UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

Una metaheurística para el problema de conjunto dominante con capacidad mínima

POR

Sebastián Ignacio Barría Barría

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)

Resumen

La presente memoria de título tiene como objetivo presentar el problema de conjunto dominante con capacidad mínima (CAPMDSP), de manera más específica, se trata de una variante del problema de conjunto dominante. CAPMDSP integra una restricción adicional que establece una capacidad máxima que puede dominar cada nodo. Con tal de resolver el CAPMDSP se plantea un algoritmo de múltiples inicios de búsqueda local iterada, el cual hace uso de un algoritmo del estado del arte llamado H-MUEC. H-MUEC es modificado y es utilizado para generar la solución inicial y es aplicado como una búsqueda local. Los resultados generan soluciones factibles pero que difieren respectos a los encontrados por el modelo de programación entera mixta, y los tiempos son mayores a la literatura, debido a la diferencia del lenguaje de programación utilizado.

Abstract

The objective of the current thesis is to present the minimum capacity dominating set problem (CAPMDSP), more specifically, it is a variant of the dominating set problem. CAPMDSP integrates an additional constraint that establishes a maximum capacity that each node can dominate. In order to solve CAPMDSP, a multi-start iterated local search algorithm is proposed, which makes use of a state-of-the-art algorithm called H-MUEC. The H-MUEC is modified and is used to generate the initial solution and is applied as a local search. The results generate feasible solutions but differ from those found by the mixed integer programming model, and the times are longer than the literature, due to the difference in the programming language used.

Tabla de contenido

Lista de tablas

Lista de Figuras

Lista de Algoritmos

1. Introducción

En el presente capítulo se introduce el problema de conjunto dominante y sus variantes. Además, se presenta el objetivo general y los objetivos específicos de la investigación. Finalmente, se presenta la estructura del documento.

1.1 Introducción

La teoría de grafos es una rama importante de la matemática, ciencias de la computación y clave en muchos problemas de investigación de operaciones. Donde se estudian distintos problemas desafiantes, dentro de los cuales es posible encontrar el problema de conjuntos dominantes (MDSP, por sus siglas en inglés, minimun dominating set problem). El MDSP tiene como objetivo encontrar un conjunto dominante de cardinalidad mínima. Dentro de las variantes de este problema es posible encontrar el problema de conjunto dominante con capacidad mínima (CAPMDSP, por sus siglas en inglés, mínimum capacitated dominating set problem). CAPMDSP consiste en encontrar un conjunto dominante de cardinalidad mínima y tiene una restricción adicional que consiste en que la cantidad de nodos dominados no deben exceder la capacidad de los respectivos nodos dominantes. En el caso de que la capacidad de cada nodo en el grafo es mayor o igual a su grado, el CAPMDSP es igual al problema de MDSP. Adicionalmente, es necesario tener en cuenta que Garey y Johnson (1979) explican que el MDSP es un problema *NP-hard.* Por tanto, encontrar el conjunto dominador resulta ser muy difícil computacionalmente hablando y a día de hoy no se conoce un algoritmo que en tiempo polinomial pueda determinar si el tamaño de un conjunto dominante es menor o igual que cierto número *k.* Aun así, existen algoritmos de aproximación eficientes e incluso exactos para ciertas clases de grafos. Por lo que, CAPMDS al ser una variante del problema original, también es *NP-hard.*

CAPMDSP tiene variadas aplicaciones relacionadas a entornos de trabajo con recursos limitados. A modo de ejemplo tenemos: la instalación de un mínimo de número de cámaras, sensores inalámbricos para vigilar el mayor número de objetos posibles dentro de una zona determinada, la recuperación de información en documentos digitales, el enrutamiento en redes inalámbrica mediante el cálculo de un camino para el tráfico de datos dentro de redes, entre otras aplicaciones (Díez Corral, 2020).

1. 2 Objetivo general

Implementar una metaheurística para resolver el problema de conjunto dominante con capacidad mínima.

1.3 Objetivos específicos

- Desarrollar una revisión de la literatura con relación al estado del arte de los métodos propuestos para resolver el CAPMDSP.
- Diseñar e implementar una metaheurística para resolver el CAPMDSP.
- Diseñar y ejecutar los experimentos computacionales.
- Analizar los resultados y compararlos con la literatura.

1.4 Estructura del documento

El documento se organiza de la siguiente manera. En el Capítulo 2 se realiza una descripción detallada del CAPMDSP junto a la formulación del modelo matemático y una revisión de literatura. Luego, en el Capítulo 3 se presenta la metaheurística, explicando la estructura general de esta y sus principales fases. El Capítulo 4 entrega los experimentos computacionales realizados, detallando su implementación, describiendo las instancias utilizadas y la calibración de los parámetros. Además, se presenta un análisis de los resultados obtenidos. Finalmente, en el Capítulo 5, se presentan las conclusiones obtenidas.

2. Problema de conjunto dominante con capacidad mínima.

En este capítulo se presenta una descripción del problema de conjunto dominante con capacidad mínima, se formula el modelo matemático como un problema de programación entera mixta. Finalmente, se hace una revisión de literatura para el CAPMDSP con sus principales metodologías de resolución.

2.1 Descripción

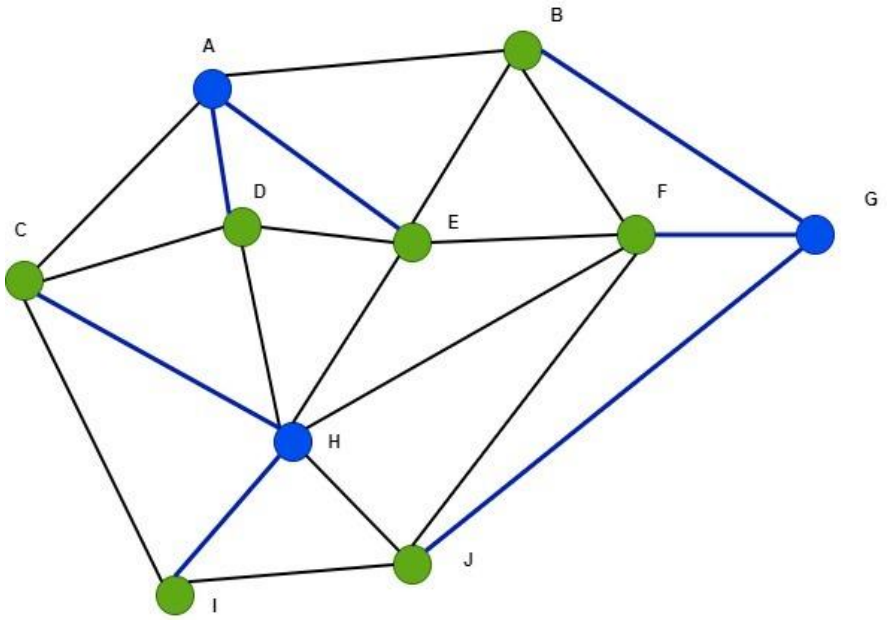
El CAPMDSP tiene como objetivo encontrar un subconjunto de vértices D mínimo tal que cada vértice del grafo es vecino a algún vértice de *D*, es decir, dado un grafo no dirigido $G = (V, E)$, donde $V = \{v_1, v_2, .., v_n\}$ denota el conjunto de vértices o nodos y *E* denota el conjunto de aristas. Entonces, se debe encontrar un subconjunto de vértices $D \subseteq V$ los cuales se denominan dominadores o nodos dominantes. El resto de los nodos de V, que no pertenece a *D*, son llamados dominados o no dominantes. Se debe tener en cuenta que un nodo dominante $(u \in V)$ domina a un nodo no dominante $v\,(v \notin D)$, y se dice que *v* está dominado por *u* si existe una arista entre u y v, es decir, $(u, v) \in E$. Además, se debe tener en cuenta que es posible crear un conjunto de dominados, tal que cada nodo que no está en *D*, está dominado por al menos un nodo de *D*.

A diferencia del MDSP original, el CAPMDSP integra una restricción adicional, la cual consiste en un límite superior en el número de nodos no dominantes que un nodo dominante puede dominar, es decir, sirve como cota para restringir el grado del nodo. Así, una instancia del problema está dada por una tupla *(G, Cap)*, que consta del grafo no dirigido *G = (V, E)* y una capacidad *(Cap)* que representa el número máximo de vértices adyacentes que los nodos pueden dominar.

La Figura 1 proporciona un ejemplo ilustrativo del CAPMDSP con 10 vértices y capacidad de tres, en el cual los vértices *A, H* y *G* actúan como nodos dominantes. Por ejemplo, el nodo *G* domina a los nodos *B, F* y *J*. Adicionalmente, es posible notar que la capacidad sirve como cota superior ante nodos con grado superior a esta. Análogamente, en la Figura 2 se proporciona el mismo caso, pero sin restricción de capacidad, es decir, como el MDSP, produciéndose una cardinalidad inferior respecto al CAPMDSP, obteniendo solo

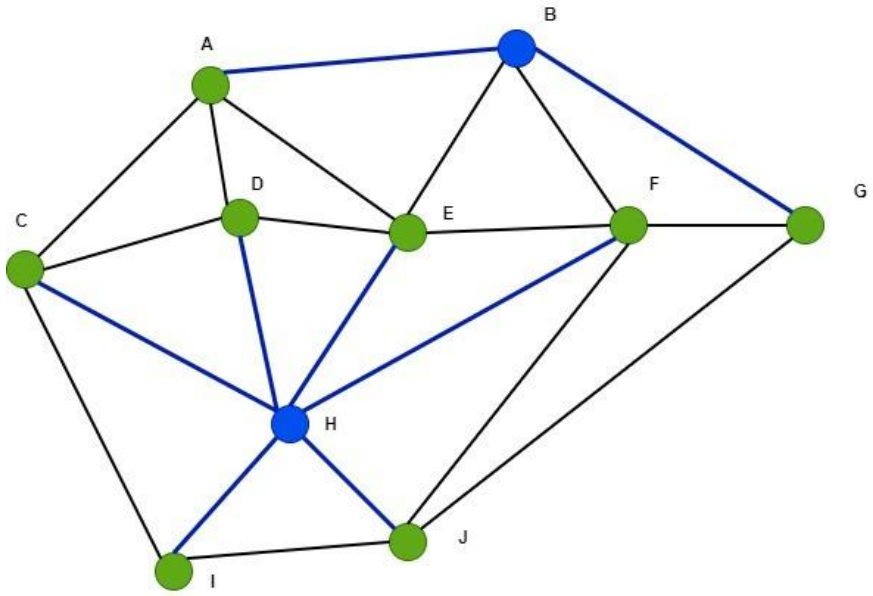
dos nodos dominantes (B y H) y donde queda en evidencia el efecto que produce el añadir la restricción de capacidad.

Figura 1: Ejemplo CAPMDSP.



Fuente: Elaboración Propia.

Figura 2: Ejemplo MDSP.



Fuente: Elaboración Propia.

2.2 Modelo matemático

El CAPMDSP tiene varias formulaciones de programación matemática. Una de las más eficientes es el modelo de programación lineal entera mixta (PLEM) propuesto por Nakkala et al. (2021), el cual se presenta a continuación. Es necesario mencionar que esta es una versión mejorada de la propuesta de Pinacho-Davinson et al. (2019).

Variables de decisión:

$$x_i: \begin{cases} 1\text{:Si el nodo } i \in V \text{es un nodo dominante.} \\ 0\text{: En cualquier otro caso.} \end{cases}$$

$$y_{ij}: \begin{cases} 1\text{:Si el nodo } v_i \in V \text{ domina al nodo } v_j \in V. \\ 0\text{: En cualquier otro caso.} \end{cases}$$

$$y_{ji}: \begin{cases} 1\text{:Si el nodo } v_j \in V \text{ domina al nodo } v_i \in V. \\ 0\text{: En cualquier otro caso.} \end{cases}$$

Modelo de programación lineal entera:

$$\text{Minimizar} \sum_{v_i \in V} x_i \tag{1}$$

Sujeto a:

$$\sum_{v_j \in N(v_i)} y_{ji} = 1 - x_i, \qquad \forall v_i \in V \tag{2}$$

$$\sum_{v_j \in N(v_i)} y_{ji} \leq \min\big(C(v_i), d_G(v_i)\big)\, x_i, \qquad \forall v_i \in V \tag{3}$$

$$y_{ij} \leq x_i, \qquad \forall v_i \in V\,, v_j \in N(v_i) \tag{4}$$

$$x_j \leq x_i, \qquad \forall\, v_i, v_j \in V \times V \tag{5}$$

$$\sum_{v_j \in V} \left\lceil \frac{w(v_i)}{q} \right\rceil x_i \geq \left\lceil \frac{|V|}{q} \right\rceil, \qquad \forall q \in W \tag{6}$$

$$x_j \in \{0,1\}, \qquad \forall v_i \in V \tag{7}$$

$$y_{ij}, y_{ji} \in [0,1], \qquad \forall (i,j) \in E \tag{8}$$

La ecuación (1) representa la función objetivo del CAPMDSP que consiste en minimizar el número de vértices dominadores. Las restricciones (2) indican que cada vértice es parte del conjunto dominante o está cubierto por un vértice perteneciente al conjunto dominante. El conjunto de restricciones (3) impone la restricción sobre el número máximo de vértices que un vértice puede dominar. Las restricciones (4) aplican la restricción de que y_{ij} puede ser distinto de cero solo cuando v_i es un vértice dominante. Las restricciones (5) hacen cumplir la restricción de vértices fuertes, es decir, se busca que el vértice v_i tenga mayor capacidad efectiva que v_j, en términos generales, se busca cumplir que $\min\big(C(v_i), d_G(v_i)\big) > \min\left(C(v_j), d_G(v_j)\right)$. Las restricciones (6) imponen un límite inferior de la cardinalidad, el cual considera $w(v_i) = \min\big(C(v_i), d_G(v_i)\big) + 1 > 0$, siendo el número máximo de vértices que puede abarcar el vértice v_i. El conjunto de restricciones (7) restringe los valores de cada $x_i \forall v_i \in V$ a 0 o 1. Finalmente, las restricciones (8) limitan cada y_{ij} y y_{ji} $\forall (i,j) \in E$ al intervalo [0,1], teniendo en cuenta que en la solución la variable *y*, tiene comportamiento entero, pero al considerarla como continua, el solucionador disminuye el esfuerzo computacional.

2.3 Revisión de literatura

El CAPMDSP es una de las variantes más populares y estudiadas del MDSP. Por tanto, hay una gran cantidad de publicaciones con diversos métodos para su resolución. A continuación, se presentan los trabajos más relevantes.

Kao y Chen (2010) modelan escenarios de asignación de necesidades de servicio, siendo una generalización del CAPMDSP, donde dado un grafo con tres parámetros: coste, capacidad y demanda. Buscan encontrar la asignación de demandas a los vértices de menor coste, tal que la demanda se satisfaga, sujeta a la restricción de capacidad de cada vértice que presta el servicio. En términos de aproximación de tiempos polinomiales, muestran aproximaciones logarítmicas en algunos modelos de asignación de demanda para grafos generales. Kuhn y Moscibroda (2010) proponen un algoritmo de aproximación distribuida para el CAPMDSP con capacidad uniforme en una familia de grafos llamada UDG[1], basado

[1] UDG: Unit disk graphs.

en el conjunto de máximo conjunto independiente. Cygan et al. (2011) desarrolla un algoritmo para resolver el CAPMDSP en $O(1.89^{|V|})$, tiempo basado en el concepto de coincidencia máxima. También presenta un esquema de aproximación exponencial para este tipo de problemas.

Potluri y Singh (2012), lleva a cabo una comparación de rendimiento entre tres heurísticas codiciosas: una heurística de menor grado de cobertura máxima (MGRA-CM), una heurística de subdivisión de clúster (SUBD-C), y una heurística de relajación mínima independiente (SD-RELAJ). Estas heurísticas fueron testeadas haciendo uso de instancias de UDG y grafos generales con capacidad uniforme y variable. Dentro de las principales conclusiones está el hecho que SUBD-C posee un buen comportamiento frente a UDGs, con baja capacidad uniforme y en especial en aquellos casos con un alto grado de conectividad. Por otro lado, tiene un pobre rendimiento para grafos generales por lo que no se recomienda su uso para estas instancias. Respecto a la heurística SD-RELAJ, tiene un comportamiento mejor en comparación a la anterior. Por último, MGRA-CM es la mejor de las variantes estudiadas, teniendo resultados favorables tanto para las instancias UDG y los grafos generales con capacidad uniforme y variable. Es necesario rescatar el hecho de que esta última heurística selecciona en cada paso el vértice que maximiza el mínimo entra la capacidad del vértice y el número de vecinos.

Potluri y Singh (2013) toman como base el estudio anteriormente mencionado para diseñar dos enfoques metaheurísticos híbridos: uno basado en la optimización de las colonias de hormigas (OCH) y el algoritmo genético (AG) para el CAPMDSP. Donde OCH y AG se hibridan con una heurística de minimización que elimina nodos redundantes. Los resultados computacionales, muestran que cuando las capacidades de los nodos son pequeñas el algoritmo funciona eficientemente. Sin embargo, el rendimiento disminuye a la hora de que la capacidad aumenta. Posteriormente, Liedloff et al. (2014) retoma los estudios relacionados a la complejidad temporal, realizando una mejora $O\left(1.8463^{|V|}\right)$ mediante el uso de programación dinámica sobre subconjuntos.

Becker (2016) desarrolla un esquema de aproximación polinomial-temporal para resolver el CAPMDSP en grafos planas bajo capacidad máxima acotada y demanda máxima acotada. Li et al. (2018), propone un algoritmo de búsqueda local que utiliza un método de puntuación dinámico, que se basa en la penalización de vértices para instruir iterativamente

un conjunto dominante capacitado. Donde el nodo con la mayor puntuación es seleccionado como nodo dominante y que, en caso de existir un empate se selecciona el nodo con mayor capacidad. Además, consideran una estrategia codiciosa y aleatoria, en la selección de los nodos dominados, con tal de producir soluciones más favorables. Este enfoque fue denominado BL_PD[2] y muestra un rendimiento significativamente mejor que OCH, AG y MGRA-CM cuando se aplica a grafos generales con capacidad uniforme y variable. Por último, dado que el algoritmo posee un número reducido de parámetros, puede ser fácilmente adaptado a otros problemas de combinatoria.

Pinacho-Davinson et al. (2019) desarrolla un enfoque denominado construir, fusionar, resolver y adaptar (CFRA). Es una propuesta híbrida que integra un algoritmo heurístico y un solucionador de programación lineal entera (PLE) para acelerar el proceso de solución. En términos generales, el modelo PLE se resuelve iterativamente en un conjunto reducido de vértices, y durante cada iteración el conjunto de vértices se actualiza mediante la construcción de un conjunto dominante capacitado a través de una heurística GRASP[3]. Tras esto, se resuelve el modelo PLE, repitiéndose el proceso una y otra vez durante un periodo de tiempo fijo y la mejor solución encontrada durante este tiempo se devuelve como la solución encontrada por CFRA. Además, se debe tener en cuenta que fueron los primeros en utilizar un PLE en todas las instancias de problemas de la literatura, obteniendo resultados computacionales que muestran que tanto CPLEX como CFRA superan a BL_PD, a costa de presentar mayores tiempos computacionales.

Pinacho-Davinson et al. (2020) propone dos algoritmos, el primero es una versión extendida de construir, fusionar, resolver y adaptar. Mientras que el segundo es un híbrido entre un algoritmo genético de clave aleatoria sesgada y un enfoque exacto denominado BARRAKUDA. BARRAKUDA utiliza en cada iteración, la población actual de individuos para generar una sub-instancia de las instancias del problema abordado. Esta sub-instancia es entonces resuelta por el solucionador exacto CPLEX y la solución resultante se transforma en un individuo que se devuelve a la población. Estas dos propuestas fueron testeadas haciendo uso de las instancias de referencia utilizados por la literatura y en un nuevo conjunto de instancias con un mayor grado de dificultad. Respecto al rendimiento de

[2] BL_PD: Búsqueda local para el problema de dominación.
[3] GRASP: Greedy randomized adaptive search procedure.

los algoritmos, ambos tuvieron un rendimiento similar, siendo eficientes al utilizar las instancias de la literatura. Mientras que BARRAKUDA obtiene un mayor rendimiento en las nuevas instancias propuestas.

Por último, Nakkala et al. (2021) propone tres enfoques para el CAPMDSP, múltiples inicios de búsqueda local iterada (MI-BLI), un enfoque exacto basado en un modelo mejorado de PLE y una combinación de ambos enfoques a través de la fusión de soluciones. Donde los resultados computacionales, en las instancias de referencia estándar muestran una superioridad de los enfoques frente a los presentados anteriormente, tanto en calidad de solución como también en términos de tiempo de ejecución. Además, se debe tener en cuenta que MI-BLI utiliza la misma heurística para la generación de la solución inicial y para la búsqueda local, demostrando que el grado de preferencia de los nodos cambia al retener una fracción de los dominantes.

3. Solución Propuesta

En este capítulo se presenta la metaheurística para resolver el CAPMDSP. Se explica el algoritmo propuesto, y se detalla cada operador involucrado en la metaheurística.

3.1 Estructura general

La metaheurística propuesta toma como inspiración la metaheurística H-MUEC y el algoritmo de múltiples inicios de búsqueda local iterada, presentada en Nakkala et al. (2021). La metaheurística propuesta considera la modificación de H-MUEC, respecto a nuevos criterios de selección de nodos y una nueva forma de eliminar los nodos redundantes. Además, también se cambia la perturbación que forma parte de la búsqueda local. Los nuevos criterios de selección de nodos buscan escoger nodos de manera más eficiente, reduciendo el número de iteraciones involucradas. Por otro lado, la eliminación de nodos redundantes busca disminuir la cardinalidad del conjunto capacitado, disminuyendo la cantidad de ciclos involucrados. Mientras, la perturbación busca asegurar una base factible entre las iteraciones y aportar diversificación a la solución. A continuación, en la Tabla 1 se presenta la notación utilizada en los algoritmos que se presentan junto a su explicación en detalle.

Tabla 1: Notación importante.

Notación	Significado
G	Grafo Original.
G_c	Grafo Actual.
V_c	Conjunto de nodos presentes en G_c.
E_c	Conjunto de pares de nodos presentes en G_c.
$d_G(v_i)$	Grado del nodo v_i en G.
$d_{G_c}(v_i)$	Grado del nodo v_i en G_c.
$C(v_i)$	Capacidad del nodo v_i.
$EC(v_i)$	Capacidad efectiva del nodo v_i en G_c.
$N_{EC_{sum}}(v_i)$	Suma de la capacidad efectiva adyacente del nodo v_i en G_c.
LB	Límite inferior.
$Best$	Mejor solución encontrada.

Fuente: Elaboración propia.

En la notación presentada en la Tabla 1, se nota la presencia de G y de G_c, estos son grafos que contienen la información de los distintos vértices V y de las aristas E que los conectan. Donde G contiene la información del grafo original, mientras que en G_c se realizan todas las actualizaciones ya sean eliminación de vértices junto a sus aristas o restauración de estos mismos. Es necesario aclarar que gracias a la información contenida en G es posible saber qué información se restaura. De lo anterior, es posible inferir que, durante las distintas iteraciones del algoritmo, las aristas de los vértices se van modificando debido a la permanencia de los vértices dentro del grafo G_c. Por tanto, sus grados varían respecto del grafo original G, por lo que esta información se recopila en $d_{G_c}(v_i)$ y $d_G(v_i)$, respectivamente. Respecto a $C(v_i)$ esta es la capacidad uniforme que es determinada como parámetro. Respecto de la capacidad efectiva $EC(v_i)$, este concepto es introducido por Potluri y Singh (2012), representando el valor mínimo entre la capacidad del nodo y su grado dentro de G_c, esto es presentado en la Ecuación (9), donde se debe reiterar que de este modo la capacidad utilizada puede tener como función ser una cota superior del vértice a tratar. Para el caso de $N_{EC_{sum}}(v_i)$, se trata de la suma de las capacidades efectivas de los vértices adyacentes al nodo a evaluar. En el caso de LB, este es el límite inferior propuesto por Nakkala et al. (2021), siendo una versión mucho más estricta al utilizado por la literatura anteriormente mencionada, este límite se presenta en la Ecuación (10) y donde se debe tener en cuenta que la suma de los valores debe realizarse de manera creciente. Finalmente, $best$ es la mejor solución encontrada durante las iteraciones.

$$EC(v_i) = \min\left(C(v_i), d_{G_c}(v_i)\right) \tag{9}$$

$$\sum_{v \in V} \min\left(C(v_i), d_{G_c}(v_i)\right) + 1 = |V| \tag{10}$$

Algoritmo 1: Múltiples inicios de búsqueda local iterada.

1. $i := 0$
2. $best := V$
3. **while** $((i < N_0) \wedge (|best| > LB))$ **do**
4. $\quad S_i := \mathrm{H} - \mathrm{MUEC}\ (\emptyset, G, G_c)$
5. $\quad no\_mejora := 0$
6. $\quad$ **while** $((no\_mejora < N_1) \wedge (|best| > LB))$ **do**
7. $\quad\quad (S, G_c) := Perturbacion(S_i, G)$
8. $\quad\quad S := H - MUEC(S, G, G_c)$
9. $\quad\quad$ **if** $(|S| < |S_i|)$ **then**
10. $\quad\quad\quad S_i := S$
11. $\quad\quad\quad$ **if** $(|S_i| < |best|)$ **then**
12. $\quad\quad\quad\quad best := S_i$
13. $\quad\quad\quad no\_mejora := 0$
14. $\quad\quad$ **else**
15. $\quad\quad\quad no\_mejora := no\_mejora + 1$
16. $\quad i := i + 1$
17. **return** $best$

La metaheurística propuesta se presenta en el Algoritmo 1. Esta realiza múltiples inicios de búsqueda local iterada, donde cada búsqueda requiere el uso del H-MUEC. El H-MUEC se encarga tanto de generar una solución inicial como de realizar la búsqueda local, tras perturbar la solución inicial. En resumidas cuentas, el Algoritmo 1 se basa en una ILS, metaheurística ampliamente utilizada en problemas de optimización combinatoria debido a sus características, como por ejemplo su simplicidad, facilidad de implementación, robustez y eficacia (Lourenço et al, 2003). Dentro de su estructura general se reconocen cuatro componentes principales, una solución inicial, la búsqueda local, el procedimiento de perturbación y el criterio de aceptación. Además, es necesario mencionar que la solución encontrada se trata de la mejor solución global entre las mejores encontradas en cada inicio. Así, el Algoritmo 1, se inicia con una variable *i* que se encarga de registrar el número de inicios (línea 1), posterior a esto la variable best se inicializa conteniendo el número de nodos

que posee el grafo a trabajar (línea 2) y luego se actualiza con el menor conjunto dominante capacitado que se ha encontrado. Posteriormente, se establecen los criterios de término (línea 3), el primero es el máximo N_0 iteraciones y el segundo es que la mejor solución encontrada best debe ser mayor al límite LB previamente calculado, ambos criterios deben ser cumplidos para que el algoritmo pueda finalizar. Tras tener los criterios de término, se procede a obtener una solución inicial S_i mediante el H-MUEC (línea 4) e inicializar un contador no_mejora (línea 5) que se encarga de registrar el número de iteraciones en que no se ha mejorado la solución encontrada. Posterior a esto, se establecen los criterios de término de cada búsqueda local iterada (línea 6), el primero busca que no se sobrepasen N_1 iteraciones sin mejorarse la solución y la segunda es que la mejor solución encontrada *best* debe ser mayor al límite *LB*. Luego, dentro del ciclo se perturba la solución inicial S_i (línea 7), la cual conserva cierta cantidad de nodos de la solución capacitada dependiendo del parámetro *A*, teniendo como consideración que se asegura una base de nodos que ingresaron por el criterio de selección de grado uno. Posterior a esto, se procede a entregarle la solución perturbada al H-MUEC (línea 8) para que procese una nueva solución. Tras esto se compara la nueva solución con la obtenida inicialmente (línea 9) y en caso de ser mejor pasa a ser la S_i (línea 10). Después, se realiza la comparación con *best* (línea 11), en caso de ser mejor, la variable *best* se actualiza (línea 12) y el contador no_mejora se reestablece (línea 13), procediendo a evaluar el realizar un nuevo inicio. En caso contrario, se aumenta el contador de no_mejora (línea 15) y se procede a evaluar si se realiza una nueva búsqueda local. Tras haberse realizado las N_1 iteraciones sin mejorar la solución o que *best* es inferior al limite *LB*, se aumenta en uno el contador *i* (línea 16) con tal de proceder a un nuevo inicio. Finalmente, ya sea porque se realizaron N_0 iteraciones o *best* es inferior al limite *LB*, se finaliza el algoritmo, obteniéndose la mejor solución obtenida en los múltiples inicios (línea 17)

3.2 Representación de la solución.

La solución obtenida por la metaheurística está representada como una lista de elementos, estos representan los nodos dominantes de un determinado grafo. Por ejemplo, para el problema presentado anteriormente en la Sección 2.1. Convirtiendo las letras en números es posible obtener lo presentado en (11). Además, es necesario mencionar que es

posible obtener una lista de nodos dominados, la cual se representa como una lista de listas. De esta manera el nodo uno domina a los nodos cuatro y cinco, como se observa en (12).

$$[\,1, 7, 8] \tag{11}$$

$$\big[[4, 5], [2, 6, 10], [3, 9]\big] \tag{12}$$

3.3 Algoritmo H-MUEC.

Este algoritmo se utiliza para poder obtener las soluciones iniciales en cada inicio y para las búsquedas locales tras entregarle una solución inicial previamente obtenida. La heurística utilizada toma como inspiración la presentada en Nakkala et al. (2021), quienes la denominan heurística para la máxima utilización efectiva de la capacidad (H-MUEC). Para la solución inicial, se comienza con un conjunto solución vacío que posteriormente mediante un proceso iterativo, va seleccionando nodos dominantes y dominados mediante criterios específicos. Además, se incluye un proceso de eliminación de nodos redundantes con tal de excluir nodos que solo incrementan la solución del grafo. La finalidad de la metaheurística es seleccionar nodos de manera que pueda utilizar la capacidad máxima que posee y al mismo tiempo, evitar nodos de menor grado que podrían incrementar la cardinalidad de la solución final. Con tal de poder comprender de mejor manera el algoritmo previamente comentado se procede a explicar los criterios de selección de nodos dominantes (Sección 3.4.1), los criterios de selección de nodos dominados (3.4.2), el proceso de eliminación de nodos redundantes (Sección 3,4,3).

3.3.1 Selección de nodos dominantes

La selección de los distintos nodos dominantes se realiza mediante los siguientes criterios.

1. Nodo aislado: Si un nodo $v_i \in G$ posee un grado de valor 0, se denomina nodo aislado, añadiéndose a la solución en las iteraciones iniciales. Además, se debe tener en cuenta que cuando $v_i \in G$ no es un nodo aislado, pero tras transcurrir una cierta cantidad de iteraciones, se convierte en un nodo aislado en G_c. Entonces, un nodo no dominante $v_j \in G$ con la mayor capacidad efectiva en G y que es adyacente a v_i en G debe ser ingresado a G_c. Se debe tener claro que esta reincorporación del nodo debe

ir acompañada con la restauración de las conexiones de los nodos que no pertenecen a la solución *S*. De este modo, se produce una iteración dentro del algoritmo sin realizarse una incorporación al conjunto dominante. Por otro lado, en caso de no existir un nodo v_j no dominante a restaurar, el propio nodo v_i se vuelve dominante, añadiéndose a la solución. Este procedimiento de restauración de nodos evita el aumento de la cardinalidad del conjunto solución, evitando a los nodos que se dominen a sí mismos y prefiriendo a los nodos adyacentes que pueden llegar a dominar con la totalidad de su capacidad. La heurística siempre comienza con la búsqueda de estos nodos aislados y en caso de no existir pasa al siguiente criterio de selección.

2. Nodo de grado uno: Si en el grafo G_c existen nodos de grado uno, se procede a obtener una lista de los adyacentes a cada uno de estos. De manera de seleccionar preferentemente aquel nodo que posea la mayor cantidad de nodos de grado uno, como adyacente. De forma de evitar que se agreguen nodos de grado cero en la solución *S*. Debido a que el único nodo adyacente que este posee en G ya forma parte de la solución. En caso de no existir al menos un nodo de grado uno, se procede al paso tres.

3. Múltiples nodos adyacentes: Si no existe un nodo aislado o un nodo grado uno presente en el grafo G_c. Entonces, un nodo v_i se selecciona como nodo dominante en función de su capacidad efectiva o de la suma de las capacidades efectivas adyacentes. Por tanto, se selecciona como nodo dominante aquel nodo que posee la capacidad efectiva más alta en G_c. En caso de existir un empate entre capacidades efectivas, se procede a obtener la suma de capacidades efectivas adyacentes en G_c, seleccionando como dominante el que posea la menor suma. Si todavía existen empates, estos se rompen arbitrariamente. La motivación de utilizar el concepto de capacidad efectiva, ayuda a minimizar el número de nodos en conjunto dominante. Así, se evitan los casos donde se presente una capacidad alta pero menos grado, lo que reduce su capacidad de grado. Adicionalmente, utilizar la mínima suma de

capacidad efectiva adyacente, previene seleccionar un nodo dominante que en la última iteración dificulte minimizar la cardinalidad de la solución *S*.

3.3.2 Selección de nodos dominados

Para la selección de los nodos dominados, se debe tener en cuenta que el número no debe exceder la restricción de capacidad del nodo dominante. La selección se realiza según los siguientes criterios:

- Si la capacidad de un nodo dominante es mayor o igual que su grado en G_c, entonces todos sus adyacentes se convierten en dominantes.
- Si el punto anterior no se cumple, solo los adyacentes iguales a la capacidad pueden ser dominados, los cuales se seleccionan uno a uno en orden no decreciente respecto al grado en G_c. Donde los empates se rompen arbitrariamente hasta que el número alcance la capacidad del nodo dominante. En este caso, se seleccionan los nodos de menor grado para así evitar que estos se vuelvan dominantes en futuras iteraciones.

Además, se debe tener en cuenta que se producen excesos de capacidad, los cuales se rellenan con nodos adyacentes al azar que pueden pertenecer al conjunto solución o al de dominados de otro nodo dominante. Por último, hay que mencionar que los nodos dominados son almacenados en una lista denominada l_v.

3.3.3 Eliminación de nodos redundantes

Con tal de poder disminuir aún más la cardinalidad del conjunto dominante, se considera una eliminación de nodos redundantes al igual que en Nakkala et al. (2021), pero cambiando el procedimiento de eliminación. Entonces, dado un conjunto dominante capacitado *D*, se le elimina un nodo $v_i \in D$ y el conjunto sigue siendo dominante, entonces, el nodo v_i se denomina nodo redundante. La estrategia consiste en actualizar la solución, eliminando nodos dominados que tras iteraciones posteriores habían sido seleccionados

como dominantes, posterior a esto se procede a seleccionar un nodo v_i que cumpla las siguientes características.

- Que exista dentro del conjunto capacitado, un nodo v_j con capacidad excedente que sea adyacente al nodo dominante v_i.
- Que exista dentro del conjunto capacitado, un nodo v_j con capacidad excedente que sea adyacente para cada elemento de $l_v(v_i)$.

En términos generales, el primer punto busca un nodo dominante dentro de *S* que pueda admitir al candidato a redundante v_i, pasando de ser un nodo dominante a un nodo dominado, almacenando v_i en r_{cand}. Mientras que el segundo punto, busca redistribuir dentro del conjunto solución los nodos dominados que poseía el nodo v_i, cuando se logra reubicar la totalidad de dominados, se almacena el nodo v_i en l_{cand}. Al cumplirse los puntos anteriores, es posible "reubicar" tanto el nodo dominante v_i como los nodos que este dominaba, es decir, se realiza una redistribución de los nodos. Además, este proceso de eliminación de nodos redundantes se realiza de manera iterativa, seleccionando los nodos redundantes ya sea con una estrategia codiciosa o una estrategia aleatoria.

4. Experimentos computacionales

En esta sección se presentan las instancias de referencia utilizadas y los distintos resultados computacionales obtenidos junto a su análisis.

La implementación de la metaheurística fue creada en el lenguaje de programación Python 3 y fue ejecutada en un computador con procesador Intel Core i5-9300HCPU @ 2.40GHz, memoria RAM de 8 GB y usando un sistema operativo Windows 10 Home. Todos los experimentos fueron ejecutados usando solo un hilo del procesador (secuencialmente).

4.1 Descripción de las instancias

El desempeño de este enfoque ha sido evaluado con las mismas instancias de referencia utilizadas en la literatura desde Potluri y Singh (2012). La instancia utilizada consta de grafos generales que han sido tomados de las instancias tipo-I desde Jovanovic (2010). Estos poseen seis tamaños diferentes los cuales son 50, 100, 250, 500, 800, 1000 nodos. Además, el número de aristas varía de 100 a 10.000 según el número de nodos en el grafo. Respecto a la capacidad utilizada, estas son uniformes, tomando valores específicos siendo el dos y el cinco.

La metaheurística propuesta utiliza cuatro parámetros, N_0, N_1, p_r y A. En todos los experimentos, el número de inicios N_0 se establece en 75 y el número máximo de iteraciones sin mejoras (N_1) en la calidad de la solución se establece en 150 por cada inicio. La estrategia aleatoria se utiliza con probabilidad $p_r = 0.4$, en el proceso de eliminación de nodos redundantes. El parámetro A se establece en 5, es decir, solo el 20% de los nodos dominantes se conservan después de la perturbación. Los valores de todos los parámetros se establecen utilizando como fuente la literatura, más específicamente, los obtenidos empíricamente por Nakkala et al. (2021).

4.2 Resultados

Al ejecutar las instancias anteriormente mencionadas, es posible obtener los resultados de soluciones promedios y mínimos, considerando las capacidades dos y cinco, utilizando el PLEM, el algoritmo propuesto (Algoritmo 1) y un algoritmo híbrido. El algoritmo híbrido, integra el PLEM en el Algoritmo 1, de tal manera que complementa al H-MUEC. De manera más específica, se genera una solución mediante H-MUEC, la que será introducida en el PLEM como solución inicial, en caso de realizarse la mejora de la solución dada, es decir, existe una disminución de la solución capacitada se procede a realizar un nuevo inicio asegurándose que se cumple lo descrito en la línea 3 del Algoritmo 1. En caso contrario, al existir una mejora se procede a mantener la solución dada la que será perturbada para luego realizarse la búsqueda local, esta solución que proviene de la búsqueda local se vuelve a evaluar mediante PLEM siguiendo un proceso iterativo. De esta manera, se busca probar si existe una disminución en los tiempos de procesamiento al otorgar una solución inicial al modelo PLEM. Los resultados específicos se encuentran en el Anexo. A continuación, se presenta la Tabla 2 y 3, en las cuales se encuentran la comparación de los algoritmos teniendo en cuenta los valores de la solución y tiempos promedios en segundos para capacidad dos y cinco respectivamente.

Tabla 2: Resultados obtenidos con capacidad 2.

Instancia	PLEM		Algoritmo propuesto		Algoritmo híbrido	
	Solución	Tiempo	Solución	Tiempo	Solución	Tiempo
V50E100	17,0	0,014	17,7	0,006	17,0	0,011
V50E250	17,0	0,011	17,3	0,011	17,0	0,019
V50E500	17,0	0,022	17,0	0,02	17,0	0,023
V100E100	34,0	0,014	34,7	0,01	34,0	0,006
V100E250	34,0	0,019	34,6	0,028	34,0	0,047
V100E500	34,0	0,025	34,3	0,043	34,0	0,061
V250E250	84,0	0,020	84,8	0,063	84,0	0,022
V250E500	84,0	0,105	84,8	0,107	84,0	0,124
V250E1000	84,0	0,131	84,2	0,189	84,0	0,253
V500E500	167,0	0,031	167,0	0,223	167,0	0,034
V500E1000	167,0	0,420	168,5	0,412	167,0	0,445
V500E2000	167,0	0,522	168,1	0,851	167,0	0,541
V800E1000	267,0	0,586	269,5	0,653	267,0	0,460
V800E2000	267,0	1,105	269,2	1,349	267,0	1,284

V800E5000	267,0	1,805	269,9	2,473	267,0	1,661
V1000E1000	334,0	0,077	334,9	0,844	334,0	0,141
V1000E5000	334,0	2,597	334,5	3,414	334,0	2,422
V1000E10000	334,0	3,983	334,4	5,428	334,0	3,402

Fuente: Elaboración propia.

Tabla 3: Resultados obtenidos con capacidad 5.

Instancia	**PLEM**		**Algoritmo propuesto**		**Algoritmo híbrido**	
	Solución	**Tiempo**	**Solución**	**Tiempo**	**Solución**	**Tiempo**
V50E100	11,9	0,020	12,5	0,003	14,7	0,020
V50E250	9,0	0,050	9,8	0,006	9,0	0,042
V50E500	9,0	0,033	9,2	0,011	9,0	0,039
V100E100	33,6	0,013	33,6	0,012	33,6	0,003
V100E250	20,5	0,092	22	0,009	22,5	0,075
V100E500	17,0	0,180	18,3	0,016	17,0	0,236
V250E250	83,3	0,019	83,3	0,07	83,3	0,017
V250E500	57,8	0,161	62,2	0,048	73,0	0,114
V250E1000	42,0	1,703	44,2	0,079	42,0	2,649
V500E500	167,1	0,036	167	0,258	167,1	0,026
V500E1000	114,2	0,416	123,4	0,19	143,2	0,363
V500E2000	84,0	22,964	94,3	0,298	84,0	10,616
V800E1000	247,0	0,128	249,1	0,608	277,0	0,075
V800E2000	164,5	263,364	178,2	0,468	180,5	188,278
V800E5000	134,0	14,434	142,4	1,064	134,0	12,927
V1000E1000	333,7	0,094	334,7	1,265	333,7	0,083
V1000E5000	167,0	36,728	182,5	1,450	167,0	34,425
V1000E10000	167,0	36,736	169,8	2,545	167,0	35,608

Fuente: Elaboración propia.

De los resultados obtenidos, es posible evidenciar que para el caso de la capacidad dos, el algoritmo propuesto entrega en promedio soluciones 0,6% menos precisas y 40% más lentas que PLEM. A pesar de que el algoritmo propuesto arroja conjuntos soluciones similares a PLEM, posee mayores tiempos computacionales, sobre todo para las instancias más grandes. Para el caso de la capacidad de cinco ocurre algo similar a la hora de comparar la cantidad de nodos seleccionados, el algoritmo propuesto presenta soluciones promedio 4%

menos precisas, pero 97,8% más rápido respecto al PLEM, es decir, la propuesta presenta tiempos menores a costa de una variación mínima de solución. Además, mediante una función encargada de comprobar la factibilidad de las soluciones corroborando que estas eran factibles. Adicionalmente, es necesario destacar que en general el algoritmo propuesto presenta variaciones de uno a dos nodos respecto al valor obtenido por el modelo matemático, aun así, existen diferencias considerables como es el caso de la instancia "V1000E5000" con una variación de quince nodos. Analizando el algoritmo híbrido respecto al PLEM, para capacidad dos, este no presenta mejoras a la hora de comparar las soluciones promedio, pero resulta ser 4,6% más rápido. Por otra parte, con la capacidad de cinco, el híbrido entrega soluciones promedio 5,1% menos precisas, pero 24,3% más rápidas. En general, el algoritmo híbrido presenta un buen comportamiento tanto en cardinalidad de la solución como en el rendimiento de los tiempos cuando se tratan de instancias grandes del problema, produciendo una mejora respecto al PLEM. Pero, cuando se tratan de instancias pequeñas los resultados y tiempos son aceptables pero peores en comparación al PLEM como al algoritmo propuesto.

Con relación a las modificaciones incorporadas en esta propuesta, primero tenemos el uso de los nodos que comparten más de un "nodo de grado uno" en la selección de los dominantes. Este proceso evita la eliminación y posterior restauración de nodos para prevenir la formación de nodos aislados que en realidad no lo son, pero con un porcentaje de ocurrencia cercano al 20% entre todas las instancias, es decir, la mejora surte efecto ante grafos con gran presencia de nodos de grado uno, pero en la mayoría de las iteraciones verificadas no presenta mayor protagonismo. Esto fue comprobado empíricamente mediante la incorporación de un contador dentro del criterio de selección de nodos aislados para la propuesta original como para la modificación y realizando la comprobación mediante la comparación directa de ambos valores. Por tanto, se evitan iteraciones de reincorporación de nodos adyacentes disponibles. Se debe tener en cuenta que tanto la comprobación de la factibilidad de las soluciones y el número de iteraciones comprometidas en el Algoritmo 1, fueron realizadas empíricamente. Respecto a la perturbación utilizada, se produce una mejora de la solución inicial, cabe destacar que esta considera que solo el 20% de los nodos de la solución inicial se conservan y la otra mitad son nodos que ingresaron a la solución mediante el criterio de selección de vértices de grado uno. Finalmente, el procedimiento de eliminación de nodos redundantes no produce los resultados esperados, debido a que se dan casos donde

a pesar de que existen candidatos con capacidad excedente para cubrir al nodo dominante y existe capacidad excedente para cubrir la mayoría de sus correspondientes nodos dominados, el algoritmo selecciona solo un nodo dominado que no pueda ser reubicado, produciendo que se mantenga el nodo evaluado (candidato a redúndate) dentro del conjunto solución.

Finalmente, es necesario tener en cuenta que el lenguaje de programación utilizado influye en los tiempos de ejecución, ya que Python 3 es aproximadamente 46 veces más lento que C++ (Pereira et al, 2017). Por tanto, los tiempos presentados por la literatura utilizan un lenguaje mucho más favorable a la hora de trabajar con metaheurísticas. Además, existen diversos factores que influyen a la hora de obtener los resultados presentados como son el hardware y la optimización del código al igual que su diseño.

5 Conclusión

En esta memoria de titulación se presenta una metaheurística para resolver el CAPMDSP, basado en una estrategia de múltiples inicios de búsqueda local iterada, que hace uso del algoritmo de búsqueda local iterada, llamado H-MUEC. H-MUEC es usado para obtener la solución inicial y para realizar las búsquedas locales. Dentro de la evaluación de las modificaciones incorporadas en la propuesta. Primero se tiene que el nuevo criterio de selección de nodos de grado uno provoca que no se produzcan tantas iteraciones de restauración de nodos debido al criterio de nodos aislados, respecto al modelo tomado como inspiración. La perturbación presenta un buen rendimiento, generando una suficiente aleatoriedad, con tal de poder obtener soluciones iniciales diferentes. Por último, el proceso de eliminación de nodos planteado no presenta los resultados esperados, ya que no se evidencia una disminución significativa de la cardinalidad del conjunto solución entre iteraciones.

Analizando los resultados obtenidos tras realizar las modificaciones nombradas anteriormente, estas muestran que las soluciones obtenidas son factibles, pero varían considerablemente en algunas instancias respecto al PLEM. Además, con relación a los tiempos de procesamiento estos son superiores a los vistos en la literatura.

Respecto a la alternativa híbrida, esta posee una mejora respecto a la PLEM cuando se trata de las instancias más grandes, por lo que entregarle una solución inicial se traduciría en menores tiempos de procesamiento. Aun así, no resulta atractivo su uso cuando se trata de instancias o capacidades pequeñas del problema.

Para finalizar, resulta interesante evaluar como solución al problema que se genera en la eliminación de nodos redundantes, realizar una segunda reubicación de nodos, en este caso, en los adyacentes al nodo que genera el problema. Finalmente, se puede evaluar el comportamiento de la propuesta en una implementación en C++.

6 Referencias

Becker. (2016). *Capacitated dominating set on planar graphs. CoRR, DOI: abs/1604.04664.*

Cygan, P. W. (2011). Capacitated domination faster than O(n2),. *Inform. Process. Lett. 111 (23–24)*, 1099-1103.

Díez Corral, &. G. (2020). Conjunto dominante conexo de un grafo. *Universidad de Zaragoza, CIEN*.

Garey, J. (1979). Computers and Intractability: a guide to the theory of NP-completeness.

H.R. Lourenço, O. M. (2003). Iterated Local Search, in: International Series in Operations Research and Management Science. *Springer*, 321–354.

Kao, C. (2010). Approximation algorithms for the capacitated domination problem. *CoRR, DOI: abs/1004.2839*.

Kuhn, M. (2010). Distributed approximation of capacitated dominating sets. *Theory Comput. Syst.*, 811-836.

Li, H. Z. (2018). A novel local search algorithm for the minimum capacitated dominating set. *J. Oper. Res. Soc. 69 (6)*, 849-863.

Liedloff, T. V. (2014). Solving Capacitated Dominating Set by using covering by subsets and maximum matching. *Discrete Appl. Math.*, 60-68.

Nakkala, S. R. (2021). Multi-start iterated local search, exact and matheuristic approaches for minimum capacitated dominating set problem. *Applied Soft Computing, 108*.

Pedro Pinacho-Davidson, C. B. (2020). BARRAKUDA: A Hybrid Evolutionary Algorithm for Minimum Capacitated Dominating Set Problem.

Pereira, R. C. (2017). Energy efficiency across programming languages: how do energy, time, and memory relate? *SLE 2017: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, 256-267.

Pinacho-Davidson, B. B. (2019). Application of CMSA to the minimum capacitated dominating set problem, in: Proceedings of the Genetic and Evolutionary Computation Conference 2019. 321-328.

Potluri, S. (2012). A greedy heuristic and its variants for minimum capacitated dominating set. *Commun. Comput. Inf. Sci. 306*, 28-39.

Potluri, S. (2013). Metaheuristic algorithms for computing capacitated dominating set with uniform and variable capacities. *Swarm Evol. Comput. 13*, 22-33.

R. Jovanovic, M. T. (2010). Ant colony optimization applied to minimum weight dominating set problem, in: Proceedings of the 12th WSEAS International

Conference on Automatic Control, Modelling and Simulation. *ACMOS'10,*, pp. 322–326.

7 Anexo

7.1 Resultados PLEM, Algoritmo propuesta, Algoritmo Híbrido.

Tabla 4: Resultados PLEM con capacidad 2.

Instancia	Solución		Tiempo	
	Mínima	Promedio	Mínimo	Promedio
V50E100	17,0	17,0	0,000	0,014
V50E250	17,0	17,0	0,000	0,011
V50E500	17,0	17,0	0,015	0,022
V100E100	34,0	34,0	0,000	0,014
V100E250	34,0	34,0	0,015	0,019
V100E500	34,0	34,0	0,016	0,025
V250E250	84,0	84,0	0,015	0,020
V250E500	84,0	84,0	0,063	0,105
V250E1000	84,0	84,0	0,078	0,131
V500E500	167,0	167,0	0,031	0,031
V500E1000	167,0	167,0	0,297	0,42
V500E2000	167,0	167,0	0,422	0,522
V800E1000	267,0	267,0	0,297	0,586
V800E2000	267,0	267,0	0,672	1,105
V800E5000	267,0	267,0	1,312	1,805
V1000E1000	334,0	334,0	0,063	0,077
V1000E5000	334,0	334,0	2,187	2,597
V1000E10000	334,0	334,0	3,188	3,983

Fuente: Elaboración propia.

Tabla 5: Resultados PLEM con capacidad 5.

Instancia	Solución		Tiempo	
	Mínima	**Promedio**	**Mínimo**	**Promedio**
V50E100	11,0	11,9	0,015	0,020
V50E250	9,0	9,0	0,015	0,050
V50E500	9,0	9,0	0,015	0,033
V100E100	33,0	33,6	0,000	0,013
V100E250	21,0	22,5	0,062	0,092
V100E500	17,0	17,0	0,094	0,18
V250E250	83,0	83,3	0,015	0,019
V250E500	57,0	57,8	0,094	0,161
V250E1000	42,0	42,0	0,781	1.703
V500E500	167,0	167,1	0,031	0,036
V500E1000	114,0	114,2	0,344	0,416
V500E2000	84,0	84,0	9.031	22.964
V800E1000	242,0	247,0	0,109	0,128
V800E2000	164,0	164,5	10,703	263,364
V800E5000	134,0	134,0	12,407	14,434
V1000E1000	333,0	333,7	0,062	0,094
V1000E5000	167,0	167,0	23,281	36,728
V1000E10000	167,0	167,0	29,907	36,736

Fuente: Elaboración propia.

Tabla 6: Resultados Algoritmo propuesto con capacidad 2.

Instancia	Solución		Tiempo	
	Mínima	**Promedio**	**Mínimo**	**Promedio**
V50E100	17,0	17,7	0,006	0,006
V50E250	17,0	17,3	0,010	0,011
V50E500	17,0	17,0	0,015	0,020
V100E100	34,0	34,7	0,001	0,010
V100E250	34,0	34,6	0,024	0,028
V100E500	34,0	34,3	0,036	0,043
V250E250	84,0	84,8	0,055	0,063
V250E500	84,0	84,8	0,095	0,107
V250E1000	84,0	84,2	0,182	0,189
V500E500	167,0	167,0	0,220	0,223
V500E1000	168,0	168,5	0,391	0,412
V500E2000	168,0	168,1	0,717	0,851
V800E1000	269,0	269,5	0,638	0,653
V800E2000	269,0	269,2	1,266	1,349
V800E5000	269,0	269,9	2,254	2,473
V1000E1000	334,0	334,9	0,808	0,844
V1000E5000	334,0	334,5	3,318	3,414
V1000E10000	334,0	334,4	4,923	5,428

Fuente: Elaboración propia.

Tabla 7: Resultados Algoritmo propuesto con capacidad 5.

Instancia	Solución		Tiempo	
	Mínima	**Promedio**	**Mínimo**	**Promedio**
V50E100	12,0	12,5	0,003	0,003
V50E250	9,0	9,8	0,006	0,006
V50E500	9,0	9,2	0,010	0,011
V100E100	33,0	33,6	0,010	0,012
V100E250	21,0	22,0	0,008	0,009
V100E500	17,0	18,3	0,015	0,016
V250E250	83,0	83,3	0,064	0,070
V250E500	60,0	62,2	0,045	0,048
V250E1000	42,0	44,2	0,070	0,079
V500E500	167,0	167,0	0,251	0,258
V500E1000	120,0	123,4	0,176	0,190
V500E2000	92,0	94,3	0,277	0,298
V800E1000	247,0	249,1	0,598	0,608
V800E2000	176,0	178,2	0,436	0,468
V800E5000	141,0	142,4	0,996	1,064
V1000E1000	334,0	334,7	1,067	1,265
V1000E5000	181,0	182,5	1,421	1,450
V1000E10000	169,0	169,8	2,395	2,545

Fuente: Elaboración propia.

Tabla 8: Resultados Algoritmo Híbrido con capacidad 2.

Instancia	Solución		Tiempo	
	Mínima	**Promedio**	**Mínimo**	**Promedio**
V50E100	17,0	17,0	0,000	0,011
V50E250	17,0	17,0	0,000	0,019
V50E500	17,0	17,0	0,015	0,023
V100E100	34,0	34,0	0,000	0,006
V100E250	34,0	34,0	0,016	0,047
V100E500	34,0	34,0	0,031	0,061
V250E250	84,0	84,0	0,015	0,022
V250E500	84,0	84,0	0,062	0,124
V250E1000	84,0	84,0	0,093	0,253
V500E500	167,0	167,0	0,016	0,034
V500E1000	167,0	167,0	0,218	0,445
V500E2000	167,0	167,0	0,469	0,541
V800E1000	267,0	267,0	0,296	0,460
V800E2000	267,0	267,0	0,797	1,284
V800E5000	267,0	267,0	1,359	1,661
V1000E1000	334,0	334,0	0,063	0,141
V1000E5000	334,0	334,0	1,844	2,422
V1000E10000	334,0	334,0	2,516	3,402

Fuente: Elaboración propia.

Tabla 9: Resultados Algoritmo Híbrido con capacidad 5.

Instancia	Solución		Tiempo	
	Mínima	**Promedio**	**Mínimo**	**Promedio**
V50E100	12,0	14,7	0,000	0,020
V50E250	9,0	9,0	0,015	0,042
V50E500	9,0	9,0	0,016	0,039
V100E100	33,0	33,6	0,000	0,003
V100E250	21,0	22,5	0,047	0,075
V100E500	17,0	17,0	0,172	0,236
V250E250	83,0	83,3	0,000	0,017
V250E500	70,0	73,0	0,093	0,114
V250E1000	42,0	42,0	1,157	2,649
V500E500	167,0	167,1	0,015	0,026
V500E1000	137,0	143,2	0,265	0,363
V500E2000	84,0	84,0	7,453	10,616
V800E1000	272,0	277,0	0,062	0,075
V800E2000	172,0	180,5	10,234	188,278
V800E5000	134,0	134,0	10,140	12,927
V1000E1000	333,0	333,7	0,047	0,083
V1000E5000	167,0	167,0	26,172	34,425
V1000E10000	167,0	167,0	30,641	35,608

Fuente: Elaboración propia.

UNIVERSIDAD DE CONCEPCIÓN – FACULTAD DE INGENIERÍA

RESUMEN DE MEMORIA DE TÍTULO

Departamento	**:** Departamento de Ingeniería Industrial
Carrera	**:** Ingeniería Civil Industrial
Nombre del memorista	**:** Sebastián Ignacio Barría Barría
Título de la Memoria	**:** Una metaheurística para el problema de conjunto dominante con capacidad mínima
Fecha de la presentación oral	**:** Septiembre de 2022
Profesor Guía	**:** Carlos Contreras Bolton
Profesor Revisor	**:** Rosa Medina Durán
Concepto	**:**
Calificación	**:**

Resumen

La presente memoria de título tiene como objetivo presentar el problema de conjunto dominante con capacidad mínima (CAPMDSP), de manera más específica, se trata de una variante del problema de conjunto dominante. CAPMDSP integra una restricción adicional que establece una capacidad máxima que puede dominar cada nodo. Con tal de resolver el CAPMDSP se plantea un algoritmo de múltiples inicios de búsqueda local iterada, el cual hace uso de un algoritmo del estado del arte llamado H-MUEC. H-MUEC es modificado y es utilizado para generar la solución inicial y es aplicado como una búsqueda local. Los resultados generan soluciones factibles pero que difieren respectos a los encontrados por el modelo de programación entera mixta, y los tiempos son mayores a la literatura, debido a la diferencia del lenguaje de programación utilizado.