

Capítulo 5

Experimentación y resultados

En este capítulo se presenta un conjunto de experimentos realizados para evaluar la calidad del algoritmo VS-LDC propuesto para la solución de BPP. Los primeros experimentos son para configurar los parámetros de componentes de VS-LDC. Posteriormente se comparó experimentalmente esta propuesta basada en LDC contra la versión original de Virtual Savant, que trabaja con la SVM. En el último experimento se comparó VS-LDC con el algoritmo del estado del arte GGA-CGT.

5.1 Instancias de experimentación

Para la experimentación se utilizaron nuevas instancias creadas por Quiroz (2019), las cuales cuentan con las siguientes características:

- Las instancias fueron generadas con un óptimo conocido, dónde todos los contenedores en la solución quedan llenos al 100%.
- Para cada contenedor los pesos de los objetos fueron generados de manera aleatoria con una distribución uniforme con valores entre 1 y una proporción de la capacidad del contenedor.
- Los pesos de los objetos están en el rango de entre 1 y la cantidad indicada por el nombre de la instancia. La clase BPP C entre 1 y C, la clase, la clase BPP.25 entre 1 y 0.25C BBP .5C entre 1 y 0.5C, la clase BPP .75C entre 1 y 0.75C, donde C es la capacidad del contenedor en la instancia.

5.2 Experimentación preliminar

Para determinar la mejor configuración del VS-LDC se realizaron dos experimentos preliminares en los cuales se buscaron los parámetros para las tareas de descubrimiento dentro de los pasos de predicción y refinamiento. Estos experimentos son independientes entre sí para determinar el impacto en la solución de cada uno de ellos.

El experimento preliminar 1 (EPre1) examinó la precisión del VSLDC escalando las instancias de entrenamiento y prueba para las etapas de aprendizaje y ejecución, debido a que la SVM realiza el escalamiento en el rango de [-1, 1], por lo que se comparó esta proporción contra la obtenida escalando las instancias a un rango de [0, 1], tabla 5.1.

Tabla 5.1. Precisión del VSLDC comparando escalamientos con rangos [-1, 1] y [0, 1].

Escalamiento	**Precisión**
[0, 1]	15.00%
[-1, 1]	19.22%

El experimento preliminar 2 (EPre2) efectuó, en el paso de predicción del VSLDC, la tarea de evaluación del predicado seleccionado durante el paso de aprendizaje, para calcular los valores de verdad de cada observación en las instancias de prueba.

Los valores de verdad, que actúan como el vector de probabilidades obtenido por la SVM, pueden ser muy bajos y afectar la generación de la población de soluciones para su posterior refinamiento y ocasionar que no se llegue al óptimo.

Por esta razón se propuso escalarlos tomando el máximo valor de verdad para llevarlos a la escala de (0, 0.8), (0, 0.9) y (0, 1],

El EPre2 también contempló la cantidad de soluciones a generar de acuerdo a *n*, la cantidad de objetos en la instancia durante el paso de predicción del VS-LDC, creando una población de soluciones con *n*, 5*n*, 10*n* y 20*n* (tabla 5.2).

Tabla 5.2. Comparativa soluciones generadas por escalamiento del valor de verdad en la predicción del VS.

Escalamiento valor de verdad (TV)	Cantidad de soluciones generadas			
	n	5*n*	10*n*	20*n*
TV Sin escalar	0	0	0	0
0.8max TV	57	54	60	67
0.9max TV	58	59	68	55
max TV	55	55	55	55

n = cantidad de objetos en instancia

5.2 Aprendizaje del VS

El entrenamiento de la VS, tanto para la basada en la SVM como la basada en LDC, se realizó la propuesta de entrenar con una cantidad pequeña de instancias y se tomaron 4 instancias de prueba del dataset 0.25C, que fueron la 13, 29, 30 y 31 que fueron seleccionados de manera aleatoria con la capacidad de 10^7 unidades.

5.2.1 Entrenamiento de la SVM

Para el entrenamiento de la SVM primero se realiza la búsqueda de mejores parámetros, coste y γ (tabla 5. 3).

Tabla 5.3. Configuración del entrenamiento de la SVM.

Parámetro	Detalle
Búsqueda de parámetros	Gridsearch
Validación	10 fold cross-validation
Kernel	Radio Basis Function (RBF)
Escalamiento	[-1, 1]

Así mismo se busca la mejor configuración de la instancia de aprendizaje: el balanceo de las observaciones para las clases 0 y 1 y, su posterior muestreo para el proceso de aprendizaje.

Por ejemplo para el ajuste de parámetros, tomando en cuenta el preprocesamiento de la instancia, en la tabla 5.4 se detalla:

- La cantidad de observaciones de los registros que pertenecen a la clase 0 (no empacar los objetos en el mismo contenedor) a incluir en la instancia para balancearla.
- De la instancia balanceada, el muestreo máximo de observaciones para crear la * instancia de aprendizaje.

Los parámetros *costo* y γ

- La precisión obtenida por el modelo de SVM construido
- El tiempo que toma el proceso de aprendizaje con cada experimentación

Tabla 5.4. Resultados de entrenamiento de la SVM.

Balanceo clase 0	Tamaño máximo muestra	Costo	Gama	Precisión	Tiempo de ejecución
1977	1000	2	16	0.55063933	779.03
3954	1000	8	8	0.55861545	775.35
19770	1000	8	1	0.62035449	852.37
1977	4000	2	32	0.52720627	12722.34
3954	4000	0.25	4	0.5219085	38810.88
19770	4000	2	32	0.55484143	41975.32

5.2.2 Entrenamiento de la LDC

El entrenamiento del clasificador LDC se realizó con el núcleo científico Universe, el cual puede realizar tareas de descubrimiento y evaluación de predicados, entre otras.

La tarea de descubrimiento es realizada mediante un algoritmo genético que busca los predicados de acuerdo al predicado simbólico especificado, además de otra configuración. Así mismo puede encontrar los parámetros de funciones de pertenencia de los estados lingüísticos y, en el caso de la FPG, incluso puede definir la forma de la gráfica del estado.

El clasificador se entrenó mediante la tarea de descubrimiento de predicados, en la cual el algoritmo de búsqueda se configuró de la siguiente manera (tabla 5.5):

Tabla 5.5 Configuración del algoritmo de búsqueda para la tarea de descubrimiento de predicados.

Parámetro	Descripción	Valor
profundidad del predicado	especifica la cantidad de operadores anidados en los predicados descubiertos	2
total de iteraciones	indica la cantidad de generaciones totales para obtener los resultados	50
valor de verdad mínimo: 0.8	Indica el valor de verdad mínimo en la evaluación del predicado descubierto para ser presentado como resultado	0.8
total de resultados	indica la cantidad máxima de predicados a desplegar en los resultados, siempre que estén por encima del valor de verdad mínimo	15
porcentaje de mutación	En el algoritmo de búsqueda, es el porcentaje para mutar un predicado o estado lingüístico	5 %

Por otra parte, para los estados lingüísticos se tomó en cuenta la siguiente configuración:

- Función de membresía para todos los estados lingüísticos: FPG con búsqueda de parámetros,
- Cantidad de estados por variable: 3.

El predicado simbólico, el cual se utiliza para definir la estructura de los predicados, descubiertos fue el siguiente:

(IMP "" x)*

Donde x es el estado lingüístico de la clase, con la función de pertenencia indicando con el valor 1, si los objetos se almacenan en el mismo contenedor.

En la tabla 5.6 se muestran los predicados descubiertos y evaluados durante la etapa de entrenamiento, calculando la precisión mediante validación cruzada de 10 bloques. El formato que se muestra es el utilizado por el núcleo científico Universe.

Tabla 5.6. Predicados descubiertos durante el aprendizaje de VS.

No.	Predicado	Valor de verdad
1	(IMP (AND (AND "CresA" "CresC" "w1C" "w2A") (AND "w2C" "CresA" "w1C" "w1A") (AND "CresB" "w1B" "w2B" "w2A" "w1A") (AND "CresC" "CresA" "w1B" "w1C")) "x")	0.4782390
2	(IMP (AND (AND "w2B" "w1B" "CresB" "CresC" "w2C" "CresA") (AND "w1B" "w1C" "w2B")) "x")	0.564375
3	(IMP (AND (AND "w2C" "w1B" "w2A") (AND "w2B" "CresC" "w1B" "w2C" "w2A" "w1C" "w1A" "CresA") (AND "w1B" "w1A" "CresA")) "x")	0.512774
4	(IMP (AND (AND "w2B" "w1C" "CresA" "w1A") (OR "CresC" "CresB") (OR "CresB" "w1A") (AND "w1A" "w2B" "w2C" "CresA") (AND "w1A" "CresA" "w2B" "CresB") (AND "w1C" "w1B" "CresC" "w2B" "w2C" "CresA") (AND "w1B" "w1C" "CresC" "CresA") (AND "w2B" "w2C" "CresB" "w2A" "w1C" "CresA" "CresC" "w1A" "w1B") (AND "w1C" "CresC" "w1A" "CresB" "w1B" "CresA" "w2B" "w2A" "w2C")) "x")	0.453447
5	(IMP (AND (AND "w2A" "w1B" "w1A") (AND "w1A" "w1C") (AND "w1B" "CresA" "w2C" "w1C" "w2B" "w1A")) "x")	0.477800
6	(IMP (AND (AND "CresB" "CresC" "w1C" "w1B") (AND "w2A" "CresB" "w1B" "CresA")) "x")	0.481683
7	(IMP (AND (AND "w1A" "w1B" "w2B" "CresC") (AND "CresC" "CresB" "w2B" "w2C")) "x")	0.440961
8	(IMP (AND (AND "w2C" "w1B" "w2B" "CresA") (AND "CresA" "w2A" "w2C" "w2B") (AND "w2C" "w1B" "w2B")) "x")	0.525561
9	(IMP (AND (AND "w2A" "w1C" "CresA" "w2C" "CresC" "w1B" "w1A" "w2B" "CresB") (AND "w1C" "w1A") (AND "CresB" "w2A")) "x")	0.469952
10	(IMP (AND (AND "w2B" "CresA" "CresB" "w2A" "w1C" "w1B") (AND "w2C" "w2A" "CresA" "CresC" "w2B")) "x")	0.428761

De los predicados descubiertos en la etapa de aprendizaje, se toma el segundo como el modelo de clasificación para la etapa de prueba, tomando en cuenta la precisión obtenida en la validación cruzada, el árbol de este predicado difuso se muestra en la figura 5.1.

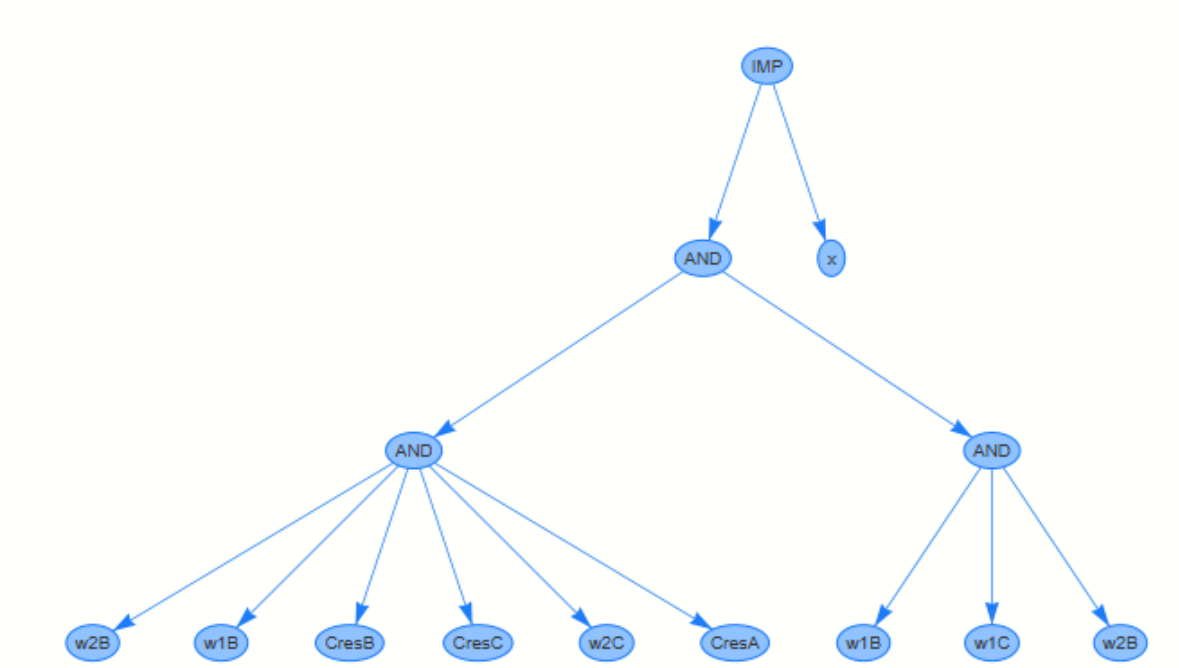

Figura 5.1. Representación en árbol del predicado difuso usado como clasificador del VS.

En la tabla 5.6 se muestran los parámetros encontrados por la tarea de descubrimiento, correspondientes a las variables en las instancias de prueba.

Tabla 5.7. Parámetros de los estados lingüísticos del predicado difuso seleccionado como clasificador del VS.

Estado	Columna	Beta	Gama	M
CresA	Cres	0.854733	0.762494	0.40637106
w1B	w1	0.228085	0.019625	0.92028355
w1C	w1	0.116405	0.005636	0.93197547
w2B	w2	0.227532	0.021588	0.98215245
w2C	w2	0.197776	0.014734	0.60007005
CresB	Cres	0.911325	0.765199	0.84003482
CresC	Cres	0.997114	0.783664	0.99318477

En las figuras 5.2 a 5.4, se muestran las gráficas de los pesos de los objetos y la capacidad residual ocasionada por el primer objeto, obtenidos por la tarea de descubrimiento.

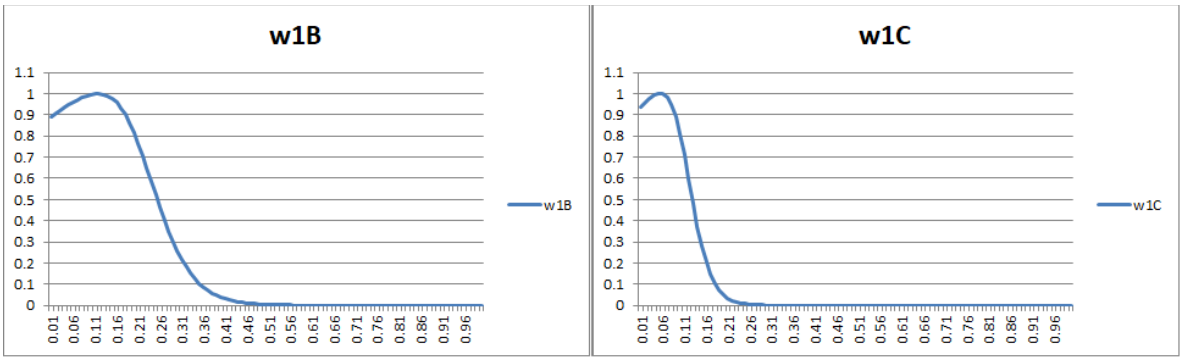

Figura 5.2. Gráficas de los estados lingüísticos w1B y w1C

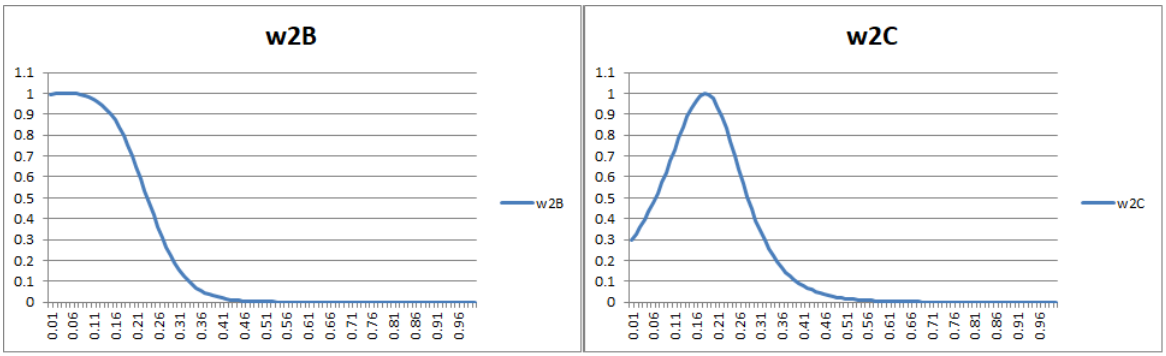

Figura 5.3. Gráficas de los estados lingüísticos w1B y w1C

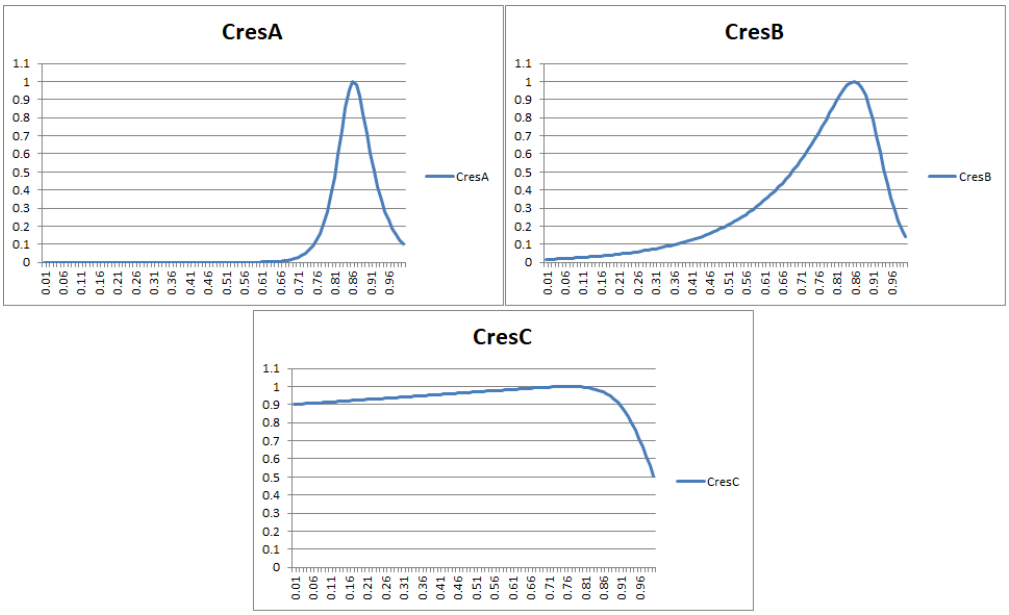

Figura 5.4. Gráficas de los estados lingüísticos de la capacidad residual.

5.2.3 Comparación de clasificadores SVM con LDC

En la tabla 5.8 se muestra un resumen de la configuración para la búsqueda de parámetros y construcción del modelo de la SVM y configuración del algoritmo de búsqueda para la tarea de descubrimiento de predicados, así como el cálculo de la precisión por medio de la validación cruzada de 10 bloques.

Tabla 5.8. Comparativa de clasificadores SVM contra LDC.

SVM	LDC
Búsqueda de parámetros: **GridSearch** Validación: **10 fold cross-validation** Kernel: **Radio Basis Function (RBF)** Escalamiento: **[-1, 1]**	Búsqueda: **Tarea de descubrimiento** Profundidad de predicados: **2 niveles** Estados x variable: **3 estados** Selección de mejor predicado: **10-fold cross-validation**
Precisión del clasificador: **62 %** Tiempo entrenamiento: **22 hrs 30 mins**	Precisión del clasificador: **56 %** Tiempo entrenamiento: **6 hrs 15 mins**

5.3 Etapa de ejecución del VS

Para la etapa de prueba del VS se tomaron las instancias 32-61 del dataset BPP.25, con una capacidad de 10^7 unidades y la ejecución fue con la VS-SVM y la VS-LCD con el modelo y el predicado descubierto en la etapa de aprendizaje, respectivamente. Los resultados preliminares con estas instancias se presentan en las tablas 5.9 y 5.10, en la cual se incluye para cada instancia:

- el identificador de la instancia,
- la cantidad de objetos incluidos (*n*),
- la capacidad del contenedor y,
- para cada algoritmo: el óptimo obtenido (*m*) y el tiempo de ejecución, en segundos.

Tabla 5.9. Comparación del VSSVM y VSLDC con el GGA-CGT.

Instancia	***n***	***c***	**GGA-CGT**		**VS LDC**		**VS SVM**	
			m	**tiempo**	***m***	**tiempo**	***m***	**tiempo**
32	128	10^7	15	235.040	16	2.924	16	1.924
33	131	10^7	15	1001.230	16	2.239	16	1.239
34	154	10^7	15	675.000	16	2.668	16	1.668
35	134	10^7	15	614.640	16	2.229	16	1.229
36	118	10^7	15	626.030	16	1.959	16	0.959
37	129	10^7	15	575.830	16	2.119	16	1.119
38	124	10^7	15	825.770	16	2.063	16	1.063
39	127	10^7	15	2504.360	16	2.112	16	1.112
40	138	10^7	15	903.260	16	2.262	16	1.262
41	129	10^7	15	171.190	16	2.118	16	1.118
42	135	10^7	15	207.260	16	2.342	16	1.342
43	143	10^7	15	1175.780	15	2.339	16	1.339
44	135	10^7	15	822.310	16	2.232	16	1.232
45	134	10^7	15	1547.560	16	2.189	16	1.189
46	135	10^7	15	795.180	16	2.221	16	1.221
47	120	10^7	15	384.770	16	1.984	16	0.984

Tabla 5.10. Comparación del VSSVM y VSLDC con el GGA-CGT (continuación).

Instancia	*n*	*c*	GGA-CGT		VS LDC		VS SVM	
			m	**tiempo**	***m***	**tiempo**	***m***	**tiempo**
48	124	10^7	15	7985.360	16	2.004	16	1.004
49	128	10^7	15	383.690	16	2.077	16	1.077
50	122	10^7	15	3612.790	16	1.995	16	0.995
51	139	10^7	15	614.010	16	2.23	16	1.23
52	123	10^7	15	259.360	16	2.093	16	1.093
53	135	10^7	15	1328.710	16	2.23	16	1.23
54	119	10^7	15	34534.070	16	1.989	16	0.989
55	129	10^7	15	1091.610	16	2.095	16	1.095
56	137	10^7	15	2251.980	15	2.378	15	1.378
57	135	10^7	15	680.680	16	2.208	16	1.208
58	128	10^7	15	2180.140	16	2.114	16	1.114
59	126	10^7	15	778.530	16	2.043	16	1.043
60	129	10^7	15	13032.840	16	2.098	16	1.098
61	126	10^7	15	13032.840	16	2.06	16	1.06

5.4 Resultados

Los resultados parciales muestran que en la experimentación preliminar para el VSLDC se obtiene una mejor precisión con el escalamiento [0, 1] que con el rango [-1, 1] por 4.22 % (tabla 5.1).

Por otra parte, la generación de soluciones para el paso de predicción, así como el escalamiento del valor de verdad en la evaluación del predicado, impacta en la mejora de la precisión en el aprendizaje y ejecución del VSLDC, de acuerdo con la tabla 5.2, donde se muestra que la combinación del escalamiento 0.8maxTV con *20n* soluciones, donde *n* es el número de objetos en la instancia, es la mejor combinación entre estos factores.

Una vez entrenado, el VSSVM y el VSLDC con las instancias 13, 29. 30 y 31, se comparó con el algoritmo GGA-CGT utilizando SVM y LDC como clasificadores, con el VSSVM se obtiene una mejor clasificación con respecto al VSLDC mejorando hasta un 6 % la precisión (tabla 5.7)

En la ejecución del VS se utilizaron las instancias 32 a 61 para pruebas, donde se observa que el VSLDC mejora ligeramente el rendimiento por 0.22 %, donde el clasificador basado en LDC tuvo un error del 6.22 % mientras que la SVM del 6.44 %.

Capítulo 6

6. Conclusiones y trabajo futuro

6.1 Conclusiones

En esta tesis se cumple el objetivo de desarrollar un método que genere optimizadores paralelos para la solución de problemas de empacado de objeto en contenedores (BPP), usando técnicas de aprendizaje automático para modelar la capacidad de solución de un algoritmo de referencia visto como caja negra.

Para ello, se analizó de manera teórica y práctica el paradigma Virtual Savant (VS) que genera clasificadores basados en máquinas de soporte vectorial. Estos clasificadores aprendidos se usan como optimizadores que resuelven de manera automática y en paralelo instancias no vistas y de mayor complejidad.

En particular, se desarrolló un nuevo método basado en el paradigma VS, incorporando lógica difusa compensatoria (VSLDC). Además, para aplicar VS-LDC a BPP se desarrollaron nuevas estrategias de transformación, clasificación y refinamiento.

Para la clasificación se reemplaza a la máquina de soporte vectorial (SVM) por un clasificador basado en lógica difusa compensatoria (LDC). Primero, instancias de BPP con solución conocida se *transforman* en instancias de clasificación, y se aprenden *clasificadores* a partir de ellas. Después, los clasificadores aprendidos producen probabilidades de asignación a clases para instancias no vistas, lo que permiten construir

soluciones. Para el refinamiento se propone una heurística que mejora las soluciones y obtiene la mejor por aptitud. Esta es la heurística reacomodo por pares, reemplazando el acomodo First Fit (FF) por el acomodo Best Fit (BF).

Para evaluar la calidad de VS-LDC se realizó un conjunto de experimentos. En el primero se comparó experimentalmente esta propuesta basada en LDC contra la versión original de Virtual Savant, que trabaja con la SVM. En el segundo experimento se comparó VSLDC con el algoritmo del estado del arte GGA-CGT.

6.2 Aportaciones

En este proyecto de tesis se cumplieron con los objetivos particulares propuestos:

1. Desarrollar una estrategia de solución de problemas NP-duros limitados por las capacidades del hardware para problemas de agrupamiento.

   En la sección 4.1 se detalla la propuesta de solución, en la que se indica que un problema de optimización se convirtió a uno de clasificación para utilizar la estrategia desarrollada por Pinel (2014), es decir, la utilización de clasificadores paralelos para generar una población de soluciones y aplicarles operadores de mejora para encontrar la mejor solución al problema de BPP.

2. Analizar de manera teórica y práctica el método que genera optimizadores paralelos VS que se basa en máquinas de soporte vectorial.

   En la sección 2.8 se revisó y analizó el paradigma VS y su funcionamiento, desde la conversión del problema de optimización a uno de clasificación. Cada observación de la instancia es tomada por un clasificador paralelo basado en la máquina de soporte vectorial (SVM).

   La salida de la SVM es un vector de probabilidades que se utiliza para generar una población de soluciones, posteriormente, se les aplica un operador de mejora, en este caso reordenamiento por pares, para tomar la mejor solución como la salida del VS de acuerdo a su función de aptitud.

3. Adaptar un módulo de lógica difusa compensatoria como clasificador para Virtual Savant.

   La sección 4.1 describe cómo se incluirán clasificadores paralelos basados en LDC para reemplazar la SVM. La sección 2.9 introduce la definición de lógica difusa compensatoria así como sus axiomas, además de los predicados difusos, los estados lingüísticos y funciones de pertenencia, entre otros.

   Además, en la sección 4.4.2 se describe la etapa de entrenamiento del clasificador basado en LDC para obtener un predicado difuso que es usado dentro del VS para la generación del vector de probabilidades y de la población de soluciones.

4. Adaptar heurísticas de búsqueda local del problema de empacado de objetos para su incorporación en Virtual Savant.

   En la sección 4.5.2 se indica que se utilizó la heurística reacomodo por pares modificándola para que en la segunda parte del algoritmo se reemplace la técnica de primer ajuste descendiente (FFD) por la heurística mejor ajuste descendente (BFD).

5. Usar un clasificador basado en lógica difusa compensatoria y comparar contra el clasificador usado actualmente por el Virtual Savant: la máquina de soporte vectorial.

   La sección 5.2.3 muestra una comparación de los clasificadores basados en la SVM contra el basado en LDC donde se aprecia que la precisión de la SVM es del 62 % contra la precisión de la LDC como clasificador, que es del 56 %, con una diferencia de 6 % entre ambos; por otra parte, el tiempo que toma el entrenamiento de la LDC es de 6 horas con 15 minutos contra 22 horas con 30 minutos de la SVM.

6. Evaluar ventajas y desventajas de la lógica difusa compensatoria como clasificador.

   De acuerdo con la sección 5.2.3, las ventajas que presenta la LDC con respecto a la SVM son: el tiempo de entrenamiento de la LDC es mucho menor que el de la SVM y el proceso, la SVM requiere de una etapa adicional en el inicio, que es el ajuste de parámetros.

Por otra parte, las desventajas de la LDC son:

- Se requiere hacer un proceso de normalización para adaptar el proceso a cualquier instancia con capacidad de contenedor distinta.
- Se requiere escalar el valor de verdad obtenido para el vector de probabilidades para mejorar la precisión en la clasificación, como se explica en la sección 5.1
- En la sección 5.2.3 se observa que la precisión de clasificación de la LDC es menor por 6 % con respecto a la SVM

7. Evaluar la influencia de diferentes clasificadores.

   En este proyecto se investigó con los clasificadores SVM y LDC para la solución del BPP, para los cuales se realizó una comparación en la predicción de observaciones de instancias transformadas, tanto en la etapa de aprendizaje como en la de ejecución. Aunque en la primera etapa la SVM resultó con una precisión 6 % más alta que la LDC, en la etapa de ejecución la diferencia no es estadísticamente significativa, siendo la VS-LDC 0.022 % más precisa que la VS-SVM que incluye una etapa de mejora de las soluciones obtenidas.

8. Aplicar la metodología propuesta en instancias no vistas y de mayor tamaño o en instancias difíciles de resolver.

   El entrenamiento para el VS basado en LDC y el basado en la SVM se realizó con cuatro instancias transformadas, mientras que en la etapa de prueba se utilizaron 30 instancias del dataset 25C, que es descrito en las secciones 5.2 y 5.3 de este documento.

9. Comparar experimentalmente la propuesta basada en lógica difusa compensatoria contra la versión original de Virtual Savant.

   En la sección de resultados 5.4, se observa que el VSLDC mejora ligeramente el rendimiento por 0.22 %, donde el clasificador basado en LDC tuvo un error del 6.22 % mientras que la SVM del 6.44 %.

6.3 Trabajo futuro

Queda como trabajo futuro la solución de instancias de mayor tamaño, ya que sólo se experimentó con instancias de mayor complejidad o llamadas difíciles de resolver.

Desarrollo y experimentación de otros problemas de agrupación, como lo es el BPP, con el paradigma VS como los siguientes:

- Agrupamiento homogéneo de elementos con múltiples atributos: El problema general de agrupamiento, particularmente aquel en el que se busca conformar grupos de igual tamaño y equitativos respecto a más de un atributo (Moreno, 2011).
- CVRP(Capacitated Vehicle Routing Problem): Es el VRP más general y consiste en uno o varios vehículos con capacidad limitada y constante encargados de distribuir los productos según la demanda de los clientes. Este problema de optimización del tipo NP-Hard, combina las características de un (BPP), con el objetivo de asignar las cargas a los vehículos capacitados, y un problema del agente viajero (TSP) que apunta a encontrar la mejor ruta para cada vehículo (citado en Cardozo, 2013).

Además de explorar con esta misma metodología de investigación:

- Otras heurísticas de refinación además de First Fit Descending (FFD), BFD (Best fit Descending) o reagrupamiento por pares descritas en este proyecto.
- Analizar el desempeño de otras técnicas de clasificación, además de la basada en LDC.
- Proponer otros métodos de transformación de instancias de optimización.
- Resolver otros tipos de instancias, incluyendo la generación de nuevos tipos en los que se establezca correlaciones de objetos, capacidad y otros criterios de caracterización de las mismas.
- Además de generar optimizadores por grado de correlación de las instancias, considerar otros criterios de caracterización de las mismas.
- Experimentar con instancias muy grandes.

6.4 Difusión de la investigación

La difusión de esta investigación se realizó mediante la publicación del capítulo

"Use of Compensatory Fuzzy Logic for Knowledge Discovery Applied to the Warehouse Order Picking Problem for Real-Time Order Batching" en el libro "Handbook of research on metaheuristics for order picking optimization in warehouses to smart cities" (Padrón-Tristán et al., 2019).

Este capítulo de libro se expuso en el "6º encuentro de jóvenes investigadores de Tamaulipas" en noviembre de 2019

Además se encuentra en proceso de publicación el artículo "Application of compensatory fuzzy logic in diabetes problem using PIMA Indian Dataset" (Padrón-Tristán et al., 2020), el cuál además fue expuesto en el taller "International Virtual Workshop of Business Analytics Eureka 2019".

Bibliografía

(Abraham, 2015) Abraham, G. T., James, A., Yaacob, N., (2015) Group-based parallel multi-scheduler for grid computing. Future Generation Computer Systems 50, 140–153.

(Albon, 2017) Albon, C., (2017) SVC Parameters When Using RBF Kernel.

(Alvim, 2004) Alvim, A. C., Ribeiro, C. C., Glover, F., & Aloise, D. J. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. Journal of Heuristics, 10(2), 205-229. (Alvim, 2004) Alvim, A. C., Ribeiro, C. C., Glover, F., & Aloise, D. J. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. Journal of Heuristics, 10(2), 205-229.

(Bagnall, 2001) Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The next release problem. Information and software technology, 43(14), 883-890.

(Barney, 2010) Barney, B. (2010). Introduction to parallel computing. Lawrence Livermore National Laboratory, 6(13), 10.

(Cardozo, 2013) Cardozo, J. P. O. (2013). Solución al problema de ruteo de vehículos con capacidad limitada" CVRP" a través de la heurística de barrido y la implementación del algoritmo genético de Chu-beasley (Doctoral dissertation, Universidad Tecnológica de Pereira. Facultad de Ingeniería Industrial. Ingeniería Industrial).

(Cejas, 2011) Cejas, J. (2011). La lógica difusa compensatoria/The compensatory fuzzy logic. Ingeniería Industrial, 32(2).

(Cheruku, 2017) Cheruku, R., Edla, D. R., & Kuppili, V. (2017). SM-RuleMiner: Spider monkey based rule miner using novel fitness function for diabetes classification. Computers in biology and medicine, 81, 79-92.

(Coffman et al, 2002) Coffman, E. G., Courcoubetis, C., Garey, M. R., Johnson, D. S., Shor, P. W., Weber, R. R., Yannakakis, M.: Perfect Packing Theorems and the

Average Case Behavior of Optimal and Online Bin Packing. SIAM Review Vol. 44 (2002).

(De Vito, 2006) De Vito, E. L., Eduardo, C. (2006). Introducción al razonamiento aproximado: lógica difusa. Revista Americana de Medicina Respiratoria, 6(3), 126-136.

(Dean, 2008) Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM.

(Díaz et al, 1996) Díaz, A., González, J. L., Laguna, M., Mascato, P., Tseng, F. T.,

(Espin, 2009) Espin, R. (2009) Manual técnico de Universe.

(Dorronsoro, 2010) Dorronsoro, B., Bouvry, P., 2010. A new parallel asynchronous cellular genetic algorithm for scheduling in grids. In: IEEE Int. Symp. on Parallel & Distributed Processing. IEEE, p. 206b.

(Falkenauer, 1996) Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. Journal of heuristics, 2(1), 5-30. (Falkenauer, 1996) Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. Journal of heuristics, 2(1), 5-30.

(Fleszar, 2002) Fleszar, K., & Hindi, K. S. (2002). New heuristics for one-dimensional bin-packing. Computers & operations research, 29(7), 821-839.

(Fleszar, 2011) Fleszar, K., & Charalambous, C. (2011). Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. European Journal of Operational Research, 210 (2), 176-184.

(Fonseca, 2016) Fonseca, A., Cabral, B., Rafael, J., Correia, I. (2016). Automatic parallelization: Executing sequential programs on a task-based parallel runtime. International Journal of Parallel Programming, 1–22.

(Fuentes, 2011) Humberto Fuentes Saenz (2011) Data mining framework for batching orders in realtime warehouse operations Iowa State University.

(Galindo 2017) Galindo, J. (2007). Conjuntos y sistemas difusos (Lógica difusa y aplicaciones). ETSI Informática.

(Glover, 2006) Glover, F., Ghaziri, H. M. (1996) Optimización Heurística y Redes Neuronales. Editorial Parainfo, España.

(Guerrero-Treviño, 2007) Treviño, I. V. H. G. (2007). Muestreo Estadístico Aplicado al Estudio del Desempeño de Algoritmos Heurísticos (Doctoral dissertation, Instituto Tecnológico de Ciudad Madero).

Gupta, J. N., & Ho, J. C. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. Production planning & control, 10(6), 598-603. Gupta, J. N., & Ho, J. C. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. Production planning & control, 10(6), 598-603.

(Harman, 2014) Harman, M., Krinke, J., Medina-Bulo, I., Palomo-Lozano, F., Ren, J., & Yoo, S. (2014). Exact scalable sensitivity analysis for the next release problem. ACM Transactions on Software Engineering and Methodology (TOSEM), 23(2), 1-31.

(Hayashi, 2016) Hayashi, Y., & Yukita, S. (2016). Rule extraction using Recursive-Rule extraction algorithm with J48graft combined with sampling selection techniques for the diagnosis of type 2 diabetes mellitus in the Pima Indian dataset. Informatics in Medicine Unlocked, 2, 92-104.

(Herrera, 2006) Herrera, F. (2006), Introducción a los algoritmos metaheurísticos. Ciencias de la Computación e IA.

(Hsu, 2003) Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). A practical guide to support vector classification.

(Ibarra, 1977) Ibarra, O. H., Kim, C. E., 1977. Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM 24 (2), 280–289.

(Jankovic, 2013) Mladen Jankovic (2013), Implementation of Hybrid Grouping Genetic Algorithm for one dimensional bin packing problem.

(López, 2004) López, A.R. (2004) Fundamentos de los computadores. Recuperado de: http://www.uhu.es/rafael.lopezahumada/descargas/tema3_fund_0405.pdf

(Maheswaran, 1999) Maheswaran, M., Ali, S., Siegel, H., Hensgen, D., Freund, R. F., 1999. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings of the Heterogeneous Computing Workshop. pp. 30–44.

(Mannila, 1996) Mannila, H. (1996). Data mining: machine learning, statistics, and databases.

Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. Discrete applied mathematics, 28(1), 59-70. Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. Discrete applied mathematics, 28(1), 59-70.

(Massobrio, 2018) Massobrio, R. (2018) Savant Virtual: generación automática de programas.

(Meschino, 2015) Meschino, G. J., Comas, D. S., Ballarin, V. L., Scandurra, A. G., & Passoni, L. I. (2015). Automatic design of interpretable fuzzy predicate systems for clustering using self-organizing maps. Neurocomputing, 147, 47-59.

(Morcillo, 2011) Morcillo, C. (2011). Lógica Difusa. Una introducción práctica. 2011, de Escuela Superior de Informática Sitio web: http://www.esi.uclm.es/www/cglez/downloads/docencia/2011_Softcomputing/LogicaDifusa.pdf.

(Moreno, 2011) Moreno, J., Rivera, J. C., & CEBALLOS, Y. F. (2011). Agrupamiento homogéneo de elementos con múltiples atributos mediante algoritmos genéticos. Dyna, 78(165), 246-254.

(Nesmachnow, 2010) Nesmachnow, S., Ares, G., & Aplicado, G. D. P. P. (2010) Computación de Alta Performance.

(Padron-Tristan, 2017) Padron-Tristan J.F., Sistema inteligente para descubrimiento de conocimiento basado en lógica difusa compensatoria, Tesis. Instituto Tecnológico de Ciudad Madero.

(Padron-Tristan et al., 2019) Cruz-Reyes L., Espin-Andrade R.A., Padrón-Tristán J.F., Martinez-Vega D.A., Llorente-Peralta C. E., Medina-Trejo C., (2019) Use of Compensatory Fuzzy Logic for Knowledge Discovery Applied to the Warehouse Order Picking Problem for Real-Time Order Batching, National Mexican Institute of Technology/Ciudad Madero, Institute of Technology, México/Autonomous University of Nuevo León, México.

(Padron-Tristan et al., 2020) Cruz-Reyes L., Espin-Andrade R.A., Padrón-Tristán J.F., Martinez-Vega D.A., Llorente-Peralta C. E., Medina-Trejo C., (2020) Application of compensatory fuzzy logic in diabetes problem using PIMA Indian Dataset. National Mexican Institute of Technology/Ciudad Madero Institute of Technology, México, Autonomous University of Nuevo León, México. En proceso de publicación

(Palomo-Lozano, 2016) Palomo Lozano, F., Del Águila Cano, I. M., & Medina Bulo, I. (2016). Un algoritmo híbrido para el problema NRP con interdependencias.

(PIMA, 2016) PIMA. (6/10/2016). Pima Indian dataset [pima], dataset perteneciente al National Institute of Diabetes and Digestive and Kidney Diseases: https://www.niddk.nih.gov/, Recuperado de https://www.kaggle.com/uciml/pima-indians-diabetes-database (PIMA, 2016) PIMA. (6/10/2016). Pima Indian dataset [pima], dataset perteneciente al National Institute of Diabetes and Digestive and Kidney Diseases: https://www.niddk.nih.gov/, Recuperado de https://www.kaggle.com/uciml/pima-indians-diabetes-database

(Pinel, 2013) Pinel, F., Dorronsoro, B., Bouvry, P., 2013. Solving very large instances of the scheduling of independent tasks problem on the GPU. Journal of Parallel and Distributed Computing 73 (1), 101–110.

(Pinel, 2014) Pinel, F. (2014). Energy-Performance Optimization for the Cloud (Doctoral dissertation, University of Luxembourg, Luxembourg).

(Quiroz, 2014) Quiroz-Castellanos, M. (2014), “Caracterización del Proceso de Optimización de Algoritmos Heurísticos Aplicados al Problema de Empacado de Objetos en Contenedores”, PhD in Computer Science, Instituto Tecnológico de Ciudad Madero.

(Quiroz. 2015) Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez, C., Huacuja, H. J. F., & Alvim, A. C. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem. Computers & Operations Research, 55, 52-64.

(Röglin, 2010) Röglin, H. (2010), Pearls of algorithms, recuperado de: http://www.roeglin.org/teaching/WS2010/Pearls/LectureNotes.pdf.

(Ruben, 1999) Ruben, R. A., & Jacobs, F. R. (1999). Batch construction heuristics and storage assignment strategies for walk/ride and pick systems. Management Science, 45(4), 575-596.

Saenz, H. F. (2011). Data mining framework for batching orders in real-time warehouse operations (master’s thesis). Iowa State University Saenz, H. F. (2011). Data mining framework for batching orders in real-time warehouse operations (master’s thesis). Iowa State University

(Scholl, 1997) 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.

(Tabak, 2014) Tabak, E., Cambazoglu, B., Aykanat, C., 2014. Improving the performance of independent task assignment heuristics MinMin, MaxMin and Sufferage. IEEE Tr. Par. Distr. Syst. 25 (5), 1244–1256.