UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

UNA MATHEURÍSTICA PARA LA RESOLUCIÓN DEL SET UNION KNAPSACK PROBLEM

POR

Alex Javier Cañete Saavedra

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

Profesor Guía
Carlos Contreras Bolton

Marzo 2022
Concepción (Chile)

A quienes, con su amor incondicional e infinito, me salvaron de mí.

Resumen

El Set Union Knapsack Problem (SUKP) es una generalización del problema de la mochila (KP) clásica, donde el objetivo es seleccionar desde un conjunto de ítems que aportan beneficio, aquellos que maximizan este último, sujeto a una restricción de capacidad. En el caso del SUKP, estos ítems se forman por elementos que son los que ocupan capacidad. Esta característica convierte al SUKP en un problema de complejidad mayor al KP clásico, esto último sumado a las múltiples aplicaciones en sistemas de producción, finanzas, bases de datos, entre otras, otorgan gran relevancia a la investigación de este problema.

En este trabajo se presenta una matheurística para la resolución del SUKP, que utiliza soluciones parciales creada mediante muestreo aleatorio, generando instancias reducidas que luego son resultas mediante CPLEX y/o MSBTS, algoritmo del estado del arte dedicado a resolver el SUKP. Se realiza una revisión de la literatura del estado del arte de los métodos para resolver el SUKP. Se reportan resultados computacionales sobre 30 instancias ejecutadas, siendo estos resultados comparados con otros algoritmos del estado del arte. Los resultados muestran un buen desempeño del algoritmo propuesto, muchas veces cercano a los métodos de vanguardia,destacando una brecha de rendimiento menor al 0.5 % en el mejor valor encontrado para la función objetivo.

Palabras clave – KP, SUKP, Matheurística, CPLEX, Optimización Combinatoria

Abstract

The Set Union Knapsack Problem (SUKP) is a generalization of the classical knapsack problem (KP), where the objective is to select from a set of items that provide benefit, those that maximize this benefit, subject to a capacity restriction. In SUKP case, these items are formed by elements that occupy capacity. This characteristic makes the SUKP a problem of greater complexity than the classical KP, the latter added to the multiple applications in production systems, finance, databases, among others, give great relevance to the investigation of this problem

In this work we present a matheuristic for the resolution of the SUKP, which uses partial solutions created by random sampling, generating reduced instances that are then solved by CPLEX and/or MSBTS, the state of the art algorithm dedicated to solve the SUKP. A literature review of state of the art methods for solving SUKP is performed. Computational results on 30 executed instances are reported, being these results compared with other state of the art algorithms. The results show a good performance of the proposed algorithm, many times close to state of the art methods, highlighting a performance gap of less than 0.5 % at the best value found for the objective function.

Keywords – KP, SUKP, Matheristic, CPLEX, Combinatorial optimization

Índice General

AGRADECIMIENTOS I

Resumen I

Abstract II

1. Introducción 1
- 1.1. Motivación . . . 1
- 1.2. Objetivo general . . . 3
- 1.3. Objetivos específicos . . . 3
- 1.4. Estructura del documento . . . 3

2. Set Union Knpasack Problem 5
- 2.1. Descripción del problema . . . 5
  - 2.1.1. Modelo matemático . . . 6
- 2.2. Revisión de la literatura . . . 8

3. Matheurística 12
- 3.1. Esquema general . . . 12
- 3.2. Preprocesamiento . . . 14
- 3.3. Generación de soluciones parciales válidas . . . 15
- 3.4. Reducción de instancia . . . 16
- 3.5. Solución de la instancia reducida . . . 18
  - 3.5.1. Solución usando CPLEX . . . 18
  - 3.5.2. Solución usando MSBTS . . . 18
- 3.6. Solución de la instancia inicial . . . 19
- 3.7. Parámetros involucrados y calibración . . . 19

4. Resultados y Análisis 21
- 4.1. Instancias de prueba . . . 21
- 4.2. Implementación . . . 22
- 4.3. Resultados computacionales . . . 22
- 4.4. Comparación con el estado del arte . . . 24

5. Conclusión 27

5.1. Conclusiones 27
5.2. Trabajo futuro 27

Referencias **29**

Apéndices **31**

Índice de Tablas

1. Parámetros y Valores Respectivos 20

2. Enumeración y Clasificación de las Instancias. 22
3. Resultados obtenidos por el algoritmo propuesto para las 30 instancias. 23
4. Medias de los resultados obtenidos por los algoritmos considerando las 30 instancias. 25
5. Medias de los resultados obtenidos por los algoritmo considerando solo instancias con $a = 0{,}15$ y $b = 0{,}85$. 25
6. Comparación resumida entre el algoritmo propuesto y los usados como comparación. 26

A.7. Resultados de la función objetivo obtenidos para cada ejecución e instancias 32
A.8. Resultados del tiempo en que el algoritmo alcanzo el valor máximo de la función objetivo por ejecución e instancia 33
A.9. Resultados de los algoritmos utilizados de forma comparativa. . . 35

Índice de Figuras

1. Una instancia del SUKP. 6

Capítulo 1

Introducción

En este capítulo se presenta el SUKP y algunas de sus aplicaciones son expuestas. También, la motivación detrás de este trabajo es abordada. Por último, el objetivo general y los específicos son enumerados.

1.1. Motivación

Consideremos una simple decisión binaria, es decir, debemos seleccionar una preferencia entre dos opciones, por ejemplo: viajar en auto o bicicleta, comprar una casa o un departamento, vacacionar dentro del país o fuera, etc. Luego de estos ejemplos, no cuesta mucho comprender que nuestras vidas se componen de un sinfín de decisiones binarias, no solo en lo personal, este tipo de elecciones también se encuentran muy presentes en el mundo profesional. Para facilitar la toma de este tipo de decisiones es usual abstraer la disyuntiva, parametrizando dos propiedades intrínsecas y muy evidentes de cada alternativa, el beneficio que genera cada opción y su costo asociado. A partir de lo anterior, es posible generar un modelo de optimización matemática, cuya solución nos dirá, bajo las consideraciones y supuestos establecidos, cual es la mejor opción dentro de las existentes.

El *Knapsack Problem* (KP)(problema de la mochila en español) es uno de los modelos más simples de programación matemática con variables binarias, pero no por ello trivial. Una de sus interpretaciones más usuales hace referencia a la existencia de una mochila física, con capacidad limitada c (en kg para este ejemplo)

y un conjunto de ítems N. Donde cada ítem tiene un beneficio y peso asociado, usualmente denotados por p y w respectivamente. El objetivo es seleccionar un subconjunto de N, que genere el máximo beneficio sin sobrepasar la capacidad c de la mochila.

KP y sus múltiples extensiones han sido ampliamente estudiadas, no solo por los problemas que resuelven y su aplicación transversal dentro de la sociedad, sino también porque son utilizadas muy ampliamente como subproblemas en algoritmos que resuelven problemas mucho más complejos (Cho, 2019).

Existen abundantes generalizaciones de este problema, de las más destacadas y conocidas: Subset Sum Problem (Kellerer *et al.*, 2004c), Unbounded KP (Kellerer *et al.*, 2004d), Multiple KP (Kellerer *et al.*, 2004b), Multidimensional KP (Kellerer *et al.*, 2004a).

En este trabajo se aborda el *Set Union Knapsack Problem* o *SUKP*, introducido en Goldschmidt *et al.* (1994). Esta generalización se plantea como sigue:

> Sea N un conjunto de ítems y M un conjunto de elementos, donde cada ítem $i \in N$ es un subconjunto de M. Cada ítem i posee un beneficio positivo p_i y cada elemento $j \in M$ tiene un peso no negativo w_j. Sea S un conjunto de ítems tal que $S \subseteq N$, el peso de S está dado por la suma de los pesos de los elementos pertenecientes al conjunto originado por la unión de los ítems en S. El objetivo del problema es seleccionar el conjunto de ítems que maximicen el beneficio sin que el peso de S supere la capacidad c.

El SUKP tiene múltiples aplicaciones, por ejemplo, sistemas de manufactura (Goldschmidt *et al.*, 1994). Consideremos una compañía manufacturera que quiere ampliar su producción, para ello selecciona un conjunto de productos desde una lista de candidatos. Para comenzarlos a producir, cada producto candidato posee un beneficio asociado y está compuesto por un conjunto de componentes más pequeños que deben ser ensamblados. Para producir estos componentes nuevas máquinas deben ser adquiridas, cada maquina puede producir un tipo de componente pero este componente puede ser producido múltiples veces. El objetivo es seleccionar el conjunto de productos nuevos que maximice el beneficio sin exceder el presupuesto para la compra de maquinas.

El SUKP también puede ser utilizado para resolver problemas en finanzas (Kellerer

et al., 2004a) y bases de datos (Navathe *et al.*, 1984; Wei y Hao, 2021a).

El SUKP pertenece a la clase NP-Hard, incluso en condiciones muy restringidas (Goldschmidt *et al.*, 1994), lo que lo vuelve computacionalmente muy complejo. El diseño de nuevos y mejores algoritmos para la resolución del SUKP, no solo ayudarían a múltiples industrias que lo aplican directamente, sino que también aportaría a una comprensión mayor de la clase NP-Hard, facilitando la resolución de todos sus problemas miembros.

En este trabajo se presenta una matheurística para la resolución del SUKP, que utiliza soluciones parciales creadas mediante muestreo aleatorio, generando instancias reducidas que luego son resueltas por un método exacto y/o una matheurística. Se ejecutan experimentos computacionales en instancias desde 85 a 500 ítems y elementos.

1.2. Objetivo general

Implementar una matheurística para el SUKP.

1.3. Objetivos específicos

1. Revisar el estado del arte de los métodos para resolver el SUKP.
2. Diseñar una matheurística para resolver el SUKP.
3. Implementar la matheurística diseñada.
4. Medir el desempeño del método propuesto.
5. Comparar los resultados obtenidos con el estado del arte.

1.4. Estructura del documento

El resto del documento se estructura como sigue. En el Capítulo 2, se presenta el problema formalmente, incluyendo modelos matemáticos relacionados y una revisión de la literatura enfocada en métodos que permitan resolverlo. Continúa el Capítulo 3 con el desarrollo y explicación de la matheurística propuesta, presentando pseudocódigos y descripción de su funcionamiento. Luego, el Capítulo

4 aborda los experimentos computacionales y resultados obtenidos. Finalmente, los resultados se discuten y analizan en el Capítulo 5, presentando también las conclusiones.

Capítulo 2

Set Union Knpasack Problem

En este capítulo se expone de forma extensa el problema, los modelos matemáticos asociados a él. También se revisa la literatura más relevante y reciente asociada al SUKP, enfocándose en los algoritmos del estado del arte que resuelven el SUKP y sus características.

2.1. Descripción del problema

El SUKP es una generalización del KP, compartiendo el mismo objetivo, maximizar el beneficio sin exceder la capacidad de la mochila. Una instancia de este problema puede ser caracterizada por dos conjuntos: $M = \{1, 2, 3, \ldots, m\}$ compuesto por elementos y $N = \{1, 2, 3, \ldots, n\}$ formado por items, donde cada item i perteneciente a N es un subconjunto de M, es decir $i \subset M$. Cada ítem i posee un beneficio no negativo p_i, cada elemento $j \in M$ posee un peso no negativo w_j y la mochila tiene una capacidad C. Notar que si: $i_1 \bigcap i_2 = \emptyset, \forall i_1, i_2 \in N$ el SUKP se transforma en el KP estándar.

Sea $S \subset N$, defínase $E(S) \subset M$ como el conjunto de elementos asociados a S (Ecuación (1)), $W(S)$ es el peso total de los elementos cubiertos por S (Ecuación (2)) y $F(S)$ es el beneficio total de S (Ecuación (3)). Luego, S es una solución factible si y solo si $W(S) \leq C$, con beneficio $F(S)$.

$$E(S) = \bigcup_{i \in S} i, \qquad S \subseteq N \tag{1}$$

$$W(S) = \sum_{j \in E(S)} w_j, \qquad S \subseteq N, E(S) \subseteq M \tag{2}$$

$$F(S) = \sum_{i \in S} p_i, \qquad S \subseteq N \tag{3}$$

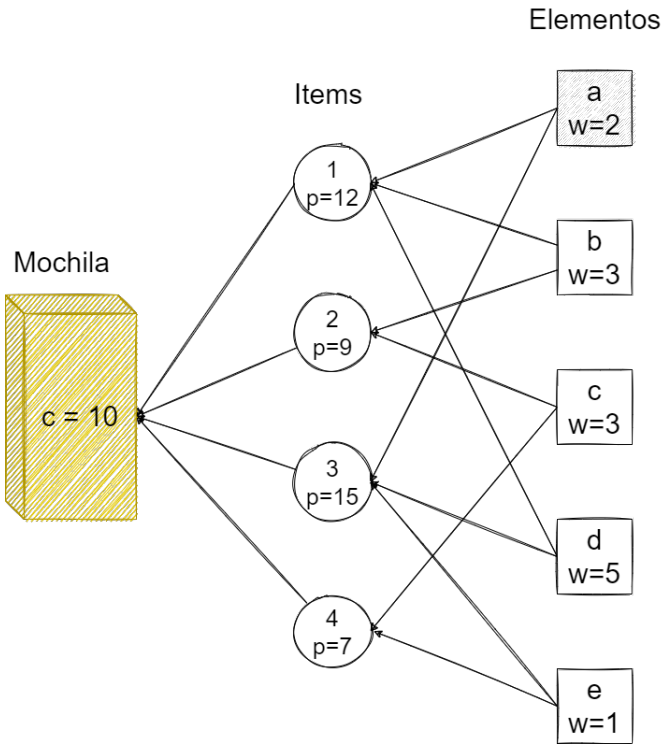

Figura 1: Una instancia del SUKP.
Fuente: Elaboración propia.

La Figura 1 ilustra una instancia pequeña del problema formada por cuatro ítems, cinco elementos y con $C = 10$. Consideremos $S_1 = \{2, 4\}$, por lo tanto $E(S_1) = \{b, c, e\}$, $F(S_1) = 16$, $W(S_1) = 7$. En consecuencia, S_1 es una solución factible para esta instancia.

2.1.1. Modelo matemático

Formalmente, el SUKP puede ser formulado con el modelo matemático propuesto por Goldschmidt *et al.* (1994) como sigue:

$$\max\ F(S) = \sum_{i \in S} p_i \tag{4}$$

$$\text{s.a.:} \qquad W(S) = \sum_{j \in E(S)} w_j \leq C \tag{5}$$

$$S \subseteq N \tag{6}$$

El problema también puede ser definido como un modelo de programación binaria (Wei y Hao, 2019). Para esto, es necesario definir dos variables de decisión binaria, una asociadas a los ítems y otra a los elementos:

$$x_i = \begin{cases} 1 \text{ si el ítem } i \text{ pertenece a la solución.} \\ 0 \text{ si el ítem } i \text{ no pertenece a la solución.} \end{cases}$$

$$y_j = \begin{cases} 1 \text{ si el elemento } j \text{ pertenece a la solución.} \\ 0 \text{ si el elemento } j \text{ no pertenece a la solución.} \end{cases}$$

También, se define $r_j \subseteq N$ como el conjunto de ítems a los que pertenece el elemento j, es decir, $i \in r_j \iff j \in i$. Luego, es posible definir el siguiente modelo de programación binaria:

$$\max \sum_{i \in N} x_i p_i \tag{7}$$

$$\text{s.a.:} \qquad \sum_{j \in M} y_j w_j \leq C \tag{8}$$

$$|r_j| y_j \geq \sum_{i \in r_j} x_i \qquad \forall j \in M \tag{9}$$

$$y_j \leq \sum_{i \in r_j} x_i \qquad \forall j \in M \tag{10}$$

$$x_i \in \{0, 1\} \qquad \forall i \in N \tag{11}$$

$$y_j \in \{0, 1\} \qquad \forall j \in M \tag{12}$$

Donde la Ecuación (7) es la función objetivo que maximiza el beneficio, la

restricción (8) es la capacidad de la mochila. Las restricciones (9) y (10) se relacionan con las variables de decisión. Por último, las restricciones (11) y (12) restringen el dominio de las variables.

2.2. Revisión de la literatura

La literatura es abundante para métodos exactos como no exactos, que resuelven el SUKP. Sin embargo, en los últimos años los esfuerzos se han centrado en desarrollar métodos heurísticos, que aunque no garanticen encontrar soluciones óptimas, retornan soluciones de buena calidad en tiempos reducidos. A continuación, se presentan algunos de los trabajos más relevantes para la resolución del SUKP.

Goldschmidt *et al.* (1994) introduce el SUKP y un método de programación dinámica de complejidad exponencial, basándose en el método de programación dinámica existente para resolver el KP clásico. Caracterizaron además, instancias del problema que permiten una resolución en tiempo polinomial con su algoritmo.

He *et al.* (2018), plantean un Binary Artificial Bee Colony Algorithm (BABC), una adaptación al típico algoritmo de colonia artificial de abejas (ABC). Dado que el ABC en su forma tradicional está diseñado para problemas en un espacio real, los autores generan una función que mapea un espacio continuo D-dimensional, a uno D-dimensional binario. Debido a la estrategia utilizada, se generan soluciones infactibles, lo cual se corrige con otro algoritmo propuesto. Este algoritmo genérico puede ser utilizado en otros algoritmos evolutivos para resolver el SUKP. Otro aporte considerable de este trabajo, es la creación de 30 instancias, desde 85 a 500 elementos e ítems, para el SUKP. También, los autores implementan: A-SUKP (Arulselvan, 2014), ABC$_{bin}$ (Kiran, 2015), Algorítmo Génetico (GA), (Schmitt, 2001), Binary Differential Evolution Algorithm (binDE) (Engelbrecht y Pampara, 2007), con el objetivo de compararlo al BACB. El algoritmo propuesto tuvo un rendimiento mayor a sus competidores, obteniendo mejores resultados, de forma más estable y rápida.

Lin *et al.* (2019), agregan a la literatura un algoritmo híbrido, que combina Binary Particle Swarm Optimization (HBPSO) y Tabu Search (TS). Para gestionar la infactibilidad de las soluciones generadas, utilizan una función de penalización, disminuyendo así el valor objetivo de soluciones infactibles para que el algoritmo las catalogue como soluciones de baja calidad. Esta función es dinámica, pues

depende del máximo global encontrado. El algoritmo comienza con una solución aleatoria, la que es mejorada mediante TS, que sirve como parámetro preliminar para el BPSO. Luego, se genera una población inicial de partículas, que representan soluciones, factibles o no, para la instancia. En cada iteración una partícula es seleccionada de forma aleatoria, para ser actualizada mediante el mecanismo de acción principal del BPSO, para luego refinar la solución mediante dos TS consecutivas de características similares. Con un ajuste adecuado de parámetros, los autores lograron manejar la inclusión del espacio no factible dentro de la búsqueda, obteniendo mejores resultados en cuanto a la calidad de las soluciones pero con un mucho mayor costo computacional, en comparación a lo presentado en He *et al.* (2018).

Wei y Hao (2019), proponen un iterated two-phase local search. Su método genera una solución inicial mediante un método greedy. Luego, emplean un Variable Neighborhood Descent, utilizando dos vecindarios creados por operadores swap, reduciendo la cardinalidad del segundo vecindario de forma probabilística. Posteriormente, aplican una TS que utiliza un operador swap para generar su vecindario de búsqueda, para intensificar aún más su búsqueda. Finalmente, para escapar de mínimos locales, perturban su solución, y comienzan una nueva iteración utilizando la solución perturbada como entrada. Amplían su contribución utilizando CPLEX para resolver de forma exacta algunas instancias empleadas frecuentemente en la literatura.

Wei y Hao (2021a), proponen un algoritmo heurístico basado en TS, búsqueda local y el concepto de Kernel o Blakbone. Este concepto establece que hay ítems que siempre están presentes en soluciones de alta calidad. También, proponen un nuevo set de 30 instancias, de 585 a 1000 ítems y elementos. Los resultados de su algoritmo son robustos, pues logra encontrar mejores cotas inferiores, o alcanzar las existentes, en menor tiempo y con menor dispersión.

Wei y Hao (2021b), proponen un Multistart solution-based Tabu Search. La búsqueda tabú es una metaheurística cuyo desempeño se puede ver afectado por las soluciones iniciales que le son provistas. Por lo anterior, los autores proponen una metodología greedy, con un componente estocástico, para generar una población inicial de soluciones factibles, las cuales funcionan como entrada al Solution Based Tabu Search (SBTS). Su desempeño en instancias pequeñas es similar a los mejores algoritmos conocidos. Sin embargo, es en las instancias de mayor envergadura

donde alcanza a los algoritmos del estado del arte, estableciendo nuevas cotas inferiores para dichas instancias, en menores tiempos computacionales y con menor dispersión. A la fecha, es el mejor algoritmo heurístico conocido para resolver el SUKP.

Dahmani *et al.* (2021), proponen un Iterative Rounding Search-Based Algorithm. Su método comienza generando un conjunto de soluciones parciales mediante un proceso iterativo de relajaciones lineales, añadiendo en cada iteración ítems a la solución parcial. Luego, con una técnica greedy complementan las soluciones parciales generadas anteriormente. Finalmente, métodos de exploración y diversificación son utilizados. El algoritmo logra un desempeño mayor al algoritmo propuesto por Wei y Hao (2019). Los autores no realizaron la comparación con los métodos propuestos por Wei y Hao (2021a) y Wei y Hao (2021b), por lo que su desempeño en comparación a estos algoritmos es desconocido.

Durgut *et al.* (2021) proponen un RLABC, un esquema adaptativo de selección de operador basado en Reinforcement Learning (RL). El esquema adaptativo se obtiene al fusionar Q-Learning y Hard-C-Means, algoritmos de RL y clustering respectivamente, donde cada operador es representado por un cluster. Utilizan como operadores tres variaciones del algoritmo ACB, corrigiendo las infactibilidades generadas mediante el algoritmo S-GROA (He *et al.*, 2018). Los resultados computacionales obtenidos al ejecutar los experimentos bajo las 30 instancias presentadas por He *et al.* (2018), señalan que la estrategia empleada por los autores produce mejores resultados que otros esquemas adaptativos diseñados para problemas binarios.

Dahmani *et al.* (2022) los autores proponen un Particle Swarm Optimization aplicando Backtracking (PSO-B). PSO es un algoritmo evolutivo basado en población que converge a óptimos locales. Para acelerar esta convergencia los autores utilizan un proceso de búsqueda que combina dos algoritmos, un Iterative Search Procedure y 2-opt, el primero explora el vecindario de una solución mediante un mecanismo Greedy, mientras que el segundo es un algoritmo de búsqueda local de profundidad. La estrategia de Backtracking empleada por los autores está basada en un Evolutionary Path Relinking, algoritmo que genera y explora una trayectoria entre dos soluciones existente, lo que permite intensificar la búsqueda alrededor de las soluciones dadas. Los autores miden el desempeño de su algoritmo efectuando los experimentos en las instancias introducidas por He *et al.* (2018) y

en otro grupo de instancias de mayor escala donde el número de ítems y elementos varían de 600 a 1000, presentadas en Dahmani *et al.* (2021). Al comparar PSO-B con otros métodos evolutivos basados en poblaciones, resulta ser un método competitivo, resaltando su desempeño en las instancias de mayor escala.

Wu *et al.* (2022), proponen un Item based Self adjusting Teaching Learning based Optimazation Algorithm (I-DTLBO). Los autores utilizan como base un Teaching Learning Based Optimization Algorithm, una metaheurística basada en población diseñada para problemas continuos, empleando la misma función de mapeo introducida por He *et al.* (2018) para migrar de un espacio continuo a uno binario. Esta migración produce soluciones infactibles las cuales son reparadas mediante un Item based Self adjusting Repair and Optimization operator, algoritmo introducido en este mismo trabajo que considera el dinamismo de los pesos de los ítems para reparar las soluciones infactibles. Para mejorar la capacidad exploratoria del algoritmo propuesto, se utiliza un operador de mutación en cada iteración sobre la mejor solución encontrada. Los resultados obtenidos sobre las treinta instancias más pequeñas del SUKP (He *et al.*, 2018), superan los obtenidos por otros algoritmos de inteligencia de enjambre, o swarm intelligence, obteniendo mejores soluciones y de manera más estable. En el mismo trabajo, se presenta un Element based Self adjusting Teaching Learning based Optimazation Algorithm (E-DTLBO), que utiliza un Element based Self adjusting Repair and Optimization operator para reparar soluciones infactibles. E-DTLBO fue ampliamente superado por I-DTLBO.

Capítulo 3

Matheurística

En este capítulo se presenta el algoritmo propuesto de forma agregada y desagregada. Se presentan pseudocódigos de sus partes principales, describiendo paso a paso el funcionamiento de la matheurística. Luego, se presentan los parámetros del algoritmo y como estos fueron calibrados.

3.1. Esquema general

El algoritmo se basa en la idea propuesta en Dahmani *et al.* (2021), construir soluciones parciales (lo que se entiende como cualquier solución en la que se puedan incorporar uno o más ítems sin que deje de ser factible), reduciendo la instancia y enfocándose en vecindades de interés, para luego utilizar CPLEX y/o MSBTS (Wei y Hao, 2021b) para resolver esta instancia reducida y complementar la solución parcial ya generada. Esta idea también se sustenta en lo propuesto en Wei y Hao (2021a), donde los autores utilizan el concepto de kernel, blackbone o núcleo, en su algoritmo. Aquí, se asocia la frecuencia de aparición de los ítems en las soluciones generadas como una propiedad positiva. Luego, en una etapa posterior los ítems con una frecuencia que supere cierto umbral se utilizan para formar una solución parcial factible.

En el Algoritmo 1 se puede observar la estructura general de la matherurística propuesta. En la línea 1, se comienza con un preprocesamiento de la instancia I_1, donde se generan soluciones utilizando un método greedy con componentes aleatorios, retornando la cantidad promedio de ítems en las soluciones generadas

l, el conjunto con los l ítems más frecuentes F, el conjunto de ítems no frecuentes NF, y el conjunto de las mejores soluciones generadas B, que también almacena las soluciones de alta calidad generadas posteriormente. Luego, una iteración del procedimiento general se estructura como sigue:

1. $GSPV$ (acrónimo para Generador de Solución Parcial Válida) en la línea 3 genera la solución parcial para la iteración donde $\alpha_1, \alpha_2, \beta_1$ y β_2 son parámetros a definir y na_1 es un número aleatorio entre 0 y 1. Con el objetivo de prevenir computaciones redundantes, se registran en $Tabu\_Sets$ la solución parcial generadas y *status*, el método que posteriormente se utiliza en la línea 4 para resolver las instancia reducida originada.

2. Se reduce la instancia I_1 dando origen a I_2. Si $na_1 \geq \gamma$, S_2 se obtiene al resolver I_2 por CPLEX, en caso contrario, se utiliza MSBTS. Para ambos métodos se utiliza un tiempo máximo de ejecución de cinco segundos (líneas 5-9).

3. La solución para I_1 de esta iteración es $S_3 = S_1 \cup S_2$. Luego, en las las líneas 11 y 12 se actualiza $Tabu\_Sets$ y el conjunto B.

Una vez que el tiempo de ejecución máximo sea alcanzado, el algoritmo se detiene y la solución para la instancia es $S^* = Max(E)$ (línea 14).

Algoritmo 1 Esquema General

Entrada: $I_1, T_{MAX}, t_{max}, \alpha_1, \alpha_2, \beta_1, \beta_2, \gamma$

1: $F, l, B \leftarrow Preprocesamiento(I_1)$
2: **while** $Tiempo < T_{MAX}$ **do**
3: $\quad S_1, na_1, Tabu\_Sets \leftarrow GSPV(F, Tabu\_Sets, l, \alpha_1, \alpha_2, \beta_1, \beta_2)$
4: $\quad I_2 \leftarrow Reduccin(I_1, S_1, NF)$
5: $\quad$ **if** $na_1 \geq \gamma$ **then**
6: $\quad\quad S_2, status \leftarrow CPLEX(I_2, t_{max} = 5)$
7: $\quad$ **else**
8: $\quad\quad S_2, status \leftarrow MSBTS(I_2, t_{max} = 5)$
9: $\quad$ **end if**
10: $\quad S_3 = S_1 \cup S_2$
11: $\quad$ Actualizar $Tabu\_Sets$ con S_1
12: $\quad$ Actualizar B con S_3
13: **end while**
14: $S^* = Max(B)$

Salida: S^*

Fuente: Elaboración Propia.

3.2. Preprocesamiento

Como se menciona anteriormente, la generación de soluciones parciales es la base del algoritmo propuesto. Por tanto, es relevante que estas soluciones sean apropiadas para las etapas futuras. Por ejemplo, cuanta capacidad debe dejar disponible una solución parcial, o visto desde otra perspectiva, cuantos ítems deben componer una solución parcial son materias a resolver. Considerando esto, conocer la cantidad promedio de ítems l que tienen las soluciones factibles para esta instancia. Esto permite generar soluciones parciales de forma rápida, mediante el muestreo aleatorio de una cantidad de ítems lo suficientemente menor a l para que la factibilidad sea muy probable.

Además, generar un conjunto de ítems F en base a su frecuencia de aparición en soluciones factibles permite acotar el espacio de muestreo, creando soluciones parciales en las cuales solo ítems con alta frecuencia estén presentes. De forma similar, el conjunto NF está compuesto por todos aquellos ítems con frecuencia de aparición 0, por lo que se asume que son ítems que no estarán presentes en soluciones de alta calidad.

Así, el propósito de este preprocesamiento es obtener conjuntos y parámetros que se emplea posteriormente para la generación de soluciones parciales. El Algoritmo 2 muestra como funciona este segmento de la matheurística. Se recibe como entrada la instancia I_1 y se retornan los conjuntos F, NF y B, y el escalar l. Primero se inicia el vector de frecuencias λ en la línea 1, como un vector nulo n dimensional y el conjunto B en la línea 2. El primero registra las apariciones de cada ítem en las soluciones generadas en este segmento. Mientras que el segundo guarda las mejores soluciones encontradas hasta el momento. En la línea 3 se inicia el escalar l, que registra la cantidad de ítems en cada solución S, generándose n soluciones. Las soluciones son creadas en la línea 5 utilizando el método Greedy Randomized Initialization, presentado por Wei y Hao (2021b). En cada iteración de este método, se selecciona una fracción de los ítems con mayor proporción beneficio sobre peso, generando una lista restringida de candidatos. Para luego, seleccionar un ítem de forma aleatoria de esta lista e incluirlo en la solución. El preprocesamiento continua hasta alcanzar las n soluciones generadas. Finalmente, en la línea 9, l se aproxima al entero inferior y los conjunto F y NF son creados en la línea 10 a partir de ϕ, seleccionando los l ítems más frecuentes y los ítems

con frecuencia 0 respectivamente.

Algoritmo 2 Preprocesamiento

Entrada: I_1
1: Iniciar $\lambda = \vec{0_n}$
2: Iniciar $B = \emptyset$
3: $l = 0$
4: **while** i <n **do**
5: $S = Greedy\_Randomized\_Initialization(I_1)$
6: $l = l + largo(S)/n$
7: Actualizar B con S
8: **end while**
9: $l = int(l)$
10: $F, NF = Crear\_F\_NF(\lambda, l)$
Salida: F, NF, l, B

Fuente: Elaboración Propia.

3.3. Generación de soluciones parciales válidas

Existen tres mecanismos para generar la solución parcial S_1 y de forma aleatoria se decide en cada iteración cuál es la utilizada. A continuación, se presentan los mecanismos:

1. Con probabilidad de ocurrencia β_1, seleccionar de forma aleatoria $l \times \alpha_1$ ítems de F (línea 3 del Algoritmo 4).
2. Con probabilidad de ocurrencia β_2, seleccionar de forma aleatoria $l \times \alpha_2$ ítems de una solución en B (líneas 5 y 6 del Algoritmo 4).
3. Con probabilidad de ocurrencia $1 - \beta_1 - \beta_2$, seleccionar de forma aleatoria dos soluciones de B y luego intersecarlas (líneas 8 y 9 del Algoritmo 4).

De los tres mecanismos señalados, solo el primero puede generar soluciones infactibles: mientras que los otros dos muestrean soluciones factibles, el primero genera soluciones aleatorias.

Luego, se debe verificar la factibilidad de S_1 (línea 5 del Algoritmo 3), si esta es una solución factible para I_1 el algoritmo avanza a la etapa de reducción de instancia. Caso contrario, se actualiza la lista tabú registrando la infactibilidad de S_1.

Es posible que S_1 haya sido generada previamente. Si esta solución parcial apareció una vez anteriormente, entonces existen cuatro posibles estados para esta:

1. S_1 es una solución infactible (status $= -1$, líneas 11 y 12 del Algoritmo 3): en este caso una nueva solución parcial debe ser generada pues los métodos posteriores no revierten infactibilidades.

2. S_1 se complementó mediante CPLEX, deteniéndose el solver por criterio de optimalidad (status = 101, líneas 13 y 14 del Algoritmo 3): en esta situación S_1 debe ser generada nuevamente ya que al haber alcanzado el óptimo en la vecindad. Entonces, no es posible encontrar mejores soluciones que la ya obtenida a partir de la solución parcial utilizada.

3. S_1 se complementó mediante CPLEX, deteniéndose el solver por alcanzar el tiempo máximo de ejecución (status 131, líneas 15 a 18 del Algoritmo 3): al ocurrir esto S_1 aún puede dar origen a una solución superior. Sin embargo, MSBTS debe ser utilizado pues CPLEX es determinista y volver a ejecutarlo generaría la misma solución obtenida en la primera ocurrencia de S_1.

4. S_1 se complementó mediante MSBTS (status = 231, líneas 19 a 22 del Algoritmo 3): en este caso, de forma similar al anterior, el algoritmo MSBTS es determinista por lo que S_1 es completada usando CPLEX.

En el caso de la tercera y posteriores apariciones de la misma solución parcial, una nueva solución parcial es generada. Los Algoritmos 3 y 4 refleja lo mencionado anteriormente.

3.4. Reducción de instancia

Una vez sea obtenida satisfactoriamente la solución parcial S_1, el algoritmo propuesto continúa con la reducción de instancia. En este procedimiento son modificados los conjuntos M_1 y N_1, aquellos que forman la instancia I_1, eliminando de N_1 los ítems en S_1 (línea 1 del Algoritmo 5) y de M_1 los elementos en $E(S_1)$(Ecuación (1))(línea 5 del Algoritmo 5). Como se menciona en el Capítulo 2, cada miembro de N_1 es un subconjunto de M_1, así cuando M_1 sea modificado, cada ítem de N_1 también debe ser modificado, sustrayendo en este caso los elementos de $E(S_1)$(líneas 2 a 4 del Algoritmo 5). También, son eliminados de N_1 los ítems contenidos en el conjunto NF (línea 1 del Algoritmo 5). El Algoritmo 5 muestra

Algoritmo 3 GSPV

Entrada: $F, Tabu\_Sets, l, \alpha_1, \alpha_2, \beta_1, \beta_2$

$na_1 = Numero\_Aleatorio()$
while Verdadero **do**
    $S_1 = GSP(F, l, \alpha_1, \alpha_2, \beta_1, \beta_2)$
    if $S_1 \notin Tabu\_Sets$ **then**
        if $W(S_1) > c_1$ **then**
            $Tabu\_Sets[S_1] = -1$
        else
            Crear $Tabu\_Sets[S_1]$
            Break
        end if
    else if $-1 = Tabu\_Sets[S_1]$ **then**
        Pass
    else if $101 = Tabu\_Sets[S_1]$ **then**
        Pass
    else if $131 = Tabu\_Sets[S_1]$ **then**
        $na_1 = 0$
        $Tabu\_Sets[S_1] = -1$
        Break
    else
        $na_1 = 1$
        $Tabu\_Sets[S_1] = -1$
        Break
    end if
end while

Salida: $S_1, na_1, Tabu\_Sets$

Fuente: Elaboración Propia.

Algoritmo 4 GSP

Entrada: $F, l, \alpha_1, \alpha_2, \beta_1, \beta_2$

$na_2 \leftarrow Numero\_Aleatorio()$
if $na_2 \leq \beta_1$ **then**
    $S_1 = Muestreo\_Aleatorio(F, l * \alpha_1)$
else if $na_2 \leq \beta 1 + \beta_2$ **then**
    $S_1 = Muestreo\_Aleatorio(B, 1)$
    $S_1 = Muestreo\_Aleatorio(S, l * \alpha_2)$
else
    $S_a, S_b = Muestreo\_Aleatorio(B, 2)$
    $S_1 = S_a \cap S_b$
end if

Salida: S_1

Fuente: Elaboración Propia.

como funciona este procedimiento.

Algoritmo 5 Reducción de Instancia

Entrada: I_1, S_1

1: $N_2 = N_1 - S_1 - NF$
2: **for** $item \in N_2$ **do**
3: $\quad item = item - E(S_1)$
4: **end for**
5: $M_2 = M_1 - E(S_1)$
6: $c_2 = c_1 - W(S_1)$
7: $I_2 \leftarrow N_2, M_2, c_2$

Salida: I_2

Fuente: Elaboración Propia.

3.5. Solución de la instancia reducida

Al generar de forma satisfactoria I_2, el algoritmo propuesto continúa con la resolución de la nueva instancia generada. Se consideran dos procedimientos para encontrar una solución a I_2: CPLEX y MSBTS, definiéndose de forma aleatoria cuál se utiliza.

3.5.1. Solución usando CPLEX

De ser seleccionado CPLEX para resolver I_2, se utiliza el modelo matemático definido por las Ecuaciones (7) - (12). Se configura un tiempo máximo de ejecución de cinco segundos, mientras que el resto de las opciones de configuración disponibles mantienen sus valores por defecto. Una vez la ejecución se detiene se obtiene S_2 y el estatus con el que finalizó la ejecución, es decir si CPLEX se detuvo por criterio de optimalidad o tiempo.

3.5.2. Solución usando MSBTS

MSBTS es un algoritmo diseñado para resolver el SUKP, que realiza una búsqueda tabú, basada en soluciones, en el vecindario de una solución generada mediante un método greedy randomizado (Wei y Hao, 2021b). El esquema general de este procedimiento se puede observar en el Algoritmo 6. Se configura el parámetro t_{max} en cinco segundos, mientras que el resto se mantienen con su valores por defecto. Cuando este algoritmo se detiene se retorna S^*, obteniéndose $S_2 = S^*$.

Algoritmo 6 MSBTS

Entrada: Instancia I,Tiempo de interrupción t_{max}, Vecindarios N, Vectores Hash H_1, H_2, H_3, Largo de los Vectores Hash L, Funciones Hash h_1, h_2, h_3

1: $S^* \leftarrow \emptyset$ (Inicializar el Vector de Solución)
2: **while** $Tiempo \leq t_{max}$ **do**
3: $S \leftarrow Greedy\_Randomized\_Initialization(I)$
4: $S_b \leftarrow Solution\_Based\_Tabu\_search(S)$ (Guardar la mejor solución encontrada durante la búsqueda tabú)
5: **if** $F(S_b) \geq F(S^*)$ **then**
6: $S^* \leftarrow S_b$ (Actualizar la mejor solución encontrada)
7: **end if**
8: **end while**

Salida: La mejor solución encontrada S^*

Fuente: Wei y Hao (2021b)

3.6. Solución de la instancia inicial

Una vez S_2 es obtenida, la solución para la instancia I_1 en esta iteración es $S_3 = S_1 \cup S_2$ (línea 10 del Algoritmo 1). Luego, se debe actualizar la lista tabú, agregando S_1 y el método por el cual se obtuvo S_2 (*status*), para no caer en futuras computaciones redundantes (línea 11 del Algoritmo 1). La iteración finaliza verificando si $F(S_3)$ es lo suficientemente buena para integrar el conjunto de las mejores soluciones B, de cumplirse esto, este conjunto se actualiza (línea 12 del Algoritmo 1). El algoritmo sigue ejecutándose mientras el tiempo máximo de ejecución no sea alcanzado. Una vez este criterio sea alcanzado, la solución global para la instancia I_1 es la mejor solución en el conjunto B, $S^* = MAX(B)$ (línea 14 del Algoritmo 1).

3.7. Parámetros involucrados y calibración

Como se menciona en las secciones anteriores, el algoritmo utiliza cinco parámetros que deben ser proporcionados por el usuario y que afectan el desempeño y la calidad de los resultados obtenidos. Cuatro de ellos, $\alpha_1, \alpha_2, \beta_1$ y β_2 son utilizados en la generación de soluciones parciales, mientras que el quinto y último, γ, se emplea para definir que método se utiliza para resolver la instancia reducida, CPLEX o MSBTS.

Para calibrar los parámetros mencionados, se usa el paquete de R, IRACE (Iterated

Racing for Automatic Algorithm Configuration) en su versión 3.4.1, ejecutándose 5000 experimentos en siete instancias del problema, que en experimentos iniciales obtuvieron la mayores desviaciones estándar en los resultados. Se aplica un nivel de confianza del 95 %. La Tabla 1 muestra el rango de búsqueda y los mejores valores obtenidos para los cinco parámetros.

Tabla 1: Parámetros y Valores Respectivos

Parámetro	Rango	Resultados
α_1	$[0{,}50 - 0{,}9]$	0.6977
α_2	$[0{,}50 - 0{,}9]$	0.7785
β_1	$[0{,}00 - 1{,}0]$	0.0944
β_2	$[0{,}00 - 1{,}0]$	0.8361
γ	$[0{,}01 - 1{,}0]$	0.2863

Capítulo 4

Resultados y Análisis

En este capítulo son presentados los resultados computacionales. También, se presentan los resultados de los algoritmos del estado del arte que resuelven este mismo problema para facilitar la medición del desempeño de la matheurística propuesta. Además, la instancias utilizadas en los experimentos computacionales son presentadas y clasificadas. Se aborda también, la implementación del algoritmo propuesto.

4.1. Instancias de prueba

Los experimentos fueron ejecutados utilizando las 30 instancias, introducidas por He *et al.* (2018), las cuales poseen de 85 a 500 ítems y elementos. Las instancias se definen según su número de elementos m, número de ítems n, densidad de elementos a y la relación de capacidad b. La Ecuación (13) define el parámetro a como la suma de las cardinalidades de los ítems dividida por el número de ítems y elementos. Mientras que b se define mediante la Ecuación (14) como la capacidad de la instancia dividida por el peso de la solución que contiene todos los ítems.

$$a = \frac{\sum_{i \in N} CARD(i)}{n \times m} \tag{13}$$

$$b = \frac{C}{W(N)} \tag{14}$$

Las instancias son nombradas como "sukp n_m_a_b", por ejemplo la instancia

denominada sukp 85_100_0.10_0.75, posee 85 ítems, 100 elementos, $a = 0{,}10$ y $b = 0{,}75$. La Tabla 2 enumera las instancias utilizadas en los experimentos y las clasifica según la relación entre ítems y elementos.

Tabla 2: Enumeración y Clasificación de las Instancias.

$n > m$	$n = m$	$n < m$
sukp 100_85_0.1_0.75	sukp 100_100_0.1_0.75	sukp 85_100_0.1_0.75
sukp 100_85_0.15_0.85	sukp 100_100_0.15_0.85	sukp 85_100_0.15_0.85
sukp 200_185_0.1_0.75	sukp 200_200_0.1_0.75	sukp 185_200_0.1_0.75
sukp 200_185_0.15_0.85	sukp 200_200_0.15_0.85	sukp 185_200_0.15_0.85
sukp 300_285_0.1_0.75	sukp 300_300_0.1_0.75	sukp 285_300_0.1_0.75
sukp 300_285_0.15_0.85	sukp 300_300_0.15_0.85	sukp 285_300_0.15_0.85
sukp 400_385_0.1_0.75	sukp 400_400_0.1_0.75	sukp 385_400_0.1_0.75
sukp 400_385_0.15_0.85	sukp 400_400_0.15_0.85	sukp 385_400_0.15_0.85
sukp 500_485_0.1_0.75	sukp 500_500_0.1_0.75	sukp 485_500_0.1_0.75
sukp 500_485_0.15_0.85	sukp 500_500_0.15_0.85	sukp 485_500_0.15_0.85

4.2. Implementación

El algoritmo propuesto fue implementado utilizando Python 3.8 y todos los experimentos fueron ejecutados en un procesador Intel i5-6200U (2.3 GHz CPU y 8 GB de RAM) bajo un sistema operativo Windows 10 Pro. Además, los experimentos fueron ejecutados de manera secuencial. Por otra parte, MSBTS está implementado en C++ utilizando el compilador g++. La condición de detención, al igual que en los trabajos citados, es de 500 segundos para todas las instancias utilizadas, ejecutándose cada una 10 veces con semillas independientes.

4.3. Resultados computacionales

Luego de ejecutar 10 veces cada instancia con semillas independientes, teniendo un tiempo máximo de ejecución de 500 segundos como condición de término, se obtuvieron los resultados presentados en la Tabla 3. La primera columna señala el nombre de la instancia. Luego, la columna f_{max} informa el mejor valor encontrado para la función objetivo, $\overline{f}$ reporta la media de los valores objetivo, f_{DE} la desviación estándar de los valores objetivo. La última columna $\overline{T}$, indica la media del tiempo en segundos en el que el algoritmo alcanzó el valor objetivo máximo de la ejecución. Finalmente, la fila Media muestra el valor promedio para

cada columna.

Tabla 3: Resultados obtenidos por el algoritmo propuesto para las 30 instancias.

Instancia	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$
sukp 085_100_0.10_0.75	12045	12019.9	71.02	110.2
sukp 085_100_0.15_0.85	12369	12348	33.81	127.7
sukp 100_085_0.10_0.75	13283	13239.2	63.25	106.4
sukp 100_085_0.15_0.85	12479	12292.9	65.29	89.49
sukp 100_100_0.10_0.75	14044	14044	0	23.03
sukp 100_100_0.15_0.85	13508	13508	0	34.91
sukp 185_200_0.10_0.75	13647	13584.7	56.84	188.75
sukp 185_200_0.15_0.85	11298	11189.9	174.65	161.6
sukp 200_185_0.10_0.75	13521	13436.1	36.83	227.29
sukp 200_185_0.15_0.85	14215	14175	85.18	173.23
sukp 200_200_0.10_0.75	12522	12261	145.64	196.08
sukp 200_200_0.15_0.85	12317	12295.3	68.62	206.93
sukp 285_300_0.10_0.75	11568	11472	93.11	248.82
sukp 285_300_0.15_0.85	11763	11687	113.92	150.44
sukp 300_285_0.10_0.75	11477	11340.2	92.04	298.26
sukp 300_285_0.15_0.85	12607	12419.4	66.78	241
sukp 300_300_0.10_0.75	12713	12558.3	188.63	262.83
sukp 300_300_0.15_0.85	11585	11408.2	92.08	167.97
sukp 385_400_0.10_0.75	10424	10149	214.17	244.85
sukp 385_400_0.15_0.85	10506	10422	105.14	263.8
sukp 400_385_0.10_0.75	11484	11113	319.97	306.82
sukp 400_385_0.15_0.85	11209	11031.1	135.84	303.3
sukp 400_400_0.10_0.75	11513	11287.4	169.4	279.84
sukp 400_400_0.15_0.85	11325	11120.6	310.53	228.05
sukp 485_500_0.10_0.75	11125	1086.5	227.83	261.53
sukp 485_500_0.15_0.85	10220	10194.5	54.03	170.23
sukp 500_485_0.10_0.75	11729	11529.2	132.13	204.21
sukp 500_485_0.15_0.85	10217	10154.7	71.57	144.81
sukp 500_500_0.10_0.75	11249	10935	212.78	209.39
sukp 500_500_0.15_0.85	10381	10269.8	83.25	186.78
Media	11944.77	11814.19	116.15	193.96

Al analizar los resultados obtenidos y presentados en la Tabla 3, destaca que en promedio, luego de 200 segundos de ejecución, el algoritmo no pudo seguir mejorando la solución encontrada quedándose estancado en el óptimo local alcanzado. Sin embargo, al analizar los datos relacionados al tiempo de forma desagregada presentados en la Tabla A.8, un 11,7 % de las veces se necesitaron más de 400 segundos para alcanzar el valor máximo reportado en esa ejecución.

Si se consideran solo aquellas instancias que poseen 300 o más ítems ($n \geq 300$), esta proporción aumenta al 16,9 %. Por lo tanto, en las instancias más grandes o de mayor complejidad, el algoritmo propuesto sigue encontrando mejoras en las últimas porciones del tiempo máximo de ejecución, no quedándose estancado en óptimos locales en etapas tempranas.

4.4. Comparación con el estado del arte

Para medir el desempeño del algoritmo propuesto, sus resultados son comparados con otros cinco algoritmos existentes diseñados para resolver el SUKP:

1. DHJaya (Wu y He, 2020).
2. HBPSO/TS (Lin *et al.*, 2019).
3. I2PLS (Wei y Hao, 2019).
4. KBTS (Wei y Hao, 2021a).
5. MSBTS (Wei y Hao, 2021b).

Los resultados de estos algoritmos fueron extraídos de Wei y Hao (2021b) y fueron obtenidos luego de 100 ejecuciones con un tiempo máximo de 500 segundos para las instancias ya mencionadas. El computador utilizado en dicho estudio posee características distintas al utilizado en este trabajo, por tanto estos valores se utilizan solo a modo de referencia.

Una manera de poder realizar una comparación real de desempeño del algoritmo propuesto y los del estado del arte, es necesario ejecutar todos los experimentos computacionales bajo las mismas condiciones, utilizando el mismo lenguaje de programación y en el mismo computador. Implementar estos algoritmos para luego poder realizar los experimentos computacionales correspondientes se escapa del alcance de este trabajo.

La Tabla 4 muestra los valores medios obtenidos para las 30 instancias, para los algoritmos que se utilizan como comparación. La Tabla 5 presenta los resultados obtenidos por los algoritmos, considerando solamente instancias con $a = 0{,}15$ y $b = 0{,}85$, donde de las 30 instancias solo 15 cumplen el criterio. Las restantes 15 poseen valores $a = 0{,}10$ y $b = 0{,}75$.

En la Tabla 6, se presenta una comparación directa entre los seis algoritmos, utilizando como indicadores f_{max} y $\overline{f}$. La columna *Ganadas* hace referencia a cuantas veces un algoritmo obtuvo el valor máximo sin que ningún otro lo iguale. La columna *Empatadas* señala cuando un algoritmo comparte el valor máximo con uno o más algoritmos. Finalmente, la columna *Perdidas* muestra cuantas veces un algoritmo no puede conseguir el valor máximo alcanzado por el mejor/es competidor/es.

Tabla 4: Medias de los resultados obtenidos por los algoritmos considerando las 30 instancias.

Algoritmo	Medias			
	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$
DHJaya	11873.8	11748.1	61.6	156.3
HBPS0/TS	11967.5	11938.1	24.29	65.2
I2PLS	11973.6	11932.4	40.5	138.5
KBTS	11973.6	11968.6	7.87	78.16
MSBTS	11973.6	11953.7	18.9	96.7
Propuesto	11944.8	11814.2	116.2	194.0

Tabla 5: Medias de los resultados obtenidos por los algoritmo considerando solo instancias con $a = 0{,}15$ y $b = 0{,}85$.

Algoritmo	Medias			
	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$
DHJaya	11611.9	11444.7	81.0	159.5
HBPS0/TS	11734.4	11710.6	26.2	57.2
I2PLS	11737.3	11663.3	72.2	174.0
KBTS	11737.3	11728.4	13.7	102.3
MSBTS	11737.3	11698.5	34.9	146.3
Propuesto	11733.3	11634.4	97.4	176.7

Los resultados expuestos en la Tabla 6 muestran un rendimiento inferior del algoritmo propuesto en materias de valor objetivo máximo alcanzado y promedio del mismo, comparándolo con el estado del arte. No obstante, al incluir en el análisis los datos que se encuentran en la Tabla 3, queda en evidencia que en cuanto a f_{max} se refiere la diferencia con los algoritmos de comparación es pequeña. Además, se obtuvo un mejor desempeño que el algoritmo DHJaya. Enfocándose solamente en aquellas instancias que poseen $a = 0{,}15$ y $b = 0{,}85$, la brecha en el desempeño medido se reduce.

Tabla 6: Comparación resumida entre el algoritmo propuesto y los usados como comparación.

Algoritmo	f_{max}			$\overline{f}$		
	Ganadas	Empatadas	Perdidas	Ganadas	Empatadas	Perdidas
DHJaya	0	14	16	0	8	22
HBPSO/TS	0	28	2	0	18	12
I2PLS	0	30	0	0	10	20
KBTS	0	30	0	3	18	9
MSBTS	0	30	0	0	20	10
Propuesto	0	5	25	0	2	28

El desempeño promedio del método, medido mediante $\overline{f}$, es siempre inferior independiente de como se desagrupen las instancias, solo superando en rendimiento en esta categoría a DHJaya.

La estabilidad de la matheurística propuesta, medida mediante f_{DE}, resulta ser inferior en comparación a sus competidores, presentado una desviación estándar varias veces mayor que los algoritmos de mejor desempeño. Esta baja estabilidad influye en el desempeño medio del algoritmo explicando la brecha de rendimiento en el indicador $\overline{f}$.

Capítulo 5

Conclusión

5.1. Conclusiones

En este trabajo se presentó una matheurística diseñada para resolver el SUKP, empleando distintos mecanismo basados en muestreo aleatorio que producen soluciones parciales. Luego, se reducen las instancias y son resueltas mediante una metaheurística y/o CPLEX. El primero entrega de manera rápida buenas soluciones pero sin garantizar optimilidad, mientras que el segundo garantiza optimalidad pero la alcanza de manera muy lenta. También, se realiza una revisión de la literatura relacionada al SUKP, enfocándose en los métodos del estado del arte que resuelven este problema. Se presentan resultados computacionales ejecutando 30 instancias ampliamente usadas en la literatura.

El método propuesto posee un buen desempeño, cercano a los algoritmos del estado del arte existentes, diseñados exclusivamente para resolver el SUKP, destacando una brecha de rendimiento menor al 0.5 % en el mejor valor encontrado para la función objetivo. Por otra parte, todos los objetivos de este trabajo, general y específicos, fueron logrados satisfactoriamente.

5.2. Trabajo futuro

Varios puntos pueden ser atacados y así lograr un mejor desempeño del algoritmo. En primer lugar, implementarlo en un lenguaje de programación compilado, como C++, pues Python es un lenguaje interpretado y suelen ser más lentos que los

compilados. Luego, utilizar tablas hash para el manejo de las listas tabú, tal como lo emplea Wei y Hao (2021b), esto podría llevar a una gestión más eficiente de la memoria. También, gestionar de mejor manera los vecindarios visitados podría ayudar a descartar rápidamente áreas del espacio de búsqueda que no resulten prometedoras. Por último, realizar experimentos con instancias más grandes podría brindar información muy útil de como se comporta el algoritmo bajo un estrés mayor, facilitando la detección de supuestos erróneos.

Bibliografía

Arulselvan, A. (2014). A note on the set union knapsack problem. *Discrete Applied Mathematics*, 169:214–218.

Assi, M. y Haraty, R. A. (2018). A survey of the knapsack problem. En *2018 International Arab Conference on Information Technology (ACIT)*, pp. 1–6. IEEE.

Cho, M. (2019). The knapsack problem and its applications to the cargo loading problem. *Anal Appl Math*, 48.

Dahmani, I., Ferroum, M., y Hifi, M. (2021). An iterative rounding strategy-based algorithm for the set-union knapsack problem. *Soft Computing*, pp. 1–23.

Dahmani, I., Ferroum, M., y Hifi, M. (2022). Effect of backtracking strategy in population-based approach: The case of the set-union knapsack problem. *Cybernetics and Systems*, 53(1):168–185.

Durgut, R., Aydin, M. E., y Atli, I. (2021). Adaptive operator selection with reinforcement learning. *Information Sciences*, 581:773–790.

Engelbrecht, A. P. y Pampara, G. (2007). Binary differential evolution strategies. En *2007 IEEE congress on evolutionary computation*, pp. 1942–1947. IEEE.

Goldschmidt, O., Nehme, D., y Yu, G. (1994). Note: On the set-union knapsack problem. *Naval Research Logistics (NRL)*, 41(6):833–842.

He, Y., Xie, H., Wong, T.-L., y Wang, X. (2018). A novel binary artificial bee colony algorithm for the set-union knapsack problem. *Future Generation Computer Systems*, 78:77–86.

Karp, R. M. (1972). Reducibility among combinatorial problems. En *Complexity of computer computations*, pp. 85–103. Springer.

Kellerer, H., Pferschy, U., y Pisinger, D. (2004a). Multidimensional knapsack problems. En *Knapsack problems*, pp. 235–283. Springer, Berlín, Alemania.

Kellerer, H., Pferschy, U., y Pisinger, D. (2004b). Multiple knapsack probelm. En *Knapsack problems*, pp. 285–315. Springer, Berlín, Alemania.

Kellerer, H., Pferschy, U., y Pisinger, D. (2004c). The subset sum problem. En *Knapsack problems*, pp. 73–116. Springer, Berlín, Alemania.

Kellerer, H., Pferschy, U., y Pisinger, D. (2004d). The unbounded knapsack probelm. En *Knapsack problems*, pp. 185–234. Springer, Berlín, Alemania.

Kiran, M. S. (2015). The continuous artificial bee colony algorithm for binary optimization. *Applied Soft Computing*, 33:15–23.

Lin, G., Guan, J., Li, Z., y Feng, H. (2019). A hybrid binary particle swarm optimization with tabu search for the set-union knapsack problem. *Expert Systems with Applications*, 135:201–211.

Navathe, S., Ceri, S., Wiederhold, G., y Dou, J. (1984). Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems (TODS)*, 9(4):680–710.

Schmitt, L. M. (2001). Theory of genetic algorithms. *Theoretical Computer Science*, 259(1):1–61.

Tu, M. y Xiao, L. (2016). System resilience enhancement through modularization for large scale cyber systems. En *2016 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pp. 1–6. IEEE.

Wei, Z. y Hao, J.-K. (2019). Iterated two-phase local search for the set-union knapsack problem. *Future Generation Computer Systems*, 101:1005–1017.

Wei, Z. y Hao, J.-K. (2021a). Kernel based tabu search for the set-union knapsack problem. *Expert Systems with Applications*, 165:113802.

Wei, Z. y Hao, J.-K. (2021b). Multistart solution-based tabu search for the set-union knapsack problem. *Applied Soft Computing*, 105:107260.

Wu, C., Gao, X., Liu, X., y Sun, B. (2022). Self-adjusting optimization algorithm for solving the setunion knapsack problem. *arXiv preprint arXiv:2202.05698*.

Wu, C. y He, Y. (2020). Solving the set-union knapsack problem by a novel hybrid jaya algorithm. *Soft Computing*, 24(3):1883–1902.

Apéndice A

Resultados desagregados del algoritmo propuesto

En este apéndice se presentan de forma desagregada todos los resultados obtenidos durante la ejecución de los experimentos computacionales. En la Tabla A.7 se observan los resultados de la función objetivo obtenidos para cada instancia en cada una de las 10 ejecuciones realizadas. De la misma forma, la Tabla A.8 presenta los tiempos asociados a dichos valores de la función objetivo, en especifico, en que segundo de la ejecución dicho valor fue alcanzado.

Tabla A.7: Resultados de la función objetivo obtenidos para cada ejecución e instancias

Instancias	Ejecución									
	1	2	3	4	5	6	7	8	9	10
sukp 085_100_0.10_0.75	12045	12045	12045	11819	12045	12045	12045	12045	12020	12045
sukp 085_100_0.15_0.85	12369	12369	12369	12299	12299	12369	12299	12369	12369	12369
sukp 100_085_0.10_0.75	13251	13251	13167	13251	13089	13283	13251	13283	13283	13283
sukp 100_085_0.15_0.85	12272	12272	12272	12272	12274	12272	12272	12479	12272	12272
sukp 100_100_0.10_0.75	14044	14044	14044	14044	14044	14044	14044	14044	14044	14044
sukp 100_100_0.15_0.85	13508	13508	13508	13508	13508	13508	13508	13508	13508	13508
sukp 185_200_0.10_0.75	13622	13556	13609	13647	13622	13647	13497	13509	13529	13609
sukp 185_200_0.15_0.85	11298	10920	11298	10920	11298	10973	11298	11298	11298	11298
sukp 200_185_0.10_0.75	13441	13402	13441	13402	13449	13449	13449	13405	13521	13402
sukp 200_185_0.15_0.85	14215	14215	14215	14215	13987	14215	14215	14044	14215	14215
sukp 200_200_0.10_0.75	12384	12190	12253	12522	12138	12253	12307	11973	12295	12295
sukp 200_200_0.15_0.85	12317	12317	12317	12100	12317	12317	12317	12317	12317	12317
sukp 285_300_0.10_0.75	11538	11538	11383	11359	11538	11538	11568	11399	11327	11538
sukp 285_300_0.15_0.85	11763	11763	11763	11507	11763	11496	11763	11714	11763	11575
sukp 300_285_0.10_0.75	11246	11239	11381	11477	11457	11257	11379	11315	11408	11243
sukp 300_285_0.15_0.85	12401	12401	12401	12607	12411	12380	12401	12411	12380	12401
sukp 300_300_0.10_0.75	12713	12585	12386	12646	12167	12695	12695	12646	12695	12355
sukp 300_300_0.15_0.85	11585	11410	11425	11425	11222	11425	11425	11425	11425	11315
sukp 385_400_0.10_0.75	10424	10397	9880	10221	10414	10146	9834	10011	10065	10098
sukp 385_400_0.15_0.85	10302	10490	10302	10506	10302	10506	10506	10506	10506	10294
sukp 400_385_0.10_0.75	11318	11088	10663	10566	11189	11261	11484	11484	11216	10861
sukp 400_385_0.15_0.85	11010	10833	11010	11209	10984	11209	10957	10880	11209	11010
sukp 400_400_0.10_0.75	11500	11028	11131	11333	11513	11317	11311	11134	11450	11157
sukp 400_400_0.15_0.85	11325	11325	11168	11325	10915	11325	11325	11325	10554	10619
sukp 485_500_0.10_0.75	10684	10610	10649	10549	11097	10972	10941	11103	11035	11125
sukp 485_500_0.15_0.85	10220	10220	10220	10220	10220	10220	10104	10220	10220	10081
sukp 500_485_0.10_0.75	11722	11598	11338	11729	11472	11598	11685	11596	11462	11722
sukp 500_485_0.15_0.85	10194	10061	10194	10194	10194	10093	10194	10012	10194	10217
sukp 500_500_0.10_0.75	11099	10860	10437	10960	10923	11249	11083	10911	10868	10960
sukp 500_500_0.15_0.85	10186	10381	10214	10214	10209	10381	10381	10309	10214	10209

Tabla A.8: Resultados del tiempo en que el algoritmo alcanzo el valor máximo de la función objetivo por ejecución e instancia

Instancias	Ejecución									
	1	2	3	4	5	6	7	8	9	10
sukp 085_100_0.10_0.75	215.06	22.17	332.34	5.47	61.7	21.19	54.24	108.25	115.15	167.21
sukp 085_100_0.15_0.85	154.4	0.14	265.71	0.05	5.76	240.93	104.87	41.28	52.37	411.89
sukp 100_085_0.10_0.75	297.37	38.37	17.03	179.94	5.73	38.84	181.56	28.13	132.19	145.02
sukp 100_085_0.15_0.85	101.62	79.48	101.89	95.97	140.12	25.74	91.66	6.82	148.77	102.82
sukp 100_100_0.10_0.75	0.07	16.12	53.41	5.38	15.77	31.95	16.56	0.35	73.92	16.73
sukp 100_100_0.15_0.85	5.73	6.07	108.85	45.08	47.36	19.34	19.86	27.44	68.81	0.55
sukp 185_200_0.10_0.75	25.25	396.08	318.4	468.15	16.18	148.46	395.82	28.77	79.74	10.59
sukp 185_200_0.15_0.85	31.07	202.79	70.88	45.26	48.21	218.23	35.64	141.13	450.47	372.45
sukp 200_185_0.10_0.75	229.36	273.44	192.71	174.54	170.48	28.98	323.74	385.09	124.51	370.01
sukp 200_185_0.15_0.85	123.14	174.96	29.41	88.11	328.64	345.03	27.37	125.75	466.61	23.2
sukp 200_200_0.10_0.75	211	107.18	176.34	175.37	214.26	99.32	230.14	235.24	322.67	189.26
sukp 200_200_0.15_0.85	480.81	102.92	80.96	345.87	228.04	41.23	390.49	224.81	51.62	122.47
sukp 285_300_0.10_0.75	311.07	197.19	44.89	109.67	429.58	286.21	146.82	479.27	69.74	413.67
sukp 285_300_0.15_0.85	21.98	369.83	91.06	219.07	65.39	207.69	96.9	34.52	342.74	55.22
sukp 300_285_0.10_0.75	136.86	254.68	323.47	379.8	130.44	33.77	498.88	471.04	387.01	366.57
sukp 300_285_0.15_0.85	207.12	119.92	178.33	87.85	424.95	124.32	335.56	358.23	355.56	218.07
sukp 300_300_0.10_0.75	203.96	434.62	73.38	399.34	302.88	70.78	325.99	473.17	309.97	34.2
sukp 300_300_0.15_0.85	29.61	455.4	230.05	68.81	306.16	222.9	5.4	159.23	28.49	173.6
sukp 385_400_0.10_0.75	399.5	419.78	471.74	5.57	245.53	19.81	5.45	418.23	388.45	74.4
sukp 385_400_0.15_0.85	320.8	476.49	310.31	115.66	286.78	123.19	11.61	482.31	168.57	342.23
sukp 400_385_0.10_0.75	382.17	472.89	24.29	431.93	478.49	411.1	339.7	93.88	288.82	144.92
sukp 400_385_0.15_0.85	400.23	440.76	390.03	129.61	272	191.79	426.57	6.2	428.72	347.55
sukp 400_400_0.10_0.75	27.8	378.47	318.55	330.69	327.17	262.12	420.33	245.45	297.92	189.83
sukp 400_400_0.15_0.85	131.53	375.36	134.18	283.94	228.36	122.19	446.45	296.72	99.72	162.04
sukp 485_500_0.10_0.75	61.43	435.5	160.53	165.53	355.73	346.03	306.79	440.51	217.6	125.62
sukp 485_500_0.15_0.85	68.18	151.71	159.15	160.2	210.36	241.64	311.58	135.39	38.41	225.64
sukp 500_485_0.10_0.75	210.3	128.76	193.33	208.66	332.6	61.93	161.26	256.1	483.49	5.65
sukp 500_485_0.15_0.85	25.27	315.13	163.15	11.96	15.85	309.16	104.31	279.72	98.9	124.6
sukp 500_500_0.10_0.75	35.73	166.06	274.35	218.07	498.35	5.56	266.46	23.27	204.3	401.76
sukp 500_500_0.15_0.85	421.62	28.04	102.05	275.37	156.86	20.86	287.54	191.9	60.8	322.7

Resultados de algoritmos previos

En este apéndice se presentan los resultados que otros algoritmos obtuvieron al enfrentarse a las mismas instancias que el algoritmo propuesto. Estos datos fueron extraídos de Wei y Hao (2021b), en donde se encuentran todos los detalles de como fueron obtenidos.

Tabla A.9: Resultados de los algoritmos utilizados de forma comparativa.

Fuente: Wei y Hao (2021b).

Instancia	Algoritmos																			
	DHJaya				HBPSO/TS				I2PLS				KBTS				MSBTS			
	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$	f_{max}	$\overline{f}$	f_{DE}	$\overline{T}$
085_100_0.10_0.75	12045	12045	0	17.199	12045	12045	0	0.056	12045	12045	0	2.798	12045	12045	0	0.075	12045	12045	0	3.117
085_100_0.15_0.85	12369	12369	0	0.342	12369	12369	0	0.088	12369	12315.53	62.6	17.47	12369	12369	0	10.175	12369	12369	0	26.24
100_085_0.10_0.75	13283	13283	0	9.477	13283	13283	0	0.098	13283	13283	0	3.094	13283	13283	0	4.082	13283	13283	0	12.77
100_085_0.15_0.85	12479	12479	0	24.414	12479	12403.15	98.97	101.122	12479	12335.13	98.78	103.757	12479	12479	0	42.992	12479	12413.78	79.79	184.323
100_100_0.10_0.75	14044	14044	0	1.374	14044	14044	0	0.518	14044	14044	0	38.245	14044	14044	0	0.023	14044	14044	0	6.639
100_100_0.15_0.85	13508	13508	0	1.572	13508	13508	0	2.923	13508	13451.5	126.49	70.587	13508	13508	0	33.403	13508	13508	0	55.103
185_200_0.10_0.75	13696	13667.63	26.56	244.205	13696	13696	0	0.489	13696	13695.6	3.68	124.136	13696	13696	0	5.851	13696	13696	0	7.089
185_200_0.15_0.85	11298	11298	0	38.439	11298	11298	0	0.486	11298	11276.17	83.78	139.865	11298	11298	0	6.373	11298	11298	0	30.689
200_185_0.10_0.75	13521	13498.22	26.1	258.213	13521	13521	0	0.49	13521	13521	0	71.984	13521	13521	0	6.988	13521	13521	0	22.528
200_185_0.15_0.85	14215	14215	0	83.129	14215	14177.38	70.84	72.041	14215	14031.28	131.46	180.809	14215	14209.87	29.17	107.407	14215	13946.15	153.67	258.541
200_200_0.10_0.75	12522	12480.62	65.05	207.667	12522	12522	0	0.8125	12522	12522	0	54.78	12522	12522	0	48.206	12522	12518.28	21.15	70.411
200_200_0.15_0.85	12317	12217.81	93.361	229.824	12317	12317	0	0.95	12317	12280.07	57.77	238.348	12317	12317	0	72.495	12317	12316.21	7.86	92.155
285_300_0.10_0.75	11568	11563.8	10.41	203.874	11568	11568	0	13.63	11568	11568	0	25.128	11568	11568	0	30.618	11568	11567.7	2.99	17.706
285_300_0.15_0.85	11714	11436.93	101.85	463.466	11802	11802	0	2.135	11802	11790.43	27.51	206.422	11802	11799.27	9.95	168.904	11802	11798.88	10.58	186.685
300_285_0.10_0.75	11385	11167.77	129.98	174.335	11563	11563	0	38.355	11563	11562.02	3.94	181.248	11563	11563	0	28.841	11563	11563	0	37.877
300_285_0.15_0.85	12402	12248.42	22.12	316.767	12607	12607	0	24.967	12607	12364.55	83.03	240.333	12607	12536.02	87.51	235.45	12607	12430.51	73.86	216.465
300_300_0.10_0.75	12736	12676.78	35.2	241.774	12817	12806.44	15.39	29.074	12817	12817	0	66.403	12817	12817	0	74.247	12817	12813.7	9.9	165.618
300_300_0.15_0.85	11425	11260.25	103.95	152.329	11585	11585	0	5.985	11585	11512.18	73.15	220.1	11585	11584.17	8.26	141.464	11585	11585	0	156.333
385_400_0.10_0.75	10483	10287.36	80.61	53.459	10600	10552.73	74.68	100.155	10600	10536.53	56.08	234.475	10600	10600	0	73.087	10600	10599.7	1.71	150.505
385_400_0.15_0.85	10302	10184.09	138	230.077	10506	10472.4	67.2	168.87	10506	10502.64	23.52	129.505	10506	10506	0	58.24	10506	10504.23	16.08	133.34
400_385_0.10_0.75	11484	11325.88	38.65	229.37	11484	11484	0	10.87	11484	11484	0	31.801	11484	11484	0	0.296	11484	11484	0	7.643
400_385_0.15_0.85	10710	10293.96	173.85	241.068	11209	11209	0	16.478	11209	11157.26	87.29	141.525	11209	11209	0	72.02	11209	11209	0	46.8
400_400_0.10_0.75	11569	11301.56	74.88	322.143	11665	11484.2	72.95	45.025	11665	11665	0	18.733	11665	11665	0	64.126	11665	11657.08	10.56	90.423
400_400_0.15_0.85	10927	10721.45	221.38	77.037	11325	11325	0	5.902	11325	11325	0	76	11325	11325	0	17.591	11325	11309.2	112.46	125.95
485_500_0.10_0.75	11036	10883.19	48.58	66.029	11321	11142.27	62.51	223.387	11321	11306.47	36	207.118	11321	11318.81	10.95	121.494	11321	11321	0	54.178
485_500_0.15_0.85	10104	9665.7	142.57	49.438	10220	10208.96	3.26	143.999	10220	10179.45	46.97	238.63	10220	10219.76	1.68	118.564	10220	10219.04	3.26	123.052
500_485_0.10_0.75	11722	11675.51	55.53	226.604	11771	11746.19	57.98	293.514	11771	11729.76	6.59	349.545	11771	11755.47	19.74	206.199	11771	11771	0	31.171
500_485_0.15_0.85	10194	9703.56	114.852	383.021	10194	10163.76	82.11	92.121	10238	10133.94	94.72	369.375	10238	10202.9	16.25	293.14	10238	10205.62	16.33	389.536
500_500_0.10_0.75	10943	10871.22	39.93	41.383	11109	11026.24	51.62	340.958	11249	11243.4	27.43	134.186	11249	11248.96	0.4	146.04	11249	11249	0	29.905
500_500_0.15_0.85	10214	10069.33	103.33	101.926	10381	10213.25	71.3	220.328	10381	10293.89	85.53	237.894	10381	10362.63	52.25	156.331	10381	10365.52	49.41	169.084
Media	11873.83	11748.06	61.55	156.33	11967.46	11938.09	24.29	65.19	11973.6	11932.39	40.54	138.47	11973.6	11968.56	7.87	78.15	11973.6	11953.72	18.98	96.72