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.

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 $P_i \in \mathbb{R} : i \in I$. Ademas, 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

En la formulación de Baldo et al. (2023) se describe la formulación del PRKP y realiza una linealización para transformar el modelo de programación no lineal a un problema de programación lineal (PL) compatible con solvers adecuados como Gurobi o Cplex. Para efectos de esta memoria, no es necesario usar la linealización del problema y se referirá a la primera formulación de baldo y se reservará su linealización para uso exclusivo como modelo de PL.

La formulación consiste en un conjunto elementos o alternativas que poseen un beneficio, un costo nominal y un costo máximo asociados, definidos como:

- 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.

Así, nuestra variable de decisión es el vector binario definido en 1

$$x_i = \begin{cases} 1 & \text{si el elemento } i \text{ es elegido} \\ \\ 0 & \text{si el elemento } i \text{ no es elegido} \end{cases} \tag{1}$$

Ahora bien, dada una solución x, algunos elementos elegidos pueden variar en sus costes, con valores entre LC_i y UP_i. El máximo número de elementos que puede variar su coste dada una solución se describe por el parámetro Γ.

Las soluciones robustas encontradas deben tener la propiedad, de que para cualquier combinación posible de costos obtenidos entre LC y UC, no debe superarse el presupuesto. Para esto se asume el peor de los casos para las variaciones, es decir, donde todos los costes que varían son elementos de la solución y además se eligen los Γ elementos con mayor variación entre coste nominal y máximo y se varían.

De esta forma se define la variación de costo de un elemento i como:

$$\Delta C_i = UC_i - LC_i$$

Y la restricción de presupuesto, como:

$$\left(\sum_{i=1}^{I} LC_i \cdot x_i \right) + \text{máx} \left(\sum_{i=1}^{I} \Delta C_i \cdot y_i \right) \leq W \tag{2}$$

Donde la variable auxiliar y_i describe si el elemento x_i varía o no, lo que por la formulación está sujeto a:

$$\sum_{i=1}^{I} y_i \leq \Gamma$$

2.2.0.1. Sinergias polinomiales

Las sinergias polinomiales corresponden a beneficios asociados a combinaciones específicas de elementos elegidos para una solución.

Cada sinergia $A \subseteq I$ tiene entonces asociado un beneficio PS_A (Profit Sinergy), y este beneficio se suma, solo si cada elemento de la sinergia está presente en la solución. Los beneficios totales obtenidos por las sinergias se muestran en 3 y se entiende de forma comprensiva que, si todos los elementos i en A tienen un x_i con un valor de 1 entonces se suma el beneficio, pero si uno de los elementos no está en la solución, es decir $x_i = 0$, entonces toda la productoria es cero y el beneficio agregado por la sinergia es cero.

$$\sum_{A \in S} \left(PS_A \cdot \prod_{i \in A} x_i \right) \tag{3}$$

2.2.0.2. Función objetivo

Dados los parámetros anteriores, se define la función objetivo **??**

$$\text{máx}\, f(x) = \sum_{i=1}^{I} p_i \cdot x_i + \sum_{A \in S} \left(PS_A \cdot \prod_{i \in A} x_i \right) - \left(\sum_{i=1}^{I} LC_i \cdot x_i + \text{máx} \sum_{i=1}^{I} \Delta C_i \cdot y_i \right) \tag{4}$$

Sujeto a la restricción de presupuesto

$$\sum_{i=1}^{I} LC_i \cdot x_i + \text{máx} \sum_{i=1}^{I} \Delta C_i \cdot y_i \leq W \tag{5}$$

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

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 su probabilidad deba 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 se puede realizar un número arbitrario de veces, mientras el clasificador mantenga una alta confianza, una vez se cumplan ciertas condiciones, se detiene el algoritmo y se resuelve la instancia final reducida para obtener la solución al problema.

Algoritmo 1 Algoritmo general

$ITo FixHistory$ ▷ Elementos que han sido fijados como cero
$OInstance$ ▷ Instancia original a resolver
$threshold$ ▷ Mínimo nivel de confianza tolerado
$Instance \leftarrow OInstance$
loop
  $F \leftarrow features$ ▷ Vector de features del elemento i
  $\tilde{x} \leftarrow Classifier(F)$ ▷ Predicción del clasificador
  $ITo Fix \leftarrow GetITo Fix(\tilde{x})$ ▷ Se obtienen los ítems para fijar como ceros
  $ItoFixHistory \leftarrow ITo Fix$
  $Instance \leftarrow Reduce(Instance, ITo Fix)$ ▷ Instancia reducida
  if $ItoFix == 0$ **or** mín$(\tilde{x}) <$ threshold **then**
    break ▷ Si el clasificador no encuentra más elementos que fijar o la predicción tiene mala confianza se detiene el algoritmo
  end if
  $y \leftarrow ExactSolver(Instance)$ ▷ Solución óptima para la instancia
end loop

Los puntos claves a definir del algoritmo general 1 son la reducción de las instancias, y el funcionamiento del clasificador, y en menor medida el cómo interpretar la predicción para fijar valores.

3.2. Reducción de instancias

Se puede reducir la complejidad de una instancia asumiendo la presencia de un elemento en la solución óptima, es decir , para un $i \in I$ se puede fijar $x_i = 0$ o $x_i = 1$, para reducir el espacio de búsqueda. Sin embargo, la formulación del problema, en específico 2, muestra que para calcular la restricción de presupuesto, y 4 la función objetivo, es necesario resolver un subproblema de optimización, que debe considerar todos los elementos de la instancia original, que estarán presentes en la solución final, debido a esto, para reducir instancias, solo es posible definir $x_i = 0$ para un i que se quiera declarar. Así, sea Z el conjunto de elementos que se quiere fijar como cero, es posible aplicar una transformación a los parámetros del problema, para generar una instancia de menor complejidad.

- $I' = I - Z$
- $S' = S - \{A \in S, \exists i \in A : i \in Z\}$

De forma comprensiva, ya no es necesario considerar los elementos de Z en la solución, pero más importante, todas las sinergias polinomiales que poseían elementos en Z, ya no

son alcanzables por ninguna solución, por lo que es seguro eliminarlas de S, notar también que Γ no es modificado.

Al aplicar esta transformación de forma iterativa es posible asumir ciertas propiedades:

- En cada paso, la solución óptima de cada iteración de la instancia estará compuesta por una proporción mayor de unos que de ceros. La proporción se le llamará, la densidad D de la instancia .
- Como Γ no cambia al eliminar ceros, es posible que se de la situacion en que $\Gamma > N$, en cuyo caso el problema pasa de ser un PRKP a PKP, puesto que para asegurar la robustes, solo es necesario considerar los costes altos de la solucion.
- Existe un punto en el que ya no quedan ceros que eliminar, el cual sucede cuando 2 se cumple para un x_i compuesto solo de unos, lo que es posible calcular a medida que se ejecuta la iteración.

Se define Z con base en la predicción del clasificador, si $\tilde{x} = Clasificador(I)$, la prediccion continua del clasificador puede ser interpretada como un vector de probabilidades, donde valores cercanos a cero son interpretados como que el clasificador predice para ese extremo y equivalente para el las predicciones cercanas a uno.

Se definen entonces dos parametros para generar Z, τ y μ:

- τ Corresponde al *threshold*, representa la confianza mínima necesaria, para decidir incluir un ítem en Z
- μ Es el número máximo de elementos que incluir en Z.

A base de estos parámetros, el algoritmo 2 construye Z

Algoritmo 2 $Z(\tilde{x}, \tau, \mu)$

$A \leftarrow \{i \in I : x_i \leq \tau\}$
$A \leftarrow Sort(A)$ ▷ Ordenar de forma ascendente en base a los valores de x_i
$Z \leftarrow \{\}$
$count \leftarrow 0$
for all $i \in A$ **do** ▷ Agregar los μ items con menor x_i
  $Z \leftarrow i$
  $count \leftarrow count + 1$
  if $count == \mu$ **then**
    break
  end if
end for
Z

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

El número máximo de elementos que bloquear en cada paso Z se ha definido como $Z = \text{máx}\{\frac{N}{\log_{10} N}, 100\}$ para cada instancia que se genera por reducción.

3.3. Clasificador

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

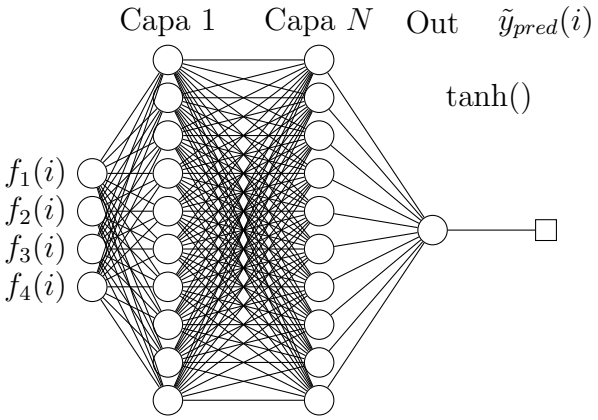


Figura 1: Estructura general de la red

Como se muestra en la figura 1, la red no intenta predecir el valor del x_i optimo directamente, sino que predice la transformación 6, que es más adecuada para la red, usando la función

de activación tangente hiperbólica para encasillar la salida al rango $[-1, 1]$. De esta forma, la salida de la red se reconstruye como $\tilde{x_i} = y^{-1}(\tilde{y}(i))$

$$y(i) = \begin{cases} 1, & \text{si } x_i = 1 \\ -1, & \text{si } x_i = 0 \end{cases} \tag{6}$$

La entrada de la red es un vector que contiene todas las features del elemento i:

$$f(i) = (f_1(i), f_2(i), \dots f_\phi(i)))$$

donde ϕ es del número de features

En base a esto es util definir la matriz de features $\times$ items como:

$$F(I) = \begin{pmatrix} f(1) \\ f(2) \\ \vdots \\ f(N-1) \\ f(N) \end{pmatrix} \tag{7}$$

Así, la red es una función definida como

Preduccion para item:

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

Prediccion para todos los items de una instancia:

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

Las features que se utilizaran, codifican P_i, LC_i,UC_i, $ContSol_i$, Γ, $Noise$, Las definiciones se encuentran en el anexo B.1.

Entre estas:

- $ContSol_i$ se refiere a la solución de la relajación continua del problema en que la variable de decisión x_i es continua.

- Noise es un generador de números aleatorios, que se usa para definir nodos vacíos extra para la red neuronal, esto da lugar para reemplazar esta feature por otra asociada al estado del algoritmo general.

3.3.1. Entrenamiento

Se realizarán 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 , los datos con los que se entrenará la red serán generados por un solver exacto que calcula x, óptimo para cada instancia.

Algoritmo 3 Primer entrenamiento

```
TrainingSet                                  ▷ Conjunto de instancias de entrenamiento
Net                                          ▷ Red neuronal
Optimizer                                    ▷ Optmizador para la red
Solver                                       ▷ Solver exacto para el problema
for Instance ∈ TrainingSet do
    x ← Solver(Instance)   ▷ Vector booleano x con la solución óptima de la instancia
    for i ∈ I do
        x̃_i ← Net(f(i))                      ▷ Prediccion de la red de x_i desde las features
        loss ← Loss(x_i, x̃_i)
        Net ← Optimizer(Net, loss)           ▷ Un paso de optimización de la red
    end for
end for
Net                                          ▷ Red entrenada
```

Notar que el algoritmo 3 realiza el proceso de optimización calculando la el gradiente y realizando el ajuste de la red por cada ítem de la instancia. Se usa la implementación de pytorch de Adam, algoritmo introducido por Kingma and Ba (2017) como alternativa para descenso de gradiente estocástico. Y se usa como función de perdida, simplemente la diferencia lineal entre los valores de entrenamiento, en este caso, entre la predicción y el valor óptimo de x_i.

Algoritmo 4 Segundo entrenamiento

TrainingSet ▷ Conjunto de instancias de entrenamiento
Net ▷ Red neuronal pre-entrenada
Optimizer ▷ Optmizador para la red
for OInstance ∈ TrainingSet **do**
  $ZHistory$
  $Instance \leftarrow OInstance$
  loop
    $\tilde{x} \leftarrow Net(F(I))$ ▷ Vector de predicción de la red
    $ZHistory \leftarrow Z(\tilde{x}, \tau, \mu)$
    $Instance \leftarrow Reduce(Instance, Z(\tilde{x}, \tau, \mu))$ ▷ Reducir la instancia
    if $\#Z == 0$ **or** mín$(\tilde{x}) \leq \tau$ **then**
      break
    end if
    $x \leftarrow Solver(Instance)$ ▷ Solución óptima de la instancia reducida
    $\tilde{x} \leftarrow Net(F(I))$ ▷ Predigo para la instancia reducida
    for $i \in I$ **do**
      $\tilde{x}_i \leftarrow Net(f(i))$
      $loss \leftarrow Loss(x_i, \tilde{x}_i)$
      $Net \leftarrow Optimizer(Net, loss)$
    end for
  end loop
end for
Net ▷ Red entrenada

El segundo entrenamiento sigue la estructura del algoritmo general 1, para así resolver instancias que son generadas por la reducción. La feature 13 que representa un numero aleatorio es reemplazada por $\frac{N_{\text{InstanciaActual}}}{N_{\text{InstanciaOriginal}}}$ para codificar un nivel de progesión en la red.

Capítulo 4

Resultados Experimentales

En este capitulo se presentará la comparación cuantitativa de resultados, se recontextualizarán los métodos de Baldo et al. (2023) con base en las diferencias de hardware, así como también

4.1. Instancias

Las instancias reales que se resolverán 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 mil, tienen una generación exponencial de sinergias, para $300 \leq I \leq 1000$, se usa una generación cuadrática de sinergias, mientras que para instancias con menos de 300 ítems, se agregan sinergias de forma lineal.

Para comparar el rendimiento del algoritmo propuesto con los solvers existentes propuestos por Baldo et al. (2023), se usará el mismo conjunto de instancias.

El hardware utilizado para evaual las instancias consta de un proecesador Intel i7-8550U con 4 nucleos 8 hilos y 32GB ram en Linux 6.6.1.

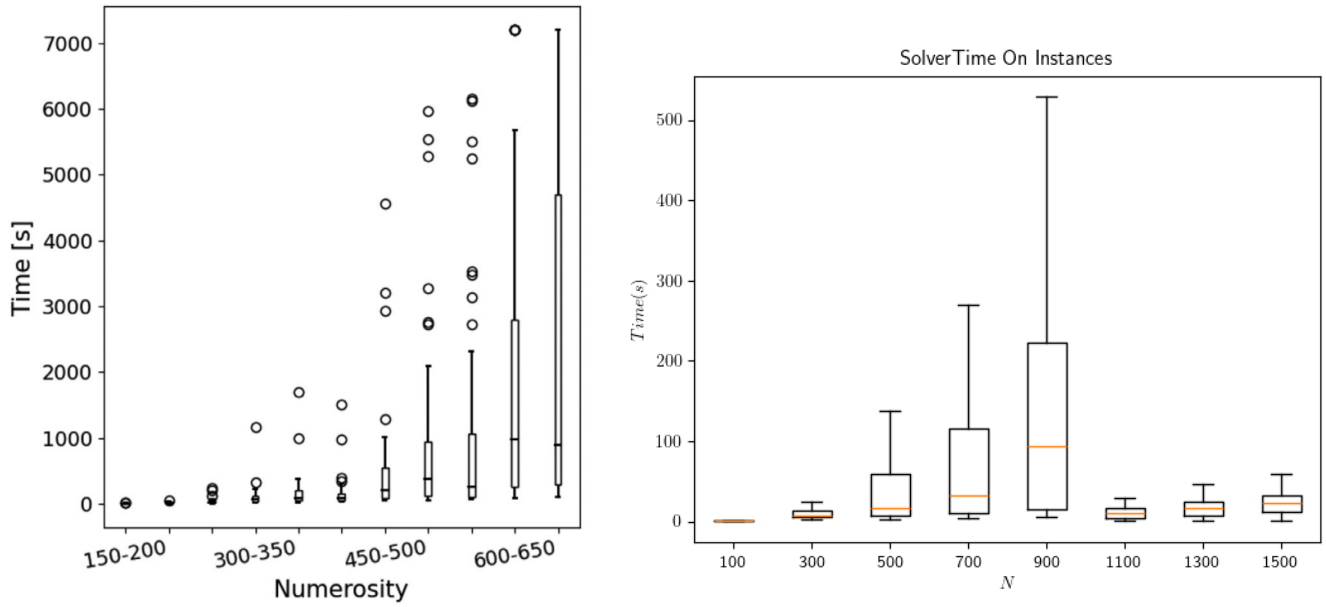
4.2. Comparaciones

Los resultados completos BaldoGA y BaldoML y la metodología aplicada por esta investigación. Como solver exacto se utilizó "Gurobi Optimizer version 11.0" y será la linea base para evaluar el rendimiento de los 3 métodos anteriores.

Resultados generales

Comparacion de solver exacto con trabajo anterior

En 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 en cerca de 550 segundos.



(a) Ejecucion Gurobi Baldo et al. (2023) **(b)** Ejecucion Gurobi

Figura 2: Tiempos de ejecucion solver exacto

Rendimiento normalizado

Se presenta en las figuras 3 y 4 la comparación general de rendimiento de los métodos. En primer lugar BaldoGA tienen el comportamiento esperado desde los resultados de Baldo et al. (2023), con un $g\bar{a}p$ de alrededor de un . por otro lado BaldoML tiene un gap promedio inferior al 1 % esto es debido a diferencias en la inicializacion del modelo randomforest al momento del entrenamiento, pero en mayor medida, por la diferencia de hardware, existe un subconjunto de instancias en las que el tiempo de ejecución del modelo, sumado al solver, superan el tiempo de ejecución del solver exacto, sobretodo al momento de calcular las features para la ejecucion, sobre las sinergias polinomiales.

Respecto a los tiempos de ejecucion, se puede notar que el algoritmo propuesto con un $\tau = 0.1$ genera resultados similares a BaldoGA mantiendo un 90 % de los tiempos de ejecución, menores a la media de los de BaldoGA. Ademas, BaldoML, a pesar de obtener

resultados con un gap muy cercano a cero, sus tiempos de ejecución tienden a ser elevados y en la figura 4 se muestra visualmente que cerca de un 20 % de los tiempos de ejecución es superior al tiempo del solver exacto.

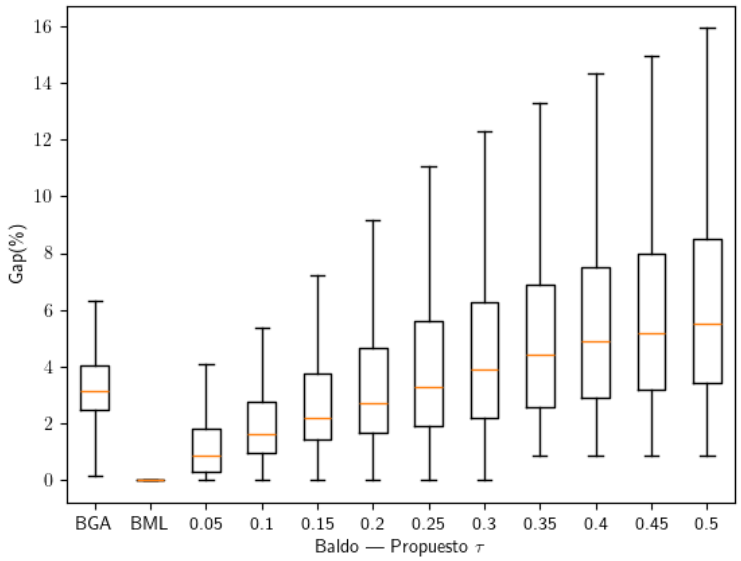


Figura 3: Gap % de todos los metodos presentados

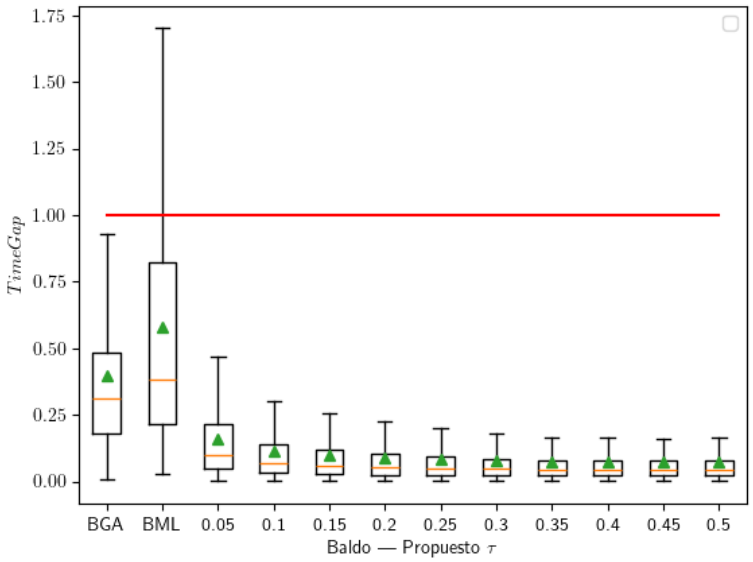


Figura 4: TimeGap de todos los metodos presentados

El cuadro 3 es un extracto de la tabla completa de resultados ??, que muestra BaldoML,

BaldoGA y El algoritmo propuesto con un $\tau = 0.25$. Es posible notar, que el algoritmo es menos preciso al encontrar soluciones de instancias relativamente pequeñas, pero se estabiliza con un gap promedio de alrededor de un 3 % para instancias de mayor tamaño.

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	0.15	0.0723	0.0751	0.105	0.037	1.03	2.01	2.02
		σ	0.339	0.151	0.202	0.218	0.0736	10.0	14.0	14.0
	Time(s)	μ	0.475	3.74	7.9	12.9	21.1	18.6	30.8	38.2
		σ	0.195	3.25	5.86	11.3	15.9	60.6	83.0	82.2
$\tau = 0.25$	Gap(%)	μ	7.47	3.94	3.42	3.37	3.7	3.43	3.52	3.33
		σ	3.81	1.97	2.0	1.9	2.29	1.97	1.93	1.7
	Time(s)	μ	0.0472	0.335	0.65	1.1	1.74	1.2	1.6	2.01
		σ	0.0202	0.118	0.278	0.383	0.655	0.614	0.818	0.904

Tabla 3: Comparación general por número de elementos

Capítulo 5

Discusión

Capítulo 6

Conclusión

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. Comput.*, 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.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer US, 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.

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.

Pisinger, D. (2007). The quadratic knapsack problem—a survey. *Discrete Applied Mathematics*, 155(5):623–648.

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.

Sur, G., Ryu, S. Y., Kim, J., and Lim, H. (2022). A Deep Reinforcement Learning-Based Scheme for Solving Multiple Knapsack Problems. *Applied Sciences*, 12(6):3068. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

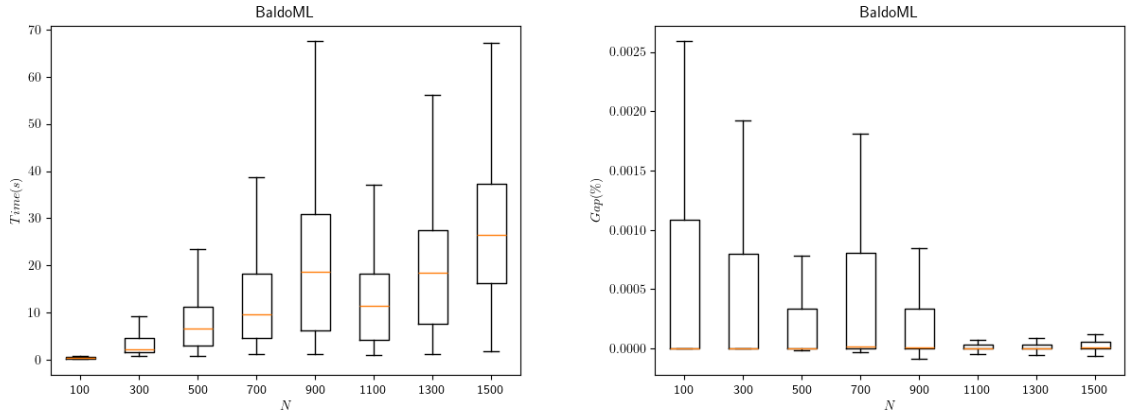
Witzgall, C. (1975). Mathematical methods of site selection for electronic message systems (EMS). Technical Report NBS IR 75-737, National Bureau of Standards, Gaithersburg, MD. Edition: 0.

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

Apéndice A

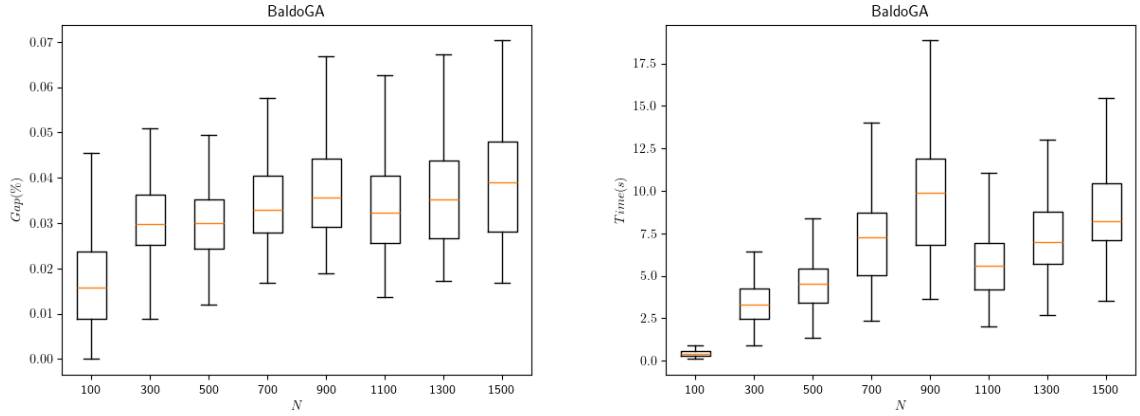
Material

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



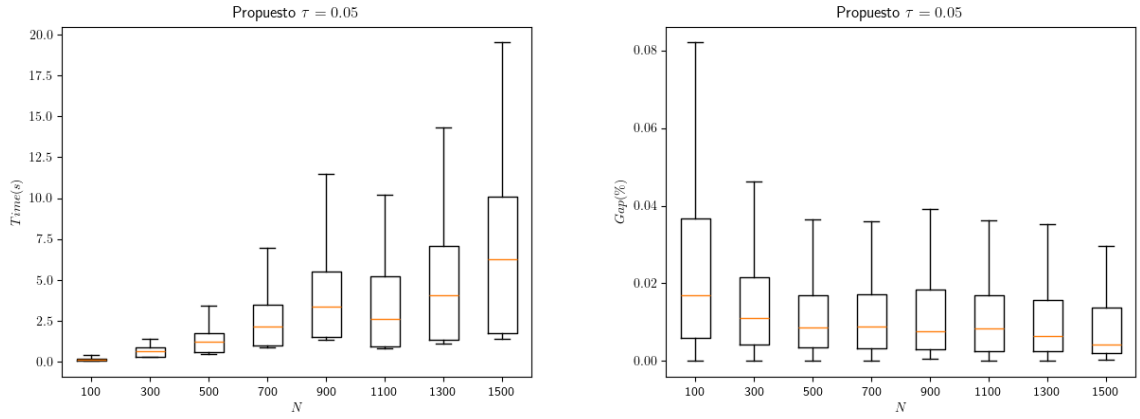
(a) Tiempo vs Numero de elementos **(b)** Gap vs Número de Elementos

Figura 5: Rendimiento BaldoML



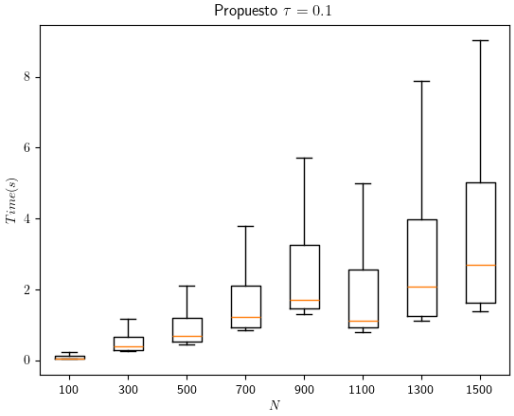
(a) Tiempo vs Numero de elementos **(b)** Gap vs Numero de Elementos

Figura 6: Rendimiento BaldoGA

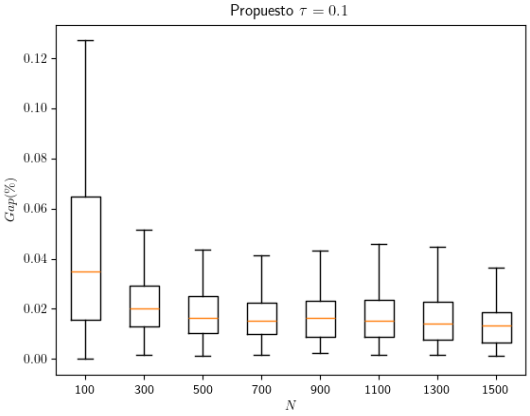


(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 7: Rendimiento Algorimo propuesto ($\tau = 0.05$)

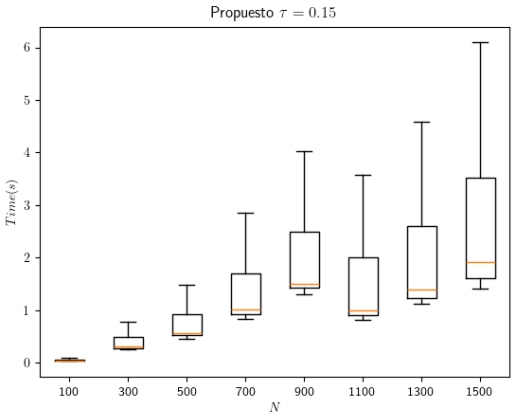


(a) Tiempo vs Numero de elementos

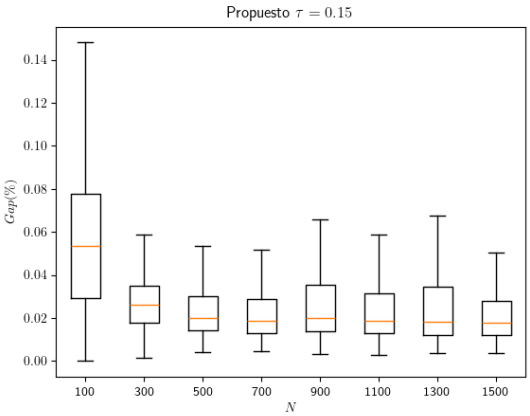


(b) $Gap(\%)$ vs Numero de Elementos

Figura 8: Rendimiento Algorimo propuesto ($\tau = 0.1$)

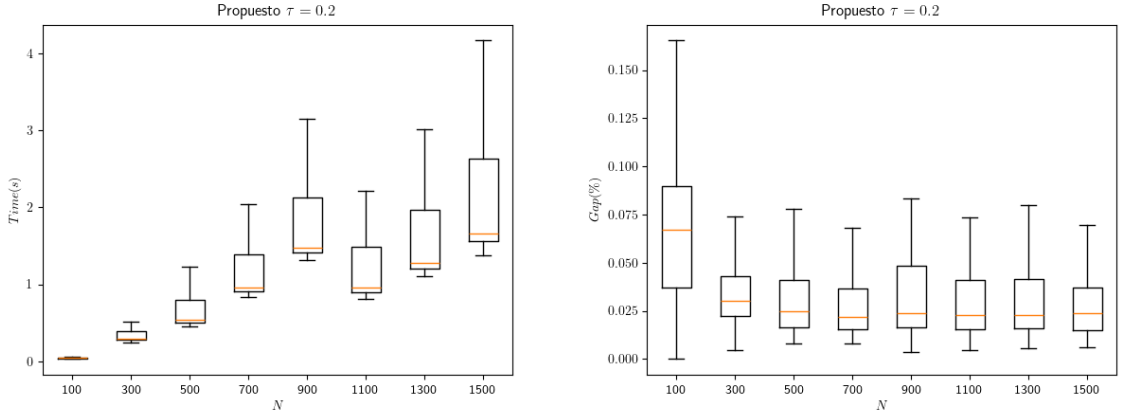


(a) Tiempo vs Numero de elementos



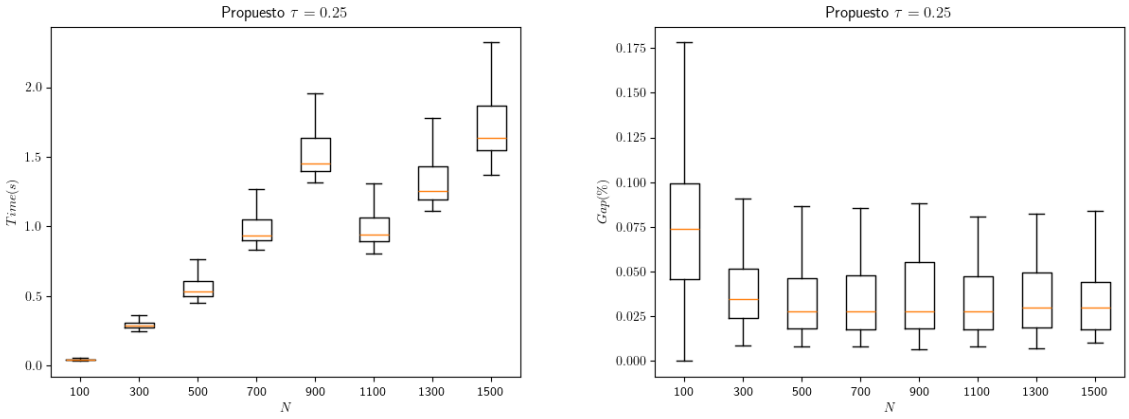
(b) $Gap(\%)$ vs Numero de Elementos

Figura 9: Rendimiento Algorimo propuesto ($\tau = 0.15$)



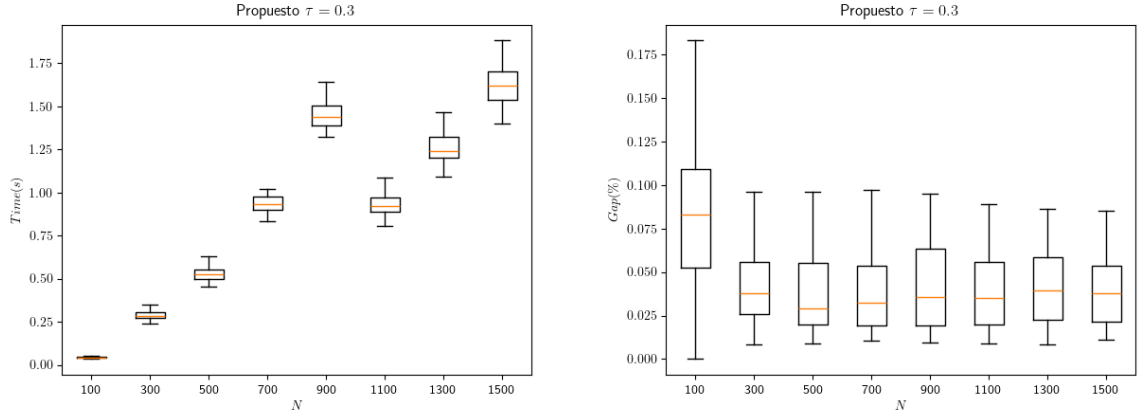
(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 10: Rendimiento Algorimo propuesto ($\tau = 0.2$)



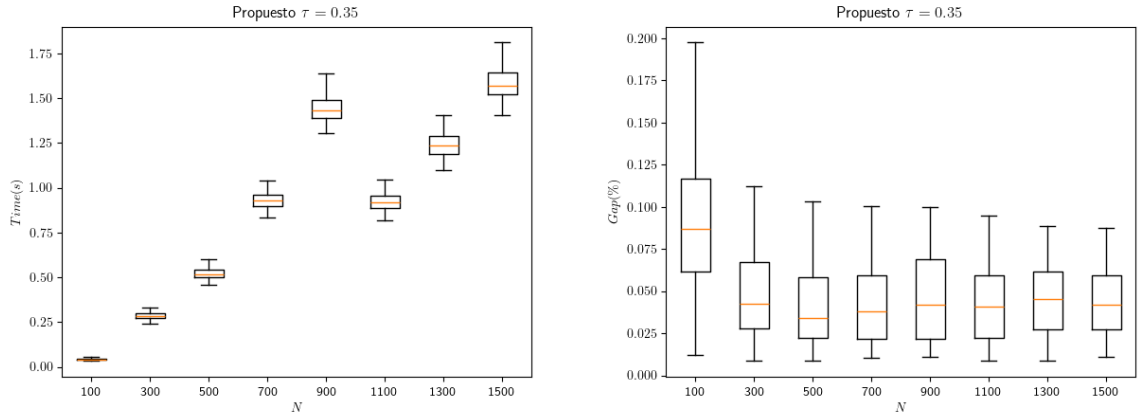
(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 11: Rendimiento Algorimo propuesto ($\tau = 0.25$)



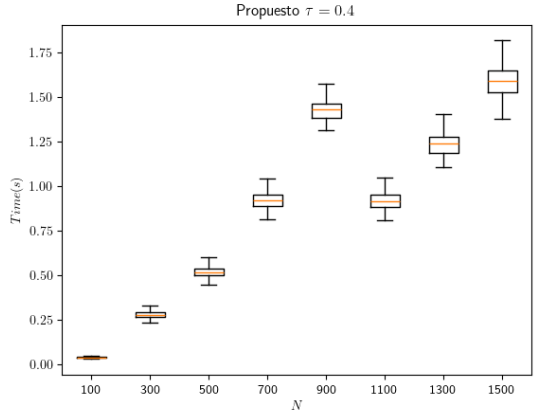
(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 12: Rendimiento Algorimo propuesto ($\tau = 0.3$)

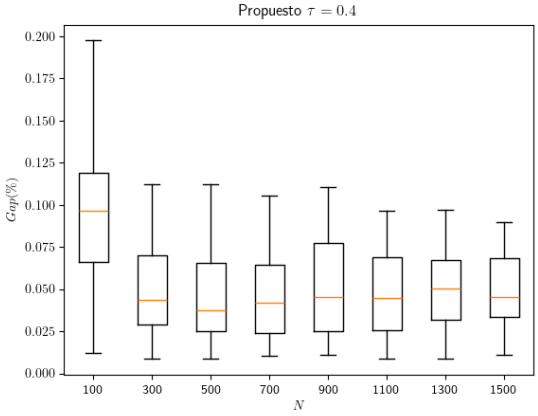


(a) Tiempo vs Numero de elementos **(b)** $Gap(\%)$ vs Numero de Elementos

Figura 13: Rendimiento Algorimo propuesto ($\tau = 0.35$)

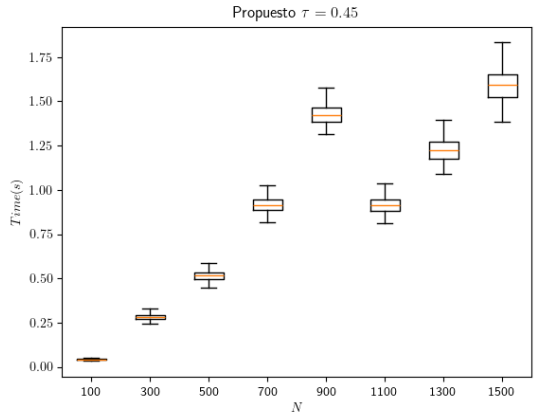


(a) Tiempo vs Numero de elementos

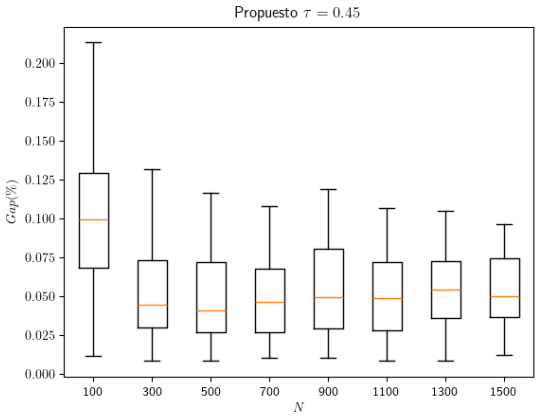


(b) *Gap*(%) vs Numero de Elementos

Figura 14: Rendimiento Algorimo propuesto ($\tau = 0.4$)

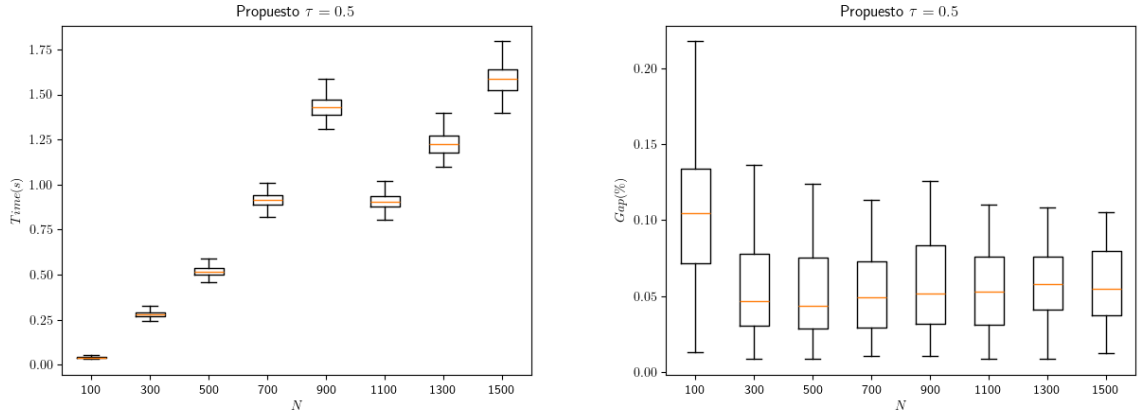


(a) Tiempo vs Numero de elementos



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

Figura 15: Rendimiento Algorimo propuesto ($\tau = 0.45$)



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

Figura 16: Rendimiento Algorimo propuesto ($\tau = 0.5$)

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	0.15	0.0723	0.0751	0.105	0.037	1.03	2.01	2.02
		σ	0.339	0.151	0.202	0.218	0.0736	10.0	14.0	14.0
	Time(s)	μ	0.475	3.74	7.9	12.9	21.1	18.6	30.8	38.2
		σ	0.195	3.25	5.86	11.3	15.9	60.6	83.0	82.2
$\tau_{0.05}$	Gap(%)	μ	2.48	1.34	1.12	1.09	1.14	1.12	1.02	0.839
		σ	2.48	1.08	0.922	0.868	0.996	1.01	0.965	0.837
	Time(s)	μ	0.138	0.665	1.32	2.46	4.08	3.43	4.84	6.68
		σ	0.104	0.391	0.874	1.74	2.98	2.8	3.83	4.69
$\tau_{0.1}$	Gap(%)	μ	4.2	2.19	1.89	1.87	1.92	1.85	1.8	1.58
		σ	3.17	1.18	1.14	1.3	1.33	1.3	1.35	1.19
	Time(s)	μ	0.0914	0.511	0.929	1.6	2.5	2.07	2.84	3.88
		σ	0.0735	0.281	0.562	0.848	1.52	1.7	2.12	3.11
$\tau_{0.15}$	Gap(%)	μ	5.65	2.81	2.49	2.42	2.65	2.42	2.45	2.15
		σ	3.36	1.34	1.47	1.55	1.76	1.6	1.67	1.32
	Time(s)	μ	0.0653	0.415	0.767	1.37	2.18	1.65	2.26	2.9
		σ	0.0491	0.207	0.392	0.66	1.25	1.22	1.6	1.97
$\tau_{0.2}$	Gap(%)	μ	6.68	3.4	2.99	2.9	3.2	2.94	2.97	2.75
		σ	3.55	1.65	1.76	1.76	2.04	1.79	1.8	1.49
	Time(s)	μ	0.0537	0.36	0.689	1.22	1.95	1.41	1.9	2.32
		σ	0.0313	0.148	0.3	0.527	1.03	0.933	1.28	1.25
$\tau_{0.25}$	Gap(%)	μ	7.47	3.94	3.42	3.37	3.7	3.43	3.52	3.33
		σ	3.81	1.97	2.0	1.9	2.29	1.97	1.93	1.7
	Time(s)	μ	0.0472	0.335	0.65	1.1	1.74	1.2	1.6	2.01
		σ	0.0202	0.118	0.278	0.383	0.655	0.614	0.818	0.904
$\tau_{0.3}$	Gap(%)	μ	8.27	4.36	3.76	3.78	4.17	3.87	4.06	3.87
		σ	4.03	2.23	2.16	2.08	2.41	2.13	2.05	1.88
	Time(s)	μ	0.044	0.317	0.611	1.04	1.58	1.06	1.41	1.79
		σ	0.0142	0.0979	0.232	0.299	0.407	0.426	0.538	0.574
$\tau_{0.35}$	Gap(%)	μ	8.89	4.75	4.12	4.21	4.62	4.32	4.53	4.4
		σ	4.17	2.49	2.32	2.25	2.6	2.28	2.15	2.1
	Time(s)	μ	0.0428	0.302	0.575	0.984	1.51	0.982	1.3	1.61
		σ	0.0112	0.0699	0.177	0.199	0.286	0.251	0.272	0.211
$\tau_{0.4}$	Gap(%)	μ	9.5	5.02	4.45	4.57	5.04	4.71	4.96	4.88
		σ	4.26	2.67	2.49	2.4	2.75	2.43	2.27	2.27
	Time(s)	μ	0.0413	0.295	0.55	0.943	1.45	0.938	1.24	1.59
		σ	4.950e-03	0.0571	0.128	0.123	0.161	0.163	0.077	0.0932
$\tau_{0.45}$	Gap(%)	μ	10.0	5.27	4.77	4.94	5.42	5.05	5.32	5.27
		σ	4.35	2.89	2.67	2.56	2.88	2.58	2.38	2.43
	Time(s)	μ	0.0416	0.292	0.535	0.921	1.43	0.918	1.23	1.59
		σ	4.670e-03	0.0485	0.0889	0.0716	0.0948	0.0513	0.0667	0.0893
$\tau_{0.5}$	Gap(%)	μ	10.4	5.49	5.03	5.29	5.73	5.36	5.64	5.61
		σ	4.5	3.08	2.83	2.75	2.98	2.75	2.52	2.61
	Time(s)	μ	0.0416	0.285	0.528	0.917	1.44	0.913	1.23	1.59
		σ	4.760e-03	0.0321	0.0676	0.0413	0.0745	0.0479	0.0702	0.0943

Tabla 4: Tabla completa de resultados

Apéndice B

Ecuaciones

B.1. Features Clasificador

$$F_1(i) = \frac{P_i}{W} \tag{8}$$

$$F_2(i) = \frac{LC_i}{W} \tag{9}$$

$$F_3(i) = \frac{UC_i}{W} \tag{10}$$

$$F_4(i) = \frac{\Gamma}{N} \tag{11}$$

$$F_5(i) = \text{ContSol}_i \tag{12}$$

$$F_6(i) = \text{Random}(i) \tag{13}$$