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

Octubre 2023
Concepción, Chile

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 cuadros

Índice de figuras

Capítulo 1

Introducción

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, que modela generalmente 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 exhaustivamente estudiado debido a su estructura sencilla, y también debido a que muchos otros problemas de optimización más complejos tienen como un subproblema al KP (Martello and Toth, 1990).

El problema tiene muchas variantes, una de las cuales es la versión robusta. El RKP (por sus siglas en inglés, Robust Knapsack Problem) 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 se plantea para encontrar soluciones que sean factibles para todas las posibles variaciones en los costos de los elementos (Monaci et al., 2013).

Otra variante es el problema 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). De este problema surge el problema polinomial robusto de la mochila (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).

Por tanto, este trabajo propone la exploración de técnicas avanzadas de machine learning para resolver el PRKP y obtener resultados más eficientes y cercanos a la solución óptima con base en requerimientos de tiempo, memoria y robustez de las soluciones.

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. Revisión de literatura

El problema polinomial robusto de la mochila es un problema de selección de alternativas, donde, dado un presupuesto, cada alternativa tiene un costo nominal y un costo máximo con el que puede variar. Distintas combinaciones de estas alternativas producen una serie de efectos en el beneficio final. El número de elementos en cada una de estas posibles combinaciones es lo que se considera, el grado de la sinergia polinomial

Existen variedad trabajos enfocados en el QKP y en menor medida para el CKP. Sin embargo, recientemente Baldo et al. (2023) ha introducido por primera vez una metodología para resolver el PRKP, utilizando un algoritmo genético y otro algoritmo basado en machine learning. Este último usa un clasificador random forest predice la probabilidad de cada elemento de estar presente en la solución óptima, para decidir basado en una heurística, si considerar cada elemento o no, fijando los elementos con mayor confianza, y resolviendo para el resto de elementos usando gurobi.

Se han realizado revisiones bibliográficas exhaustivas para el KP, como en Kellerer et al. (2004) y Pisinger (2007) El primero revisa una variedad de definiciones para el problema, así como métodos exactos, algoritmos aproximados, versiones relajadas y descripciones de variaciones comunes del problema en las que se encuentra el QKP. Mientras que Pisinger (2007), se enfoca directamente en el QKP, revisando multitud de enfoques para solucionarlo, descomposiciones y relajaciones lagrangianas, linealizaciones de los parámetros y otras metodologías avanzadas. Por otro lado, el CKP ha sido abordado por Forrester and Waddell (2022) mejorando las formulaciones lineales clásicas para resolver este problema.

Es interesante revisar enfoques basados en machine learning para abordar este tipo de problemas. 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) usa 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, para luego, mediante el descenso de gradiente y el uso de características de los elementos, decidir cuáles de los anteriormente excluidos, agregar a la solución inicial obtenida. El modelo de machine learning usado para evaluar que elementos son incluidos muestra resultados competitivos con los demás clasificadores e impactos en tiempo computacional insignificantes.

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 voraz. El algoritmo propuesto construye las soluciones con base en las decisiones del modelo y genera soluciones con una razón de similitud con el óptimo del 99.9 % usando una arquitectura de A2C con un paso de cuantización de características.

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 2

Marco teórico

2.1. Modelo matemático

En la formulación de Baldo et al. (2023) se describe la formulación del PRKP y realiza una linearización para transformar el problema 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 linearización del problema y se referiŕa a la primera formulación de baldo y se reservará su linearización para uso exclusivo como modelo de PL.

La formulación clásica 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 2.1.1

$$x_i = \begin{cases} 1 & \text{si el elemento } i \text{ es elegido} \\ \\ 0 & \text{si el elemento } i \text{ no es elegido} \end{cases} \tag{2.1.1}$$

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:

$$\left(\sum_{i=1}^{I} LC_i \cdot x_i\right) + \text{máx}\left(\sum_{i=1}^{I} \Delta C_i \cdot y_i\right) \leq W \tag{2.1.2}$$

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

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

2.1.0.1. 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 2.1.3 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.

$$\sum_{A \in S} \left(PS_A \cdot \prod_{i \in A} x_i \right) \tag{2.1.3}$$

2.1.0.2. Función objetivo

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

$$\text{máx}\, f(x) = \sum_{i=1}^{I} p_i \cdot x_i + \sum_{A \in S} \left(PS_A \cdot \prod_{i \in A} x_i \right) - \left(\sum_{i=1}^{I} LC_i \cdot x_i + \text{máx} \sum_{i=1}^{I} \Delta C_i \cdot y_i \right) \tag{2.1.4}$$

Sujeto a la restricción de presupuesto

$$\sum_{i=1}^{I} LC_i \cdot x_i + \text{máx} \sum_{i=1}^{I} \Delta C_i \cdot y_i \leq W \tag{2.1.5}$$

2.1.1. Complejidad

Si bien la complejidad del espacio de búsqueda del KP tradicional es de $O\left(2^n\right)$, la complejidad del problema ha demostrado ser $O\left(nW\right)$ usando técnicas de branch and bound y acercamientos de programación dinámica, no así con esta variante.

La dificultad de trabajar con el PRKP está en su complejidad espacial de $O\left(2^n\right)$ asociada a las sinergias polinomiales, cada posible combinación de elementos puede tener un beneficio independiente, por lo que cualquier algoritmo que resuelva el problema de forma exacta debe, como mínimo, leer el espacio de búsqueda.

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

$ITo Fix History$ ▷ 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
  $Ito Fix History \leftarrow ITo Fix$
  $Instance \leftarrow Reduce(Instance, ITo Fix)$ ▷ Instancia reducida
  if $Ito Fix == 0$ **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 2.1.2, muestra que para calcular la restricción de presupuesto, y 2.1.4 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 solucion, pero mas importante, todas las sinergias polinomiales que poseían elementos en Z, ya no son alcanzables por ninguna solucion, por lo que es seguro eliminarlas de S, notar tambien que Γ no es modificado.

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

- En cada paso, la solucion optima de cada iteracion de la instancia estará compuesta por una proporcion mayor de unos que de ceros. La proporcion 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 2.1.2 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 item 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

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.

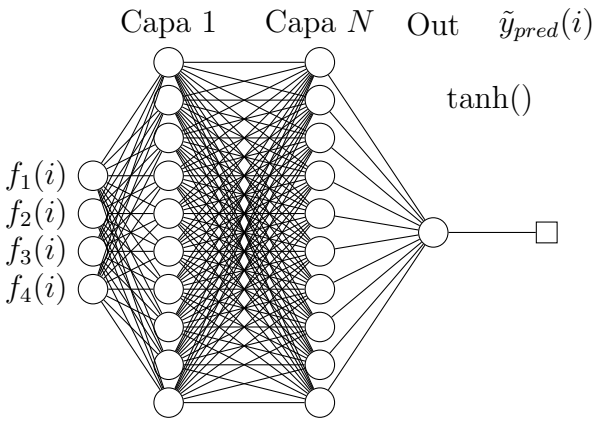

Figura 3.3.1: Estructura general de la red

Como se muestra en la figura 3.3.1, la red no intenta predecir el valor del x_i optimo directamente, sino que predice la transformación 3.3.1, 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 $\tilde{x_i} = y^{-1}(\tilde{y}(i))$

$$y(i) = \begin{cases} 1, & \text{si } x_i = 1 \\ -1, & \text{si } x_i = 0 \end{cases} \tag{3.3.1}$$

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:

$$F(I) = \begin{pmatrix} f(1) \\ f(2) \\ \vdots \\ f(N-1) \\ f(N) \end{pmatrix} \tag{3.3.2}$$

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:

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

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**
  $x \leftarrow Solver(Instance)$ ▷ Vector booleano x con la solución óptima de la instancia
  for $i \in I$ **do**
    $\tilde{x_i} \leftarrow Net(f(i))$ ▷ Prediccion de la red de x_i desde las features
    $loss \leftarrow Loss(x_i, \tilde{x_i})$
    $Net \leftarrow Optimizer(Net, loss)$ ▷ 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 ← OInstance
    loop
        x̃ ← Net(F(I))                        ▷ Vector de predicción de la red
        ZHistory ← Z(x̃, τ, μ)
        Instance ← Reduce(Instance, Z(x̃, τ, μ))   ▷ Reducir la instancia
        if #Z == 0 or mín(x̃) ≤ τ then
            break
        end if
        x ← Solver(Instance)                 ▷ Solución óptima de la instancia reducida
        x̃ ← Net(F(I))                        ▷ Predigo para la instancia reducida
        for i ∈ I do
            x̃_i ← Net(f(i))
            loss ← Loss(x_i, x̃_i)
            Net ← 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.

3.4. 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 generacion 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 usara el mismo conjunto de instancias.

Estas instancias se ejecutarán resueltas en un i7-8550U con 4 nucleos 8 hilos,

32GB ram en Linux 6.6.1.

Capítulo 4

Resultados Experimentales

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

Apéndice B

Ecuaciones

Features entrenamiento 1 del clasificador

$$F_1(i) = \frac{P_i}{W} \tag{B.0.1}$$

$$F_2(i) = \frac{LC_i}{W} \tag{B.0.2}$$

$$F_3(i) = \frac{UC_i}{W} \tag{B.0.3}$$

$$F_4(i) = \frac{\Gamma}{N} \tag{B.0.4}$$

$$F_5(i) = \mathrm{CSol}_i \tag{B.0.5}$$

$$F_6(i) = \frac{\#\{A \in S, i \in A\}}{\#S} \tag{B.0.6}$$

$$F_7(i) = \frac{\#\{A \in S, i \in A : PS_A > 0\}}{W} \tag{B.0.7}$$

$$F_8(i) = \frac{\#\{A \in S, i \in A : PS_A < 0\}}{W} \tag{B.0.8}$$

$$F_9(i) = \frac{\sum_{A:i\in A}^{S} PS_A}{\sum_{A}^{S} PS_A} \tag{B.0.9}$$

$$F_10(i) = \mathrm{Random}(i) \tag{B.0.10}$$