UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA CIVIL INDUSTRIAL

UN ALGORITMO BASADO EN MACHINE LEARNING PARA EL PROBLEMA POLINOMIAL ROBUSTO DE LA MOCHILA

Por: José Ignacio González Cortés

Memoria de titulo presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Industrial

Diciembre 2023

Profesor Guía: Carlos Contreras Bolton

Dedicatoria

AGRADECIMIENTOS

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Resumen

Keywords – Knapsack Problem

Abstract

Keywords – Knapsack Problem

Índice general

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

En este capítulo se presentan los antecedentes generales acerca del Problema polinomial robusto de la mochila, las variantes de las cuales se deriva y metodologías de solución asociadas a estas. Los objetivos generales y específicos de esta memoria y la estructura del documento.

1.1. Antecedentes generales

El problema de la mochila (KP, por sus siglas en inglés, Knapsack Problem) es un problema clásico de la investigación de operaciones. Generalmente, el KP modela la necesidad de elegir un conjunto de elementos con costos y beneficios individuales, con una restricción de capacidad máxima, con el fin de maximizar el beneficio. El KP ha sido ampliamente estudiado por su estructura sencilla, y también debido a que muchos otros problemas de optimización más complejos lo tienen como subproblema (Martello and Toth, 1990).

El KP tiene muchas variantes, una ellas es la versión robusta, RKP (por sus siglas en inglés, Robust Knapsack Problem). Este fue formulado originalmente por Eilon (1987) para resolver problemas de asignación de presupuesto con aplicaciones reales, muchos de los parámetros del problema están asociados a incertidumbre. El RKP fue planteado para encontrar soluciones que sean factibles para todas las posibles variaciones en los costos de elementos (Monaci et al., 2013).

Otra variante es la versión polinomial de la mochila (PKP, por sus siglas en inglés, Polynomial Knapsack Problem) que incluye el concepto de sinergias, es decir, que la elección de una o más alternativas específicas otorga un beneficio o costo extra según estas

relaciones. El PKP sirve para modelar sistemas cuyas alternativas presentan conflictos entre ellas, o que cooperan para generar mayor beneficio (Baldo et al., 2023).Así, de este problema surge el polinomial robusto (PRKP, por sus siglas en inglés, Polynomial Robust Knapsack Problem). El PRKP toma en cuenta parámetros inciertos y sinergias polinomiales para modelar problemas de selección de alternativas, que se perjudican o benefician entre sí y además muestran comportamiento estocástico.

Debido a la complejidad espacial del PRKP, se han explorado aplicaciones del problema cuadrático de la mochila (QKP, por sus siglas en inglés, Quadratic Knapsack Problem) (Gallo et al., 1980) y el problema cúbico de la mochila (CKP, por sus siglas en inglés, Cubic Knapsack Problem) (Forrester and Waddell, 2022) El QKP presenta sinergias entre dos elementos y ha demostrado ser útil en un gran espectro de aplicaciones como posicionamiento satelital (Witzgall, 1975), localizaciones de centros de transporte como aeropuertos, ferrocarriles o terminales de carga (Rhys, 1970). El CKP es extendido desde el QKP y considera sinergias hasta con tres elementos. Además, posee aplicaciones como en el problema de satisfacción Max 3-SAT (Kofler et al., 2014), el problema de selección (Gallo et al., 1989), el problema de alineación de redes (Mohammadi et al., 2017), y la detección y tratamiento de enfermedades de transmisión sexual (Zhao, 2008).

Este trabajo propone la exploración de técnicas de machine learning para resolver el PRKP, encontrar una metodología competitiva con aquellas planteadas en la literatura y obtener resultados comparables.

1.2. Objetivos

Objetivo general

Implementar una heurística basada en machine learning para resolver el PRKP.

Objetivos específicos

- Revisar la literatura relacionada con problemas de la mochila similares y metodologías aplicables.
- Diseñar una heurística basada en machine learning para el PRKP.
- Implementar la heurística propuesta basada en machine learning.
- Evaluar los resultados y comparar el rendimiento con las metodologías expuestas

anteriormente desde la literatura.

1.3. Estructura del documento

El documento consta de 6 capítulos, en los cuales se discutirá, el modelo matemático representando el problema (2), la metodología propuesta para resolverlo (3), un análisis comparativo de los resultados experimentales (4), y finalmente una revisión y discusión para concluir el documento. (5).

Capítulo 2

Problema Polinomial Robusto de la mochila

Este capítulo comprende la definición formal del problema junto al modelo matemático de programación entera. Además, explora literatura asociada a técnicas y metodologías empleadas para la resolución de otros problemas de la mochila que pueden ser extendidos al PRKP.

2.1. Descripción del problema

El PRKP se define formalmente como un conjunto de elementos $I = \{1, 2, \ldots, N\}$ donde estos tienen un beneficio asociado $P_i \in \mathbb{R} : i \in I$. Además, los elementos poseen un coste de comportamiento estocástico que puede variar de forma continua entre una cota inferior y una superior, $C_i \in [LC_i, UC_i]$, donde LC_i es el costo base, y UC_i el costo máximo. De forma aleatoria solo algunos elementos tienen comportamiento estocástico, aquellos que no, cumplen que $C_i = LC_i$. El parámetro que define cuantos elementos de I pueden tener comportamiento estocástico es $\Gamma \leq N$, es decir $|\{i : C_i > LC_i\}| \leq \Gamma$. Además existe un presupuesto máximo W para los costes.

El conjunto de sinergias polinomiales se define como $S = \{A : A \subseteq I \wedge |A| > 1\}$. Para cada subconjunto de elementos de I, existe un beneficio asociado PS_A.

El objetivo del problema es encontrar un subconjunto de elementos $X \subseteq I$ que maximice el beneficio total de los elementos de X sumado a los beneficios de sinergias, que aplican cuando $A \subseteq X$, y que además, para cualquier variación estocástica en costes de los

elementos.

Ejemplo de PRKP

$$I = \{1, 2, 3\}$$

$$W = 10.0$$

$$\Gamma = 2$$

i	1	2	3
PS_i	3.0	6.0	10.0
LC_i	2.0	3.0	4.0
UC_i	5.0	4.0	7.0

Tabla 1: Beneficios y costos de los ítems

A	PS_A
$\{1, 2\}$	1.0
$\{1, 3\}$	2.0
$\{2, 3\}$	-1.0
$\{1, 2, 3\}$	3.0

Tabla 2: Beneficios por sinergias

2.2. Modelo matemático

Baldo et al. (2023) describe la formulación del modelo matemático linealizado del PRKP compatible con solvers exactos de programación lineal. El modelo se describe a continuación.

2.2.1. Parámetros

- I, El conjunto de elementos posibles, de cardinalidad N
- P_i, El beneficio asociado al elemento i
- LC_i, El coste nominal del elemento i
- UC_i, El coste máximo del elemento i
- W, El presupuesto o coste máximo asociado al problema.

- S, El conjunto de sinergias polinomiales.
- PS_A, El beneficio de la sinergia A
- Γ, El número máximo de elementos que pueden variar de su costo base.

2.2.2. Variables de decisión:

$$x_i = \begin{cases} 1 & \text{si } i \in X \\ \\ 0 & \text{si } i \notin X \end{cases} \tag{1}$$

$$\mathcal{Z}_A = \begin{cases} 1 & \text{si } i \in A, \forall i \in X \\ 0 & \text{si } \exists i \in A : i \notin X \end{cases} \tag{2}$$

$$\rho, \pi_i \in \mathbb{R}^+ \tag{3}$$

2.2.3. Función objetivo

$$\text{Maximizar} \sum_i^I P_i x_i + \sum_{A \in S} PS_A \cdot \mathcal{Z}_A - \sum_i^I LC_i x_i - \left(\Gamma \rho + \sum_i^I \pi_i \right) \tag{4}$$

2.2.4. Restricciones

$$\sum_i^I LC_i x_i + \Gamma \rho + \sum_i^I \pi_i \leq W \tag{5}$$

$$\rho + \pi_i \geq (UC_i - LC_i) \cdot x_i \qquad \forall i \in I \tag{6}$$

$$\sum_{i \in A} x_i \leq |A| - 1 + Z_A \qquad \forall A \in S : PS_A < 0 \tag{7}$$

$$Z_A \leq x_i \qquad \forall i \in A \in S : PS_A > 0 \tag{8}$$

$$x_i \in 0, 1, \forall i \in I, \quad Z_A \in \{0, 1\} \forall A \in S \tag{9}$$

$$\rho \geq 0, \quad \pi_i \geq 0 \forall i \in I \tag{10}$$

La función objetivo en la ecuación 4 representa el beneficio de los beneficios y sinergias de los elementos restando el coste máximo posible que estos puedan tener.

La restricción de presupuesto se define con las ecuaciones 5 y 6, en principio X debe ser robusto a las variaciones en los costes por lo que se usa un acercamiento que asume el peor de los casos, es decir el cálculo del coste será siempre maximizado según las posibles variaciones. En este contexto para linealizar el problema Baldo et al. (2023) define las variables de decisión auxiliares 3, que transforman el coste máximo a un problema de minimización.

Para fijar la variable auxiliar $\mathcal{Z}_{\mathcal{A}}$ se especifican las restricciones 7 y 8, que favorece las sinergias con beneficio positivo y desfavorece las con beneficio negativo.

2.3. Revisión de literatura

El trabajo de Baldo et al. (2023) ha introducido por primera vez una metodología para resolverlo, proponiendo un algoritmo genético y un algoritmo basado en machine learning. Este último utiliza un clasificador random forest que predice la probabilidad de cada elemento de estar presente en la solución óptima. Esta predicción se usa para fijar variables de decisión, reduciendo así el tiempo de ejecución de un solver exacto.

Li et al. (2021) ha estudiado el uso de redes neuronales para obtener predicciones de KPs con funciones objetivo no lineales, obteniendo buenos resultados con una estructura basada en teoría de juegos, junto al uso de redes neuronales adversarias.

Rezoug et al. (2022) aborda el KP usando distintas técnicas para evaluar las características de los elementos, entre ellos redes neuronales, regresión de procesos gaussianos, random forest y support vector regression. Así, resuelve el problema original, usando solo un subconjunto de los elementos. Luego, mediante el descenso del gradiente y el uso de características de los elementos, decide cuáles de los elementos excluidos agregar a la solución inicial obtenida. En sus resultados muestra que el modelo machine learning entrega soluciones similares a los otros clasificadores, con menores tiempos de ejecución.

Afshar et al. (2020) propuso un algoritmo para generar soluciones para el KP usando un modelo de Deep Reinforcement Learning que selecciona los elementos de forma greedy. El algoritmo propuesto construye las soluciones en base a las decisiones del modelo y genera soluciones con una razón de similitud con el óptimo del 99.9 %. Además, usa una arquitectura de A2C con un paso de cuantización de características, que reduce la complejidad espacial del problema en la red, resultando en modelos que usan menos memoria y se ejecutan más rápido.

Si bien estos trabajos no se relacionan directamente con la variante del problema propuesto, sí evidencian que las técnicas de Machine learning pueden usarse de forma efectiva para caracterizar y construir soluciones para el PRKP.

Capítulo 3

Metodología

En este capitulo se expondrá el algoritmo propuesto para resolver el PKRP, así como las definiciones del modelo machine learning que actúa como clasificador, las características de los ítems que se usarán y el proceso de reducción de instancias.

3.1. Algoritmo propuesto

Se propone un algoritmo iterativo con el objetivo de reducir la complejidad del problema de forma gradual, a costa de un pequeño nivel de confianza sobre la solución final obtenida. El algoritmo usa una red neuronal como clasificador. Que con base en ciertas características de cada elemento, decide la probabilidad de estar o no en la solución final. Estas predicciones se usan para asumir ceros en la solución y reducir la instancia original a una más pequeña mediante una transformación. Este proceso puede ejecutarse un número arbitrario de veces, mientras se mantenga la precisión del modelo, al predecir instancias con cada vez menos elementos.

Algoritmo 1 Algoritmo general

1: τ ▷ Umbral de confianza del clasificador
2: I ▷ Conjunto de elementos inicial
3: S ▷ Conjunto de sinergias
4: **loop**
5: $\mu \leftarrow \frac{|I|}{\log_{10}|I|}$ ▷ Máximo número de elementos a bloquear por paso
6: $F \leftarrow$ Características ▷ Cálculo de características de los elementos
7: $\tilde{x} \leftarrow$ Clasificador(F) ▷ Generación de predicciones continuas
8: $Z \subset I \leftarrow \mathrm{Z}(\tilde{x}, \tau, \mu)$ ▷ Selección de elementos para eliminar
9: **if** $|Z| = 0$ **or** $\min(\tilde{x}) > \tau$ **then**
10: **break**
11: **end if**
12: $I \leftarrow I - Z$ ▷ Reducción de la instancia
13: $S \leftarrow S - \{A \in S, \exists i \in A : i \in Z\}$ ▷ Actualización de sinergias
14: **end loop**
15: $X \leftarrow$ SolverExacto(I) ▷ Resolución exacta del conjunto reducido

El algoritmo 1 inicia con la definición del umbral de confianza del clasificador τ en la línea 1. Se establece el conjunto inicial de elementos I en la línea 2, y el conjunto de sinergias S en la línea 3.

El bucle principal comienza calculando las características de los elementos en la línea 6. Posteriormente, el clasificador genera predicciones $\tilde{x}$ en la línea 7 que se encuentran en el rango $[0, 1]$. A continuación, se selecciona un conjunto de elementos Z para eliminar en la línea 8 en función de las predicciones y el umbral τ.

El umbral τ se establece por el usuario para definir la precisión y velocidad de ejecución del algoritmo y pertenece al rango de valores $[0, 0.5]$.

El tamaño maximo de Z se define por μ en función del numero de elementos que existe en la instancia reducida, como se muestra en la linea 5.

El bucle continúa hasta que el conjunto Z está vacío o el valor mínimo de las predicciones $\tilde{x}$ supera el umbral τ, como se verifica en la línea 9. Durante cada iteración, se eliminan los elementos de Z de I y se descartan todas las sinergias que incluyan algún elemento de Z.

Finalmente, el algoritmo resuelve exactamente el conjunto reducido I utilizando el *solver* exacto en la línea 15, y el resultado se almacena en X.

3.2. Clasificador

El clasificador utilizado es una red neuronal de cinco capas. Para la entrada usa features calculadas para un elemento de la instancia, así, se genera una predicción sobre si el elemento tendrá un valor de cero o uno en la solución final.

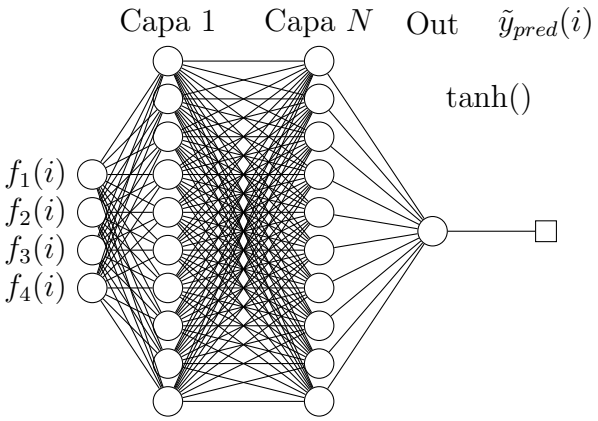


Figura 1: Estructura general de la red

$$\tilde{x}(i) = \frac{1}{2}\left(\tilde{y}(i) + 1\right) \tag{11}$$

Como se muestra en la Figura la entrada de la red es un vector que contiene todas las features del elemento i:

$$f(i) = \left(f_1(i), f_2(i), \ldots f_\phi(i)\right)$$

donde ϕ es del número de features.

Através de la función de activación tangente hiperbólica, la red genera un una salida $\tilde{y}$ que se encuentra en el rango [-1,1], la que es posible reconstruir usando la transformación de la ecuación 11.

En base a esto es útil definir la matriz $F_{\phi \times N}$ como en la ecuación 12:

$$F_{\phi \times N}(I) = \begin{pmatrix} f(1) \\ f(2) \\ \vdots \\ f(N-1) \\ f(N) \end{pmatrix} \tag{12}$$

Así, la red es una función definida como:

Predicción para ítem:

$$Net(f) = \tilde{x_i} : f \in [0,1]^{\phi} \rightarrow \tilde{x}_i \in [0,1]$$

Predicción para todos los ítems de una instancia:

$$Net(F) = \tilde{x} : F \in [0,1]^{\phi \times N} \rightarrow x \in [0,1]^{N}$$

Las features que se utilizan, codifican los parámetros P_i, LC_i,UC_i, Γ, además de, $ContSol_i$,$Noise$. Las definiciones se encuentran en el anexo B.1.

Entre estas:

- $ContSol_i$ se refiere a la solución de la relajación continua del PRKP en que la variable de decisión x_i es continua.
- Noise es un generador de números aleatorios, que se usa para definir una entrada vacía en la red neuronal, esto da lugar para reemplazar esta feature por otra asociada al estado del algoritmo general.

3.2.1. Entrenamiento

Se realizan dos etapas de entrenamiento sobre un conjunto de instancias. La primera etapa consiste en entrenar la red para predecir la solución óptima de cada instancia, y así crear un clasificador general. En la segunda etapa, la red se afina con instancias generadas desde el algoritmo iterativo de reducción. Además, las soluciones con las que se entrenará la red son generadas por un solver exacto que calcula óptimo para cada instancia.

Algoritmo 2 Primer entrenamiento

1: TrainingSet ▷ Conjunto de instancias de entrenamiento
2: Net ▷ Red neuronal definida anteriormente
3: Optimizer ▷ Optimizador para la red
4: Solver ▷ Solver exacto para el problema
5: Loss ▷ Funcion de perdida
6: **for** Instance ∈ TrainingSet **do**
7: $I \leftarrow Instance_I$
8: $x \leftarrow Solver(I)$ ▷ Vector booleano x con la solución óptima de la instancia
9: **for** $i \in I$ **do**
10: $\tilde{x_i} \leftarrow Net(f(i))$ ▷ Predicción de la red de x_i desde las features
11: $loss \leftarrow Loss(x_i, \tilde{x_i})$
12: $Net \leftarrow Optimizer(Net, loss)$ ▷ Un paso de optimización de la red
13: **end for**
14: **end for**
15: Net ▷ Red entrenada

El algoritmo 2 de entrenamiento comienza designando un conjunto de instancias en la linea 1 cada una de las estas define el conjunto de parámetros del modelo de programación lineal. Se define la red que se usará en la linea 2, inicializándose con pesos aleatorios.En la linea 5 se declara la función de perdida, que corresponde a la diferencia lineal entre el valor predicho y el valor real.

En la linea 3 se declara el optimizador encargado de reajustar los pesos de la red en cada paso de entrenamiento, se usa el algoritmo Adam introducido por Kingma and Ba (2017), que funciona como alternativa al descenso de gradiente estocástico. Finalmente se declara el solver exacto a usar, Gurobi 10.0.3.

A continuación, se inicia un bucle sobre cada instancia del conjunto de entrenamiento en la línea 6. Para cada instancia, se obtiene el conjunto de elementos así como sus otros parámetros, definiendo I de la instancia, en la línea 7. Se obtiene la solución óptima x para la instancia usando el solver en la línea 8.

Posteriormente, se inicia un bucle interno sobre los elementos de la instancia en la línea 9. Dentro de este bucle, la red neuronal realiza predicciones $\tilde{x_i}$ sobre las características de la instancia utilizando la función $f(i)$ en la línea 10. Se calcula la perdida entre la predicción y la solución óptima en la línea 11, y la red se actualiza mediante un paso de optimización utilizando Adam en la línea 12.

Este proceso de predicción, cálculo de pérdida y optimización se repite para todos los índices en el bucle interno. Una vez completado, se regresa al bucle externo para la siguiente

instancia del conjunto de entrenamiento.

Después de que se han procesado todas las instancias del conjunto de entrenamiento, la red neuronal Net se considera entrenada y se devuelve como resultado del algoritmo en la línea 15.

Algoritmo 3 Segundo entrenamiento

```
1: τ                                              ▷ Umbral de confianza del clasificador
2: TrainingSet                                    ▷ Conjunto de instancias de entrenamiento
3: Net                                            ▷ Red neuronal pre-entrenada
4: Optimizer                                      ▷ Optmizador para la red
5: Loss                                           ▷ Función de perdida
6: for Instance ∈ TrainingSet do
7:     I ← Instance_I
8:     loop                                       ▷ Bucle de reducción
9:         μ ← |I| / log10 |I|                    ▷ Maximo numero de elementos a bloquear por ciclo
10:        x̃ ← Net(F(I))                          ▷ Vector de predicción de la red
11:        Z ⊂ I ← Z(x̃, τ, μ)
12:        if |Z| = 0 or mín(x̃) > τ then
13:            break
14:        end if
15:        I ← I − Z                              ▷ Reducción de la instancia
16:        S ← S − {A ∈ S, ∃i ∈ A : i ∈ Z}        ▷ Actualización de sinergias
17:        x ← Solver(I)                          ▷ Solución óptima de la instancia reducida
18:        x̃ ← Net(F(I))                          ▷ Predicción para la instancia reducida
19:        for i ∈ I do
20:            loss ← Loss(x_i, x̃_i)              ▷ Cálculo de la perdida
21:            Net ← Optimizer(Net, loss)         ▷ Actualización de los pesos de la red
22:        end for
23:    end loop
24: end for
25: Net                                           ▷ Red entrenada
```

El segundo proceso de entrenamiento descrito en el algoritmo 3 sigue la estrategia del algoritmo general 1. Inicialmente se declaran los parámetros τ y μ, asi como el conjunto de instancias de entrenamiento, la red neuronal pre-entrenada, el optimizador y la función de perdida, en las líneas 1 a 5. Se ejecuta el bucle de reducción por cada instancia, usando red la predicción en la línea 10. Al igual que en el algorimo general 1 , se usa esta predicción, para generar una instancia reducida, este proceso ocurre en las líneas 11 a 16. Luego se calcula la solución óptima para la instancia reducida, en la línea 17, asi como la predicción de la red en la línea 18. Finalmente, se realiza un paso de optimización por cada elemento en la instancia reducida donde se calcula la perdida entre la predicción y el valor óptimo

en la linea 20 y se ajustan los pesos de la red en la línea 21. Después de ejecutar este proceso por todas las instancias del conjunto de entrenamiento, y actualizar los pesos usando todos los pasos de reducción de las instancias, se obtiene la red entrenada final.

Un cambio que mejora la precisión de la red resolver instancias reducidas, es reemplazar la feature 19 que correspondía un valor aleatorio, por una métrica que representa cuanto se ha reducido la instancia desde su tamaño original. Por esto en las ocaciones que se calcula $F(I)$ para el segundo entrenamiento, f_6 es reemplazada por la feature de la ecuación 13, donde la instancia actual se refiere $|I|$ de cada ciclo del bucle de reducción.

$$f_7(i) = \frac{\text{N instancia Original}}{\text{N instancia actual}} \tag{13}$$

3.3. Construcción de Z y reducción de instancias

EL proceso de reducción descrito en las lineas 12 y 13 del algoritmo general 1 requiere definir el proceso de creación de Z con base en los parámetros.

Algoritmo 4 $Z(\tilde{x}, \tau)$

1: $B \leftarrow \{i \in I : \tilde{x}_i \leq \tau\}$
2: $B \leftarrow Sort(A)$ ▷ Ordenar de forma ascendente en base a los valores de x_i
3: $Z \leftarrow \{\}$
4: $c \leftarrow 0$
5: **for all** $b \in \{1, 2, 3..., |B|\}$ **do** ▷ Agregar los μ items con menor x_i
6: $Z \leftarrow B_b$
7: $c \leftarrow c + 1$
8: **if** $c = \mu$ **then**
9: **break**
10: **end if**
11: **end for**
12: Z

El algoritmo 4 se encarga de construir Z de forma que contenga los elementos ordenados con predicciones mas cercanas a cero que estén bajo el umbral τ, con un tamaño máximo de μ. Comienza creando un conjunto B que posee todos los elementos cuya predicción $\tilde{x}_i$ es menor que τ en la linea 1. En la línea B este conjunto es ordenado de manera ascendente, de forma que $\tilde{x}_{B_b} < \tilde{x}_{B_{b+1}} \forall b \in [1, |B|]$. Así, en un bucle que itera sobre los indices de B, los elementos son agregados a Z en la linea 6 de forma estrictamente ordenada. Este blucle termina si B se queda sin elementos que agregar a Z, o si la cardinalidad de Z supera el

número máximo elementos definido por μ. De esto se encarga el contador c, definido en la linea 4 el que cuenta cada elemento agregado a Z en la linea 7 y detiene el blucle en el control de flujo de la linea 8.

Al aplicar el proceso de reducción a una instancia de forma repetida, es posible que sucedan los siguientes casos.

- En cada paso la proporción de unos vs ceros en la solución óptima sera cada vez mayor, hasta el punto en que será irreducible.
- Γ no cambia al eliminar ceros. Es posible que se llegue a que $\Gamma > |I|$, dejando la posibilidad de reformular el problema como PKP.

Capítulo 4

Resultados Experimentales

En este capitulo se presentará la comparación cuantitativa de resultados, se re-contextualizarán los métodos de Baldo et al. (2023) con base en las diferencias de hardware y se mostrará la evaluación del algoritmo propuesto.

4.1. Instancias

Las instancias reales que se resuelven provienen de un generador de instancias implementado por Baldo et al. (2023) cuyas sinergias tienen en su mayoría beneficios de cero. Las instancias con I mayor a 1000, tienen una generación exponencial de sinergias. Para $300 \leq I \leq 1000$, se usa una generación cuadrática de sinergias. Para instancias con menos de 300 elementos, se generan sinergias de forma lineal.

Para comparar el rendimiento del algoritmo propuesto con los solvers existentes propuestos por Baldo et al. (2023), se usó el mismo conjunto de instancias propuesto, además de otras de mayor tamaño con el objetivo de comparar rendimiento para problemas de un orden de magnitud más grande.

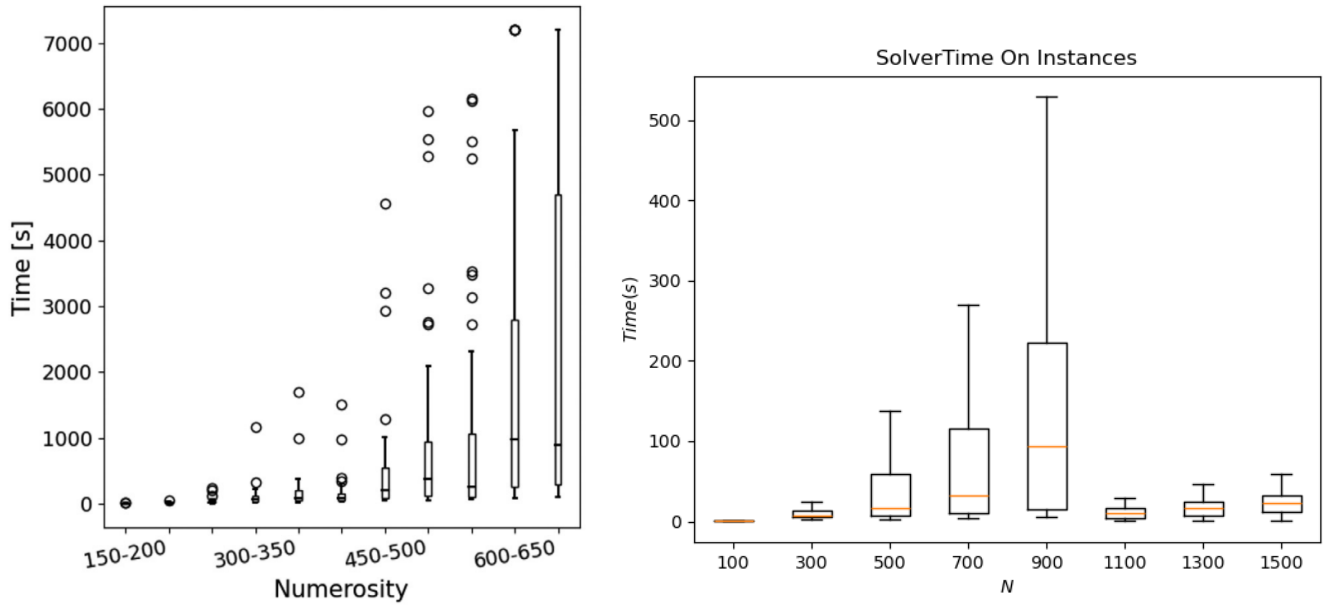
4.2. Comparaciones

Se calcularon los resultados para todas las instancias de Baldo et al. (2023), usando los métodos de BaldoGA y BaldoML, además de el algoritmo propuesto.

Resultados generales

Comparación de solver exacto con trabajo anterior

En la figura 2 se muestran los tiempos para resolver instancias usando el solver exacto, en contraste con Baldo et al. (2023), que solo resolvió instancias de hasta 650 elementos con tiempos hasta 7000s, en este caso los tiempos están acotados a cerca de 550 segundos.



(a) Ejecución Gurobi Baldo et al. (2023) **(b)** Ejecución Gurobi actual

Figura 2: Tiempos de ejecución solver exacto

Rendimiento normalizado

Se presenta en las figuras 3 y 4 la comparación general de rendimiento de los métodos. BaldoGA tiene un $g\bar{a}p = 3.29\,\%$ y BaldoML tiene un gap promedio del $2.1\,\%$. Estos resultados son congruentes con el estudio de Baldo et al. (2023), mostrando comparabilidad con los resultados.

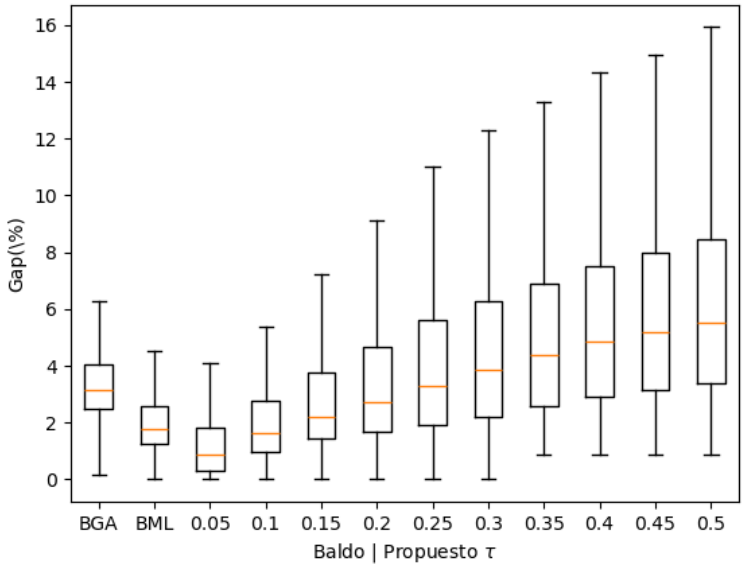


Figura 3: Gap % de todos los métodos presentados

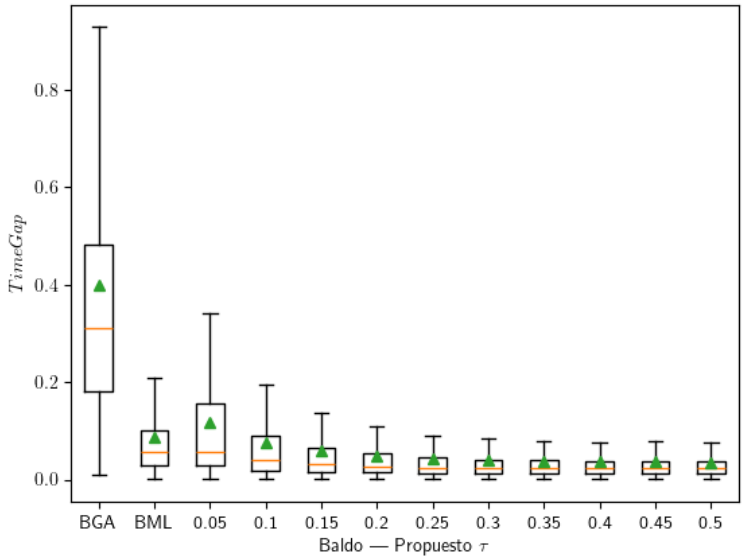


Figura 4: TimeGap(%) de los métodos usados

Método			N							
			100	300	500	700	900	1100	1300	1500
BaldoGA	Gap(%)	μ	1.74	3.07	3.04	3.42	3.74	3.4	4.1	3.9
		σ	1.15	0.905	0.866	0.844	1.09	1.07	6.89	1.18
	Time(s)	μ	0.44	3.3	4.5	7.18	9.86	5.61	7.22	8.89
		σ	0.207	1.32	1.75	2.93	4.08	2.19	2.44	3.21
BaldoML	Gap(%)	μ	3.16	3.0	2.2	2.0	1.85	1.41	1.37	1.39
		σ	1.87	1.19	0.907	0.72	0.66	0.56	0.6	0.49
	Time(s)	μ	0.099	0.36	0.48	0.79	1.19	1.5	2.12	2.85
		σ	0.0147	0.0577	0.119	0.277	0.47	0.881	1.33	1.67
$\tau = 0.05$	Gap(%)	μ	2.45	1.79	1.36	1.40	1.41	1.40	1.23	1.02
		σ	2.50	1.70	1.35	1.36	1.38	1.36	1.26	1.15
	Time(s)	μ	0.20	0.88	1.46	2.61	3.80	3.41	4.67	6.28
		σ	0.09	0.37	1.05	2.06	2.97	2.73	3.47	4.49

Tabla 3: Comparación general por número de elementos

La tabla 3 es un extracto de los resultados en **??**, que muestra BaldoML, BaldoGA y El algoritmo propuesto con un $\tau = 0.05$. Es posible notar que el propuesto encuentra soluciones con un menor gap promedio que BaldoML, esto a costa de tener desviaciones estándar más altas y tiempos de ejecucion comparativamente mas altos.

En la figura ?? se ilustra la principal diferencia entre baldoML y el propuesto. Los métodos tienen un gap medio de 2.05 % y 1.51 % respectivamente, esta diferencia de 0.54 % de ambas medias es significativa con un $p = 1.8 \cdot 10^{-26} < 0.05$ en una prueba T-student, rechazando la hipótesis de igualdad de medias.

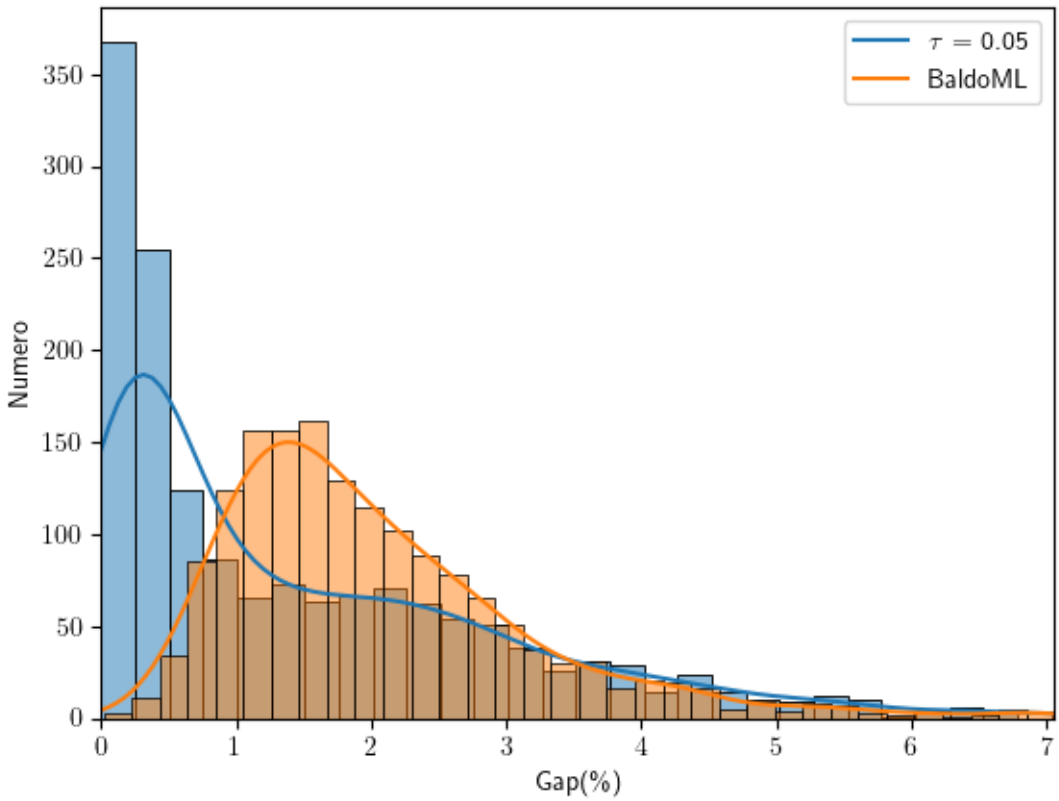


Figura 5: Contraste distribución gap BaldoML vs Propuesto

Respecto a los tiempos de ejecución, baldo resolvió todas las instancias en 1859.86 segundos, lo que corresponde a 1.1719s en promedio por instancia. Mientras que el método propuesto tardó 4611.1s en resolver el conjunto completo de instancias, con un promedio de 2.9 segundos. Debido a que las instancias tienen distintos tamaños y los tiempos de ejecución varían exponencialmente según este, se presenta en la figura ?? una distribución de los tiempos normalizados. Se aprecia visualmente que ambos tiempos se distribuyen de forma similar, con un desplazamiento positivo de la media del método propuesto. En promedio baldo se tarda 8.5 % del tiempo del solver óptimo en encontrar una solución, mientras que el propuesto 15.93 %.

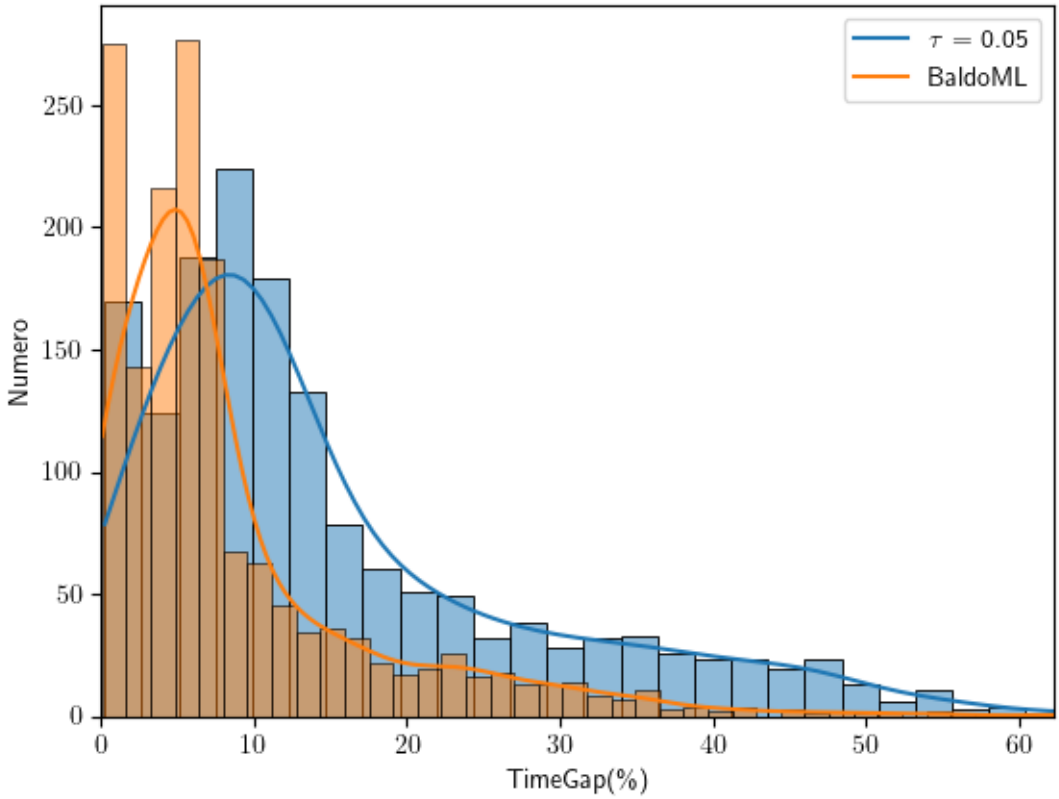


Figura 6: Contraste distribución tiempos de ejecución normalizados BaldoML vs Propuesto

Instancias grandes

Se resolvieron 50 instancias de $|I| = 20000$ con el objetivo de comparar las capacidades de escalamiento de los métodos BaldoML y el propuesto. Debido a la inviabilidad de utilizar el solver exacto para estos tamaños de instancias los valores de la función objetivo son comparados a razón de $\frac{\text{Propuesto}_{Of}}{\text{BaldoML}_{Of}}$ para obtener su gap relativo.

En todos los casos, baldoML encontró mejores soluciones para las instancias, pero como se ilustra en la figura 7 la diferencia entre esos es marginal. Las soluciones del método propuesto fueron en promedio un 4.47 % más pequeñas que las soluciones obtenidas por baldoML, variando entre un 6.2 % y 2.85 % en el peor y mejor de los casos respectivamente.

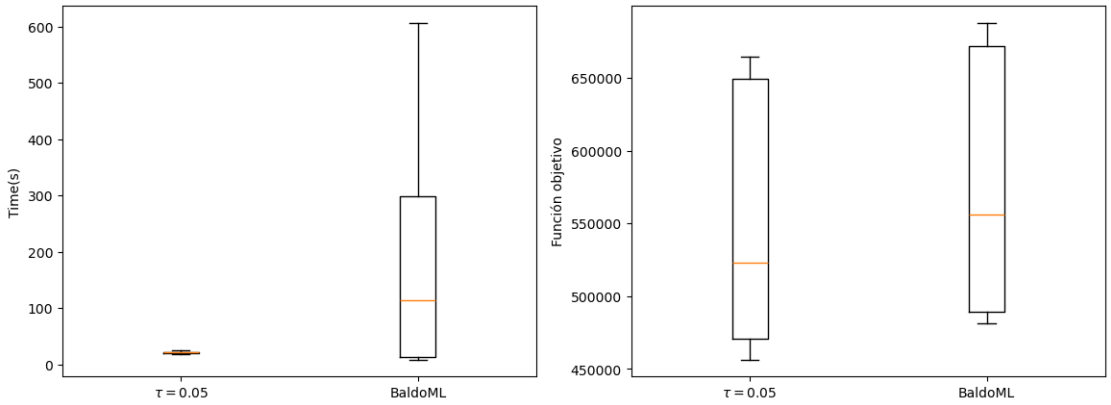


Figura 7: Comparación BaldoML y Propuesto en instancias de 20000 elementos

En cuanto a los tiempos de ejecución, la diferencia ha tenido mayor magnitud. Se estableció un límite de ejecución de 600 segundos por instancia. Mientras que baldoML tardo 2 horas y 26 minutos en resolver todo, el método propuesto usó poco menos de 18 minutos. Además, el tiempo de ejecución del propuesto, parece mantenerse relativamente constante en estos tamaños, mientras los tiempos de baldoML se distribuyen de forma más amplia, con una desviación estándar muy alta.

	BaldoML	Propuesto
Tiempo total	8809s	1064.02s
Tiempo promedio	176.18s	21.28s
Desviación estandar	164.44s	1.60s

Tabla 4: Estadísticas tiempos de ejecución en instancias grandes

Capítulo 5

Conclusiones

En esta memoria de título se describe un algoritmo basado en machine learning para abordar el problema polinomial robusto de la mochila, usando un clasificador que opera de forma iterativa sobre una instancia para reducir su complejidad. Se ha definido la estructura de la red neuronal, el algoritmo general de operación, los 2 métodos de entrenamiento y el método de reducción de la instancia.

Respecto a los resultados, el algoritmo muestra un rendimiento objetivamente superior al algoritmo genético propuesto por Baldo et al. (2023) en terminos de precisión y tiempos de ejecucion y si bien tiende a tardar mas que BaldoML, presenta mejores soluciones. Además, tiene mejores características de escalamiento en términos de rendimiento y tiempos de ejecución para instancias de tamaño no contemplado en el conjunto de entrenamiento. La metodología propuesta es entonces más adecuada para abordar instancias que son in-factibles para resolver con un solver exacto pese a tener altos tiempos de ejecución en instancias pequeñas.

En el futuro sería importante explorar el uso de técnicas más avanzadas de machine learning para abordar el problema, implementar sistemas de decisión automática para los parámetros de los solvers mostrados y además explorar el uso de características derivadas de estimadores basados en machine learnig, como la arquitectura A2C introducida por Mnih et al. (2016), con el fin de predecir características sin necesidad de derivarlas de la solución óptima.

Bibliografía

Afshar, R. R., Zhang, Y., Firat, M., and Kaymak, U. (2020). A State Aggregation Approach for Solving Knapsack Problem with Deep Reinforcement Learning. In *Proceedings of The 12th Asian Conference on Machine Learning*, pages 81–96. PMLR. ISSN: 2640-3498.

Baldo, A., Boffa, M., Cascioli, L., Fadda, E., Lanza, C., and Ravera, A. (2023). The polynomial robust knapsack problem. *European Journal of Operational Research*, 305(3):1424–1434.

Eilon, S. (1987). Application of the knapsack model for budgeting. *Omega*, 15(6):489–494.

Forrester, R. J. and Waddell, L. A. (2022). Strengthening a linear reformulation of the 0-1 cubic knapsack problem via variable reordering. *Journal of Combinatorial Optimization*, 44(1):498–517.

Gallo, G., Grigoriadis, M., and Tarjan, R. (1989). A Fast Parametric Maximum Flow Algorithm and Applications. *SIAM J. Computing.*, 18:30–55.

Gallo, G., Hammer, P. L., and Simeone, B. (1980). Quadratic knapsack problems. In Padberg, M. W., editor, *Combinatorial Optimization*, Mathematical Programming Studies, pages 132–149. Springer, Berlin, Heidelberg.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Kofler, C., Greistorfer, P., Wang, H., and Kochenberger, G. (2014). A Penalty Function Approach to Max 3-SAT Problems. *Working Paper Series, Social and Economic Sciences*. Number: 2014-04 Publisher: Faculty of Social and Economic Sciences, Karl-Franzens-University Graz.

Li, D., Liu, J., Lee, D., Seyedmazloom, A., Kaushik, G., Lee, K., and Park, N. (2021). A Novel Method to Solve Neural Knapsack Problems. In *Proceedings of the 38th International Conference on Machine Learning*, pages 6414–6424. PMLR. ISSN: 2640-3498.

Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. Wiley-Interscience series in discrete mathematics and optimization. J. Wiley & Sons, Chichester ; New York.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs] version: 2.

Mohammadi, S., Gleich, D. F., Kolda, T. G., and Grama, A. (2017). Triangular Alignment (TAME): A Tensor-Based Approach for Higher-Order Network Alignment. *IEEE/ACM transactions on computational biology and bioinformatics*, 14(6):1446–1458.

Monaci, M., Pferschy, U., and Serafini, P. (2013). Exact solution of the robust knapsack problem. *Computers & Operations Research*, 40(11):2625–2631.

Rezoug, A., Bader-el den, M., and Boughaci, D. (2022). Application of Supervised Machine Learning Methods on the Multidimensional Knapsack Problem. *Neural Processing Letters*, 54(2):871–890.

Rhys, J. M. W. (1970). A Selection Problem of Shared Fixed Costs and Network Flows. *Management Science*, 17(3):200–207. Publisher: INFORMS.

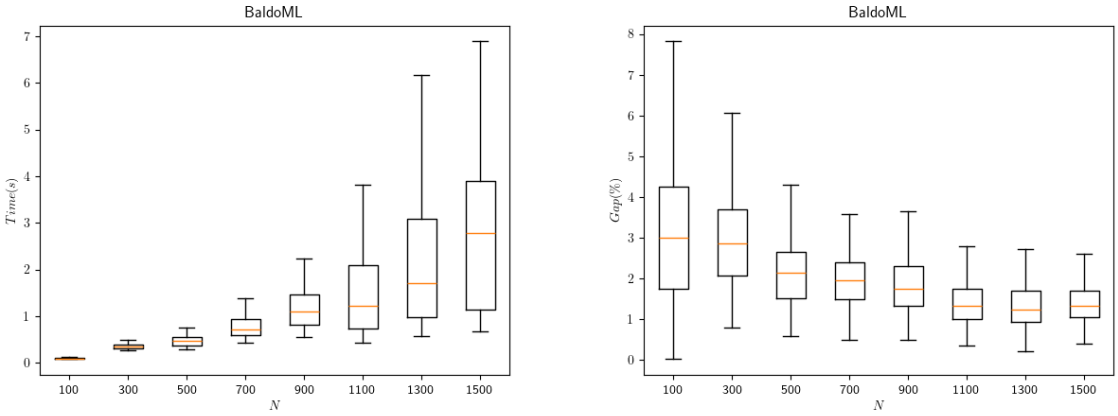
Witzgall, C. (1975). Mathematical methods of site selection for electronic message systems (EMS). Technical Report NBS IR 75-737, National Bureau of Standards, Gaithersburg, MD. Edition: 0.

Zhao, K. (2008). Treatments of Chlamydia Trachomatis and Neisseria Gonorrhoeae. *Mathematics Theses*.

Apéndice A

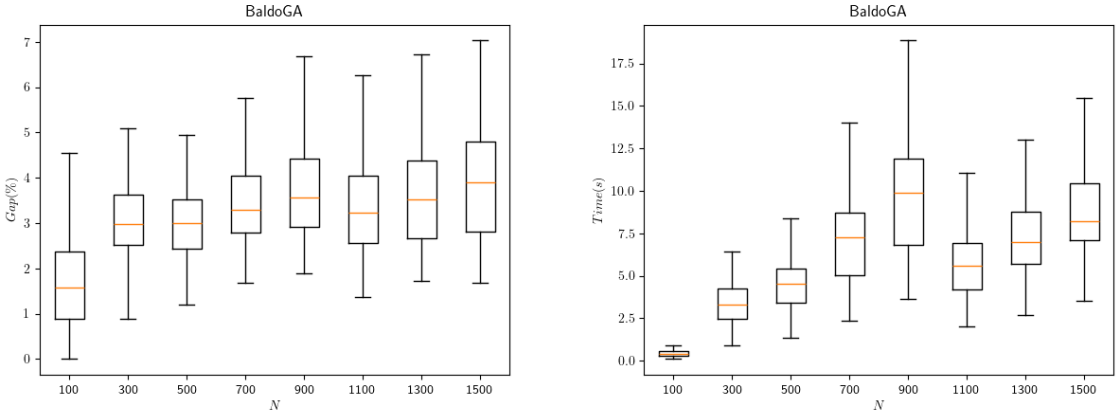
Material

A.1. Rendimiento de métodos en distintos tamaños de entrada



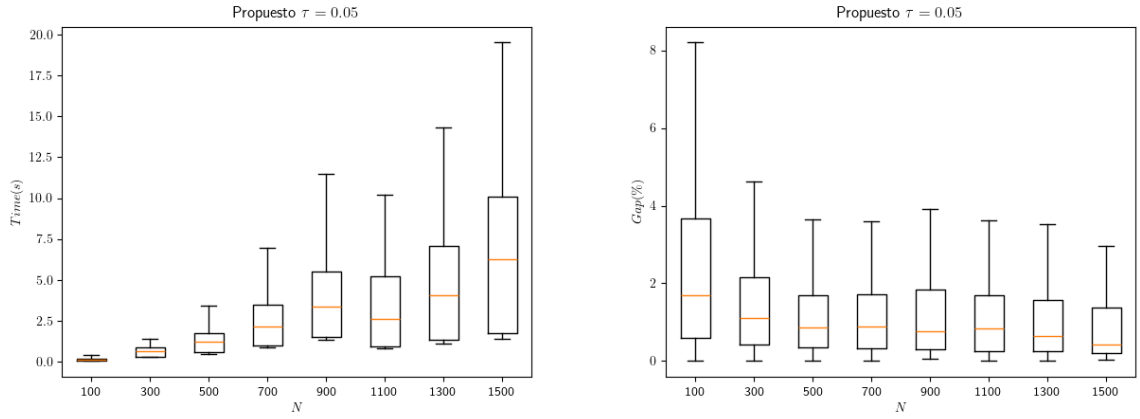
(a) Tiempo vs Numero de elementos **(b)** Gap vs Número de Elementos

Figura 8: Rendimiento BaldoML



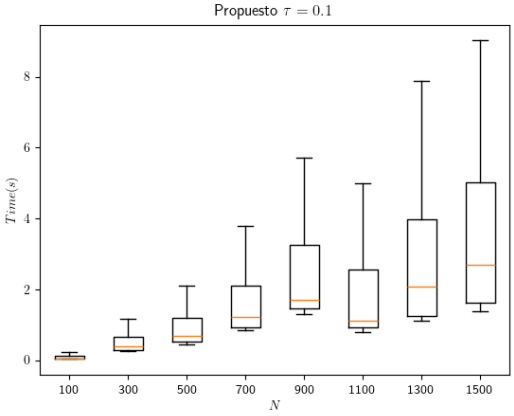
(a) Tiempo vs Numero de elementos **(b)** Gap vs Numero de Elementos

Figura 9: Rendimiento BaldoGA

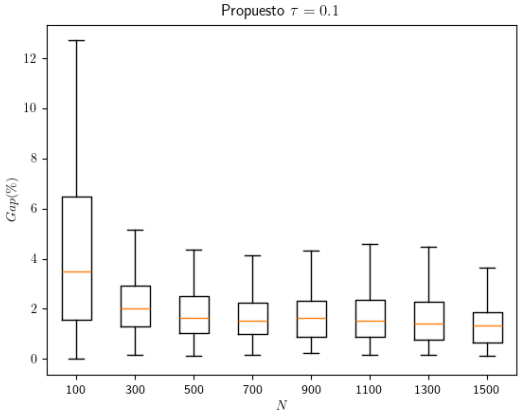


(a) Tiempo vs Numero de elementos **(b)** *Gap*(%) vs Numero de Elementos

Figura 10: Rendimiento Algorimo propuesto ($\tau = 0.05$)

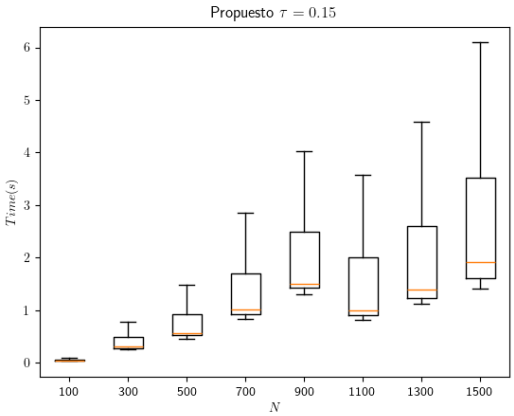


(a) Tiempo vs Numero de elementos

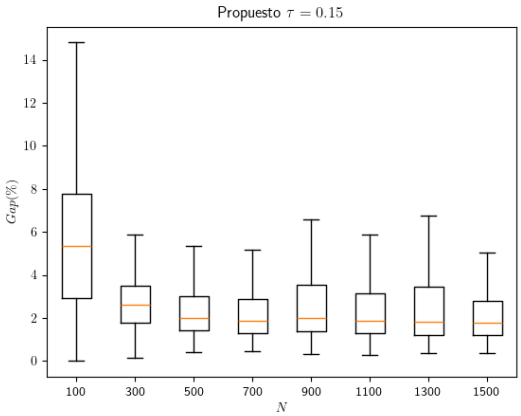


(b) *Gap*(%) vs Numero de Elementos

Figura 11: Rendimiento Algorimo propuesto ($\tau = 0.1$)

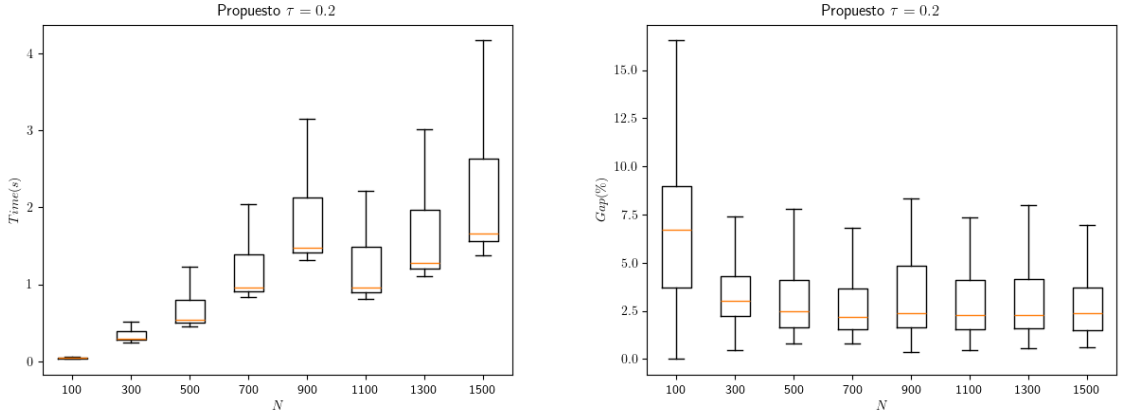


(a) Tiempo vs Numero de elementos



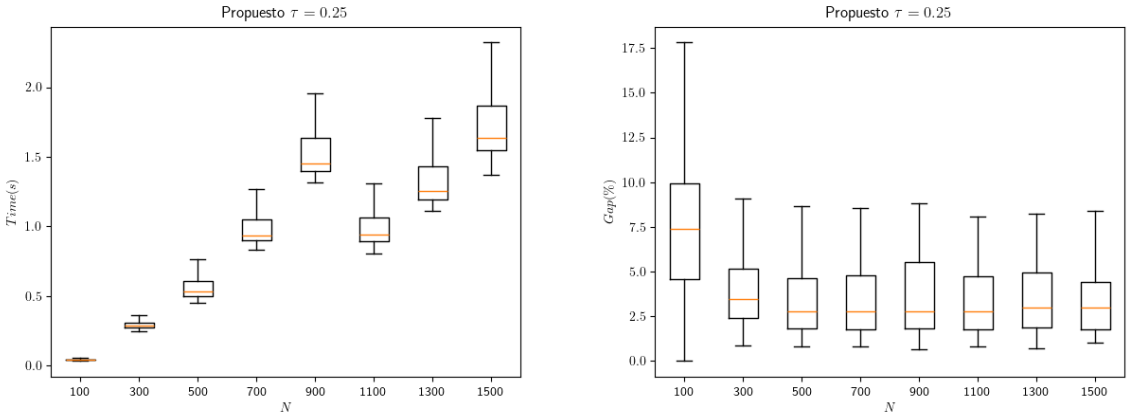
(b) *Gap*(%) vs Numero de Elementos

Figura 12: Rendimiento Algorimo propuesto ($\tau = 0.15$)



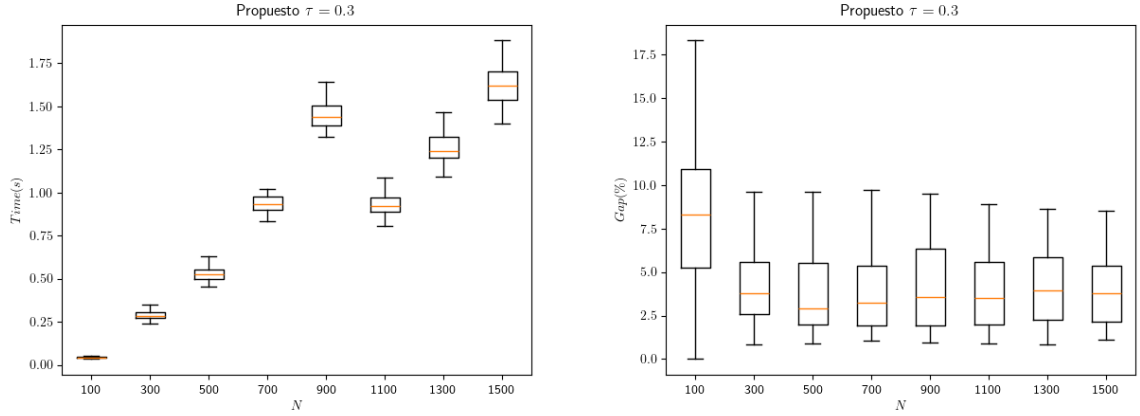
(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 13: Rendimiento Algorimo propuesto ($\tau = 0.2$)



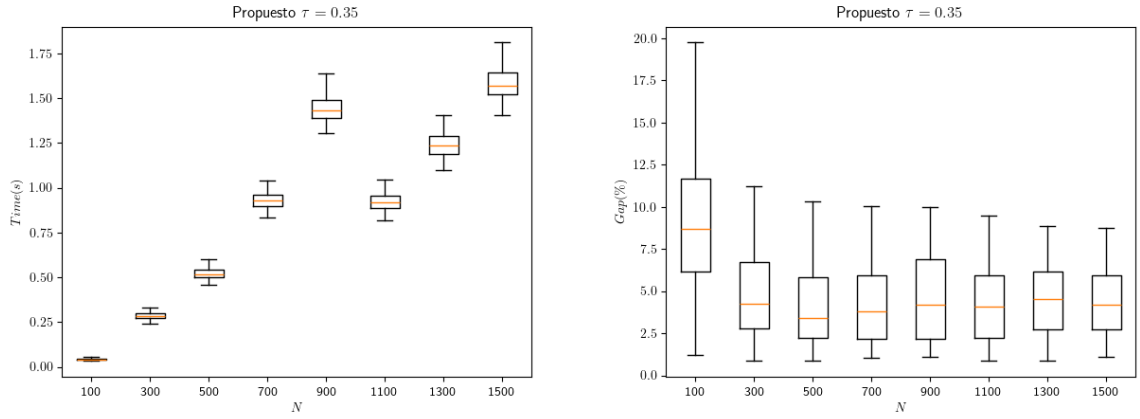
(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 14: Rendimiento Algorimo propuesto ($\tau = 0.25$)



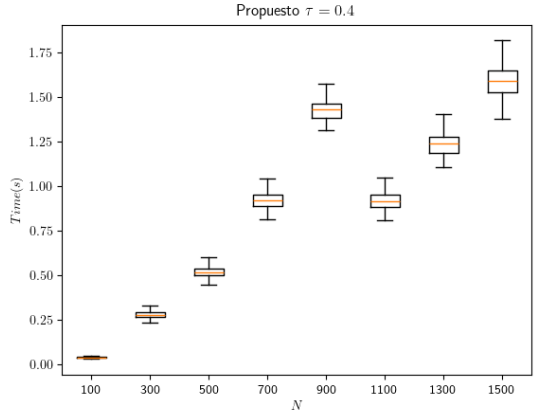
(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 15: Rendimiento Algorimo propuesto ($\tau = 0.3$)

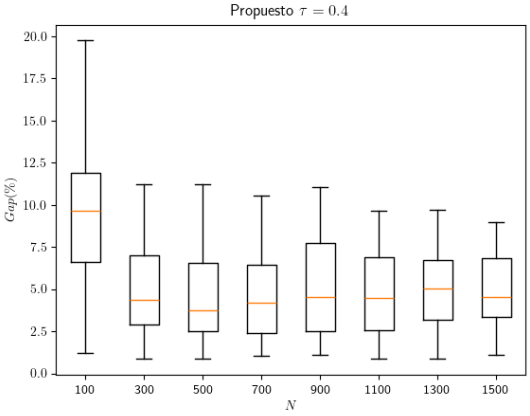


(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 16: Rendimiento Algorimo propuesto ($\tau = 0.35$)

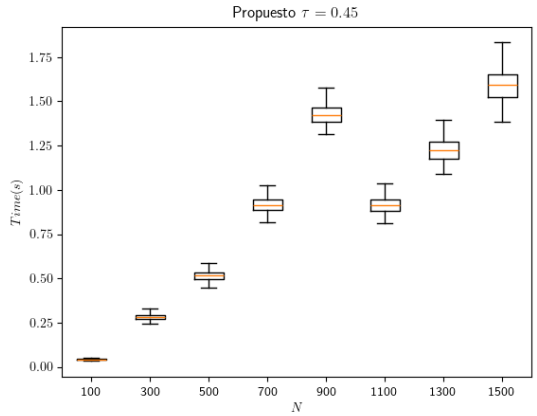


(a) Tiempo vs Numero de elementos

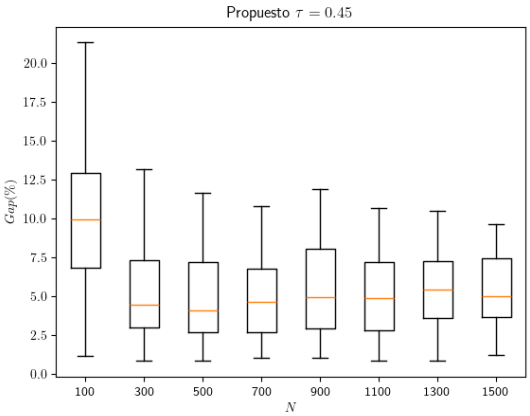


(b) *Gap*(%) vs Numero de Elementos

Figura 17: Rendimiento Algorimo propuesto ($\tau = 0.4$)

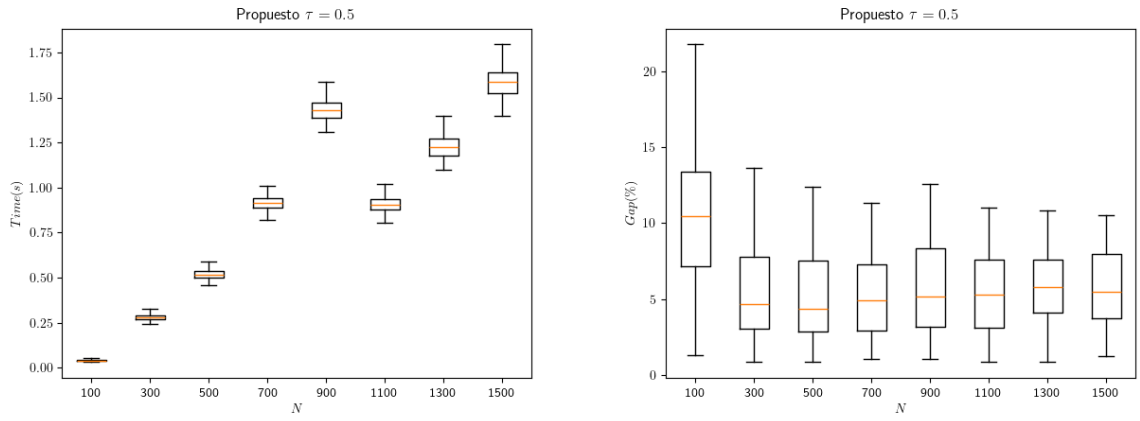


(a) Tiempo vs Numero de elementos



(b) *Gap*(%) vs Numero de Elementos

Figura 18: Rendimiento Algorimo propuesto ($\tau = 0.45$)



(a) Tiempo vs Numero de elementos **(b)** *Gap*(%) vs Numero de Elementos

Figura 19: Rendimiento Algorimo propuesto ($\tau = 0.5$)

Apéndice B

Ecuaciones

B.1. Features Clasificador

$$f_1(i) = \frac{P_i}{W} \tag{14}$$

$$f_2(i) = \frac{LC_i}{W} \tag{15}$$

$$f_3(i) = \frac{UC_i}{W} \tag{16}$$

$$f_4(i) = \frac{\Gamma}{N} \tag{17}$$

$$f_5(i) = \text{ContSol}_i \tag{18}$$

$$f_6(i) = \text{Random}(i) \tag{19}$$