TESIS DOCTORAL

Nuevos enfoques metaheurísticos para resolver problemas de dominación

Autor

Alejandra Casado Ceballos

Directores

Abraham Duarte Muñoz

Sergio Pérez Peló

Programa de doctorado en Tecnologías de la Información y las Comunicaciones

Escuela Internacional de Doctorado

2025

Esta Tesis Doctoral ha sido desarrollada bajo la financiación de los fondos asociados al proyecto TEC-2024/COM-404 de la Comunidad Autónoma de Madrid, el proyecto PID2021-125709OA-C22 del Ministerio de Economía y Competitividad, el proyecto TSI-100930-2023-3 del Ministerio para la Transformación Digital y de la Función Pública. Por último, este trabajo ha sido financiado mediante el contrato predoctoral C1PREDOC2022, concedido por Universidad Rey Juan Carlos, en el marco de las ayudas destinadas a la formación de personal investigador.

Agradecimientos

🪄 *Juro solemnemente que esto es una Tesis Doctoral*

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

***RefSet* UPD** *Reference Set Update*.

ACO-LS *Ant Colony Optimization Local Search*.
ACO-LS-S *Ant Colony Optimization Local Search Start*.
ACO-PP-LS *Ant Colony Optimization Pre-Processing Local Search*.
APSS *Agglomerative Parallel Scatter Search*.

BVNS *Basic Variable Neighborhood Search*.

CDP *Capacitated Dispersion Problem*.
CGR Constructivo*Greedy Random*.
CPSS *Cooperative Parallel Scatter Search*.
CRG Constructivo*Random Greedy*.

DGM *Diversification Generation Method*.

GA1 *Genetic Algorithm 1*.
GRASP *Greedy Randomized Adaptive Search Procedure*.

HPSS *Hoarding Parallel Scatter Search*.

IG *Iterated Greedy*.
ILP *Integer Linear Programming*.
IMP *Improvement Method*.

JVNS *Jump Variable Neighborhood Search*.

MCP *Max Cut Problem*.
MDSP *Minimum Dominating Set Problem*.
MPP *Monitor Placement Problem*.
MVCP *Minimum Vertex Cover Problem*.

PFSS *Parallel Framework for Scatter Search*.

RCL *Restricted Candidate List*.

RLS *Randomised Local Search*.

SBD *Select Best and Diverse*.
SCM *Solution Combination Method*.
SGM *Subset Generation Method*.
SS *Scatter Search*.
SSS *Sequential Scatter Search*.

VNS *Variable Neighborhood Search*.

WTDP *Weighted Total Domination Problem*.

Resumen

En el contexto actual, donde los sistemas complejos, modelados normalmente como redes, son esenciales para numerosas actividades humanas, resulta crucial identificar rápidamente elementos clave dentro de redes de gran tamaño. Los problemas de dominación en grafos modelan situaciones en las que se debe seleccionar un subconjunto mínimo de nodos estratégicos que cubran eficientemente al resto de la red. Este tipo de problemas es relevante en aplicaciones como el despliegue de infraestructuras de telecomunicaciones, redes de sensores, campañas de vacunación o la detección de vulnerabilidades en sistemas informáticos. Resolverlos con eficacia tiene un efecto directo en la eficiencia operativa, la reducción de costes y la resiliencia de los sistemas.

Sin embargo, su exigencia computacional, especialmente en variantes complejas, limita el uso de algoritmos exactos en escenarios de gran escala debido al crecimiento exponencial del espacio de búsqueda. Aunque estos métodos garantizan soluciones óptimas, sus altos tiempos de cómputo restringen su aplicabilidad práctica. En contraste, los algoritmos heurísticos permiten explorar el espacio de soluciones de manera eficiente, ofreciendo resultados de alta calidad en tiempos razonables, aunque sin asegurar optimalidad global.

Las heurísticas son especialmente valiosas en contextos dinámicos o con recursos limitados, y su flexibilidad permite adaptarse a distintas variantes del problema, considerando múltiples restricciones y objetivos. Así, el desarrollo de métodos heurísticos eficaces para problemas de dominación representa tanto un avance metodológico como una aportación práctica relevante en múltiples dominios donde la toma de decisiones basada en grafos es esencial.

El Problema de Dominación en Grafos, un problema $\mathcal{NP}$-difícil, consiste en identificar un subconjunto mínimo de nodos tal que todos los nodos del grafo estén en él o sean adyacentes a alguno de los elementos de ese subconjunto. Sus aplicaciones incluyen la colocación de sensores, el diseño de redes de vigilancia o la optimización de campañas sanitarias. Entre sus variantes destacan la dominación total, la dominación conexa y la dominación ponderada, que

introducen distintas restricciones y objetivos según el contexto.

Estas variantes pueden modelarse como problemas de optimización, donde el objetivo común es lograr configuraciones eficientes que garanticen una cobertura efectiva del grafo, minimizando el número de nodos dominantes u optimizando otros criterios como coste o robustez.

En esta Tesis Doctoral se proponen diversos algoritmos heurísticos para resolver varias versiones del Problema de Dominación en Grafos. Se han empleado técnicas como *Variable Neighborhood Search* (VNS), *Iterated Greedy* (IG) y *Scatter Search* (SS), evaluadas sobre instancias sintéticas y reales, obteniendo soluciones competitivas frente a los métodos del estado del arte.

Abstract

In the current context, where complex systems (typically modeled as networks) are essential for numerous human activities, it becomes crucial to quickly identify key elements within large-scale networks. Domination problems in graphs model situations where a minimum subset of strategic nodes must be selected to efficiently cover the rest of the network. These types of problems are relevant in applications such as the deployment of telecommunications infrastructure, sensor networks, vaccination campaigns, or the detection of vulnerabilities in information systems. Effectively solving them has a direct impact on operational efficiency, cost reduction, and system resilience.

Among their variants, total domination, connected domination, and weighted domination stand out, each introducing different constraints and objectives depending on the context. These variants can be modeled as optimization problems, where the common goal is to find efficient configurations that ensure effective graph coverage while minimizing the number of dominating nodes or optimizing other criteria such as cost or robustness.

However, these are $\mathcal{NP}$-hard problems, which limits the use of exact algorithms in large-scale scenarios due to the exponential growth of the search space. While these methods guarantee optimal solutions, their high computational time restricts their practical applicability. In contrast, heuristic algorithms allow the solution space to be explored efficiently, offering high-quality results in reasonable timeframes, although without guaranteeing global optimality.

Heuristics are particularly valuable in dynamic contexts or when resources are limited, and their flexibility allows them to adapt to different problem variants, taking into account multiple constraints and objectives. Thus, the development of effective heuristic methods for domination problems represents both a methodological advancement and a practical contribution in various domains where graph-based decision-making is essential.

This PhD Dissertation proposes several heuristic algorithms to solve dif-

ferent versions of the Graph Domination Problem. Techniques such as *Variable Neighborhood Search* (VNS), *Iterated Greedy* (IG), and *Scatter Search* (SS) have been employed, evaluated on both synthetic and real instances, obtaining competitive solutions when compared to state-of-the-art methods.

Parte I

Tesis doctoral

Capítulo 1

Introducción

Este capítulo tiene como objetivo contextualizar el trabajo desarrollado en la Tesis, ofreciendo una introducción al área de la optimización. En la segunda sección se detallan la hipótesis del estudio y los objetivos que se derivan de ella. La última sección tiene como propósito orientar al lector en adelante, describiendo la estructura del documento y el contenido de cada capítulo.

1.1. Optimización

La optimización es una disciplina que ha evolucionado a lo largo de los siglos, con raíces que se remontan a la antigüedad. Desde los estudios de los griegos en geometría hasta los avances en cálculo de variaciones en el siglo XVII y el desarrollo del álgebra lineal en el siglo XX, la optimización ha sido una herramienta esencial en las matemáticas aplicadas, la economía, la ingeniería y la informática. Su objetivo general es encontrar, entre todas las posibles soluciones de un problema, aquella que minimice o maximice una determinada función objetivo, bajo un conjunto de restricciones.

Un problema de optimización puede expresarse de forma general como:

$$\min_{S \in \mathbb{S}} f(S) \tag{1.1}$$

donde S representa una solución candidata perteneciente al conjunto de soluciones factibles $\mathbb{S}$, y $f(S)$ es una función objetivo que se desea minimizar. Las restricciones del problema están implícitas en la definición del conjunto S.

Cuando el espacio de soluciones $\mathbb{S}$ es discreto o finito, como sucede en problemas donde las decisiones son del tipo binario (sí o no), entero o permutación, se habla de optimización combinatoria. Este tipo de problemas aparece con frecuencia en campos como la logística, las telecomunicaciones, la bioinformática y, especialmente, en teoría de grafos, véase [2].

En términos de complejidad computacional, los problemas de optimización se dividen en dos grandes categorías. Los problemas polinomiales, también llamados $\mathcal{P}$, pueden resolverse en un tiempo que crece de forma polinómica con el tamaño de la entrada, es decir, se puede encontrar la solución óptima en un tiempo de cómputo razonable. En cambio, para los problemas no polinomiales, o $\mathcal{NP}$, y en particular los $\mathcal{NP}$-difíciles, no se conocen algoritmos exactos que los resuelvan en tiempo polinómico, aunque sus soluciones pueden verificarse en tiempo polinomial. La mayoría de los problemas relevantes en optimización combinatoria, como los problemas de dominación sobre grafos, pertenecen a esta última categoría.

Dada la complejidad de estos problemas, especialmente los $\mathcal{NP}$-difíciles, se recurre a estrategias de búsqueda que no garantizan encontrar la solución óptima, pero que permiten hallar buenas soluciones en tiempos de cómputo razonables. En este contexto, surge la necesidad de equilibrar dos conceptos fundamentales: la intensificación (también conocida como explotación) y la diversificación (o exploración). La intensificación busca mejorar la solución permaneciendo en la misma región, mientras que la diversificación busca explorar nuevas regiones del espacio de soluciones para evitar que el algoritmo quede atascado en óptimos locales.

En esta Tesis Doctoral se abordan diversos problemas de optimización combinatoria relacionados con la dominación en grafos, un área de investigación con múltiples aplicaciones prácticas. Para resolver estos problemas, se utilizan varias metaheurísticas (ver Sección 3.1). Concretamente, se emplean las siguientes: *Greedy Randomized Adaptive Search Procedure* (GRASP), una metaheurística multiarranque con una propuesta centrada en la construcción de soluciones diversas; *Variable Neighborhood Search* (VNS), basada en cambios sistemáticos de vecindad; *Scatter Search* (SS), un método basado en la combinación de soluciones de alta calidad para explorar nuevas áreas del espacio de búsqueda; e *Iterated Greedy* (IG), un enfoque iterativo que destruye y reconstruye parcialmente una solución para escapar de óptimos locales y mejorar progresivamente.

El tema principal de la tesis es la aplicación y análisis de estas técnicas para abordar variantes de la familia de problemas de dominación en grafos,

buscando estrategias eficientes desde el punto de vista computacional que proporcionen soluciones de alta calidad a estos problemas.

1.2. Hipótesis y objetivos

En el ámbito científico, toda investigación rigurosa debe sustentarse en una hipótesis bien formulada. La hipótesis actúa como el punto de partida del proceso, ya que permite delimitar el objeto de estudio, establecer una dirección clara en el análisis y ofrecer una base para la verificación empírica o teórica. La hipótesis representa una afirmación fundamentada que guía la recogida de datos, la construcción de modelos o la formulación de algoritmos, según el enfoque metodológico adoptado.

Además, la presencia de una hipótesis permite dotar de coherencia interna al trabajo, relacionando las preguntas de investigación con los objetivos planteados. En investigaciones aplicadas, como ocurre a menudo en el estudio de problemas combinatorios y de optimización, la hipótesis también sirve como criterio para evaluar la eficacia de las soluciones propuestas o la validez de las aproximaciones adoptadas.

En este sentido, incluir una hipótesis no sólo responde a una exigencia metodológica, sino que también aporta valor científico al trabajo realizado, al situarlo dentro de un marco de razonamiento crítico. Así, una hipótesis bien definida no limita la creatividad investigadora, sino que la orienta y la convierte en conocimiento estructurado y verificable.

La hipótesis sobre la que se sustenta esta Tesis Doctoral es la siguiente:

> *El problema de la dominación en grafos, siendo un problema de optimización combinatoria clasificado como $\mathcal{NP}$-difícil, presenta un alto interés tanto para la comunidad científica como para profesionales del ámbito de las redes. Por este motivo, resulta especialmente relevante el diseño de algoritmos capaces de resolverlo de forma eficiente, proporcionando soluciones óptimas o, al menos, de alta calidad, en tiempos de cómputo reducidos. Dada la naturaleza compleja del problema, los algoritmos aproximados cobran especial protagonismo, destacando entre ellos las técnicas metaheurísticas por su eficacia y eficiencia a la hora de abordar este tipo de problemas. En esta investigación se estudiarán tanto metaheurísticas poblacionales (que gestionan múltiples soluciones de forma simultánea) como*

metaheurísticas trayectoriales (que parten de una única solución y exploran su entorno generando una trayectoria en el espacio de búsqueda).

El objetivo principal de esta Tesis Doctoral consiste en diseñar y desarrollar algoritmos metaheurísticos capaces de generar soluciones de alta calidad para diferentes variantes de los problemas de dominación sobre grafos, que sean competitivos con respecto al estado del arte existente.

Para alcanzar este objetivo general, se definen los siguientes objetivos específicos:

- Realizar una revisión exhaustiva del estado del arte en relación con los problemas de dominación sobre grafos.
- Analizar y evaluar distintos algoritmos metaheurísticos propuestos en la literatura.
- Adaptar y seleccionar técnicas metaheurísticas para su aplicación a problemas concretos de dominación.
- Participar activamente en la redacción de artículos científicos que documenten los avances y resultados obtenidos.
- Difundir los resultados de la investigación en congresos especializados, fomentando el intercambio de conocimientos con la comunidad investigadora.
- Aplicar los métodos desarrollados a la resolución de problemas reales mediante colaboraciones con el entorno empresarial.

1.3. Estructura de la memoria

Este documento resume el trabajo de investigación desarrollado durante la Tesis Doctoral, y se organiza de la siguiente manera:

- Parte I: Se presenta la investigación llevada a cabo, describiendo el problema abordado, así como la metodología seguida y los resultados obtenidos. Finalmente, se exponen las conclusiones derivadas del trabajo.
    - En el Capítulo 1, se introduce el área de investigación en optimización, describiendo los problemas de optimización combinatoria. También se formulan las hipótesis y los objetivos que guían este trabajo.

- En el Capítulo 2, se presenta la familia de problemas de dominación sobre grafos y se describen las distintas variantes en las que se ha trabajado dentro de esta familia de problemas, además de una propuesta metodológica.
- En el Capítulo 3, se describe la metodología empleada para abordar los problemas previamente definidos. Se detallan las metaheurísticas implementadas y las técnicas utilizadas para alcanzar los resultados obtenidos.
- En el Capítulo 4, se analizan y discuten los resultados obtenidos para cada una de las variantes del problema, destacando las principales aportaciones derivadas del estudio.
- En el Capítulo 5, se presentan las conclusiones extraídas de la investigación, así como posibles líneas de trabajo futuro.

- Parte II: Se recopilan los artículos publicados en revistas indexadas en el JCR, así como los trabajos presentados en congresos nacionales e internacionales asociados a esta tesis.

Capítulo 2

Problemas de dominación en grafos

En esta sección se introduce el concepto de problemas de dominación en grafos, una familia de problemas combinatorios de gran relevancia dentro del análisis de redes. A continuación, se describen las variantes de la familia de problemas de dominación sobre grafos abordados en esta tesis, además de una nueva propuesta metodológica.

2.1. Problemas de dominación en grafos

Los problemas de dominación en grafos constituyen una de las áreas más activas y versátiles dentro de la teoría de grafos, con un amplio abanico de aplicaciones en distintas disciplinas. Esta familia de problemas se basa en la idea general de seleccionar un subconjunto de vértices que ejerza algún tipo de control o cobertura sobre el resto del grafo, teniendo en cuenta distintas restricciones estructurales, operativas o de coste. A este subconjunto de nodos le llamaremos Conjunto Dominante.

A pesar de la simplicidad de su planteamiento original, los problemas de dominación han dado lugar a numerosas variantes que permiten modelar fenómenos complejos en contextos reales. Estos problemas tienen una fuerte motivación práctica. Por ejemplo, surgen de manera natural en el diseño y operación de redes de comunicación, donde es necesario ubicar nodos de control, retransmisión o supervisión de forma que toda la red quede adecuadamente

cubierta. También son útiles en la localización de sensores en redes inalámbricas, en la planificación de servicios públicos (como estaciones de emergencia, centros logísticos o puntos de distribución) y en la propagación de información o influencia en redes sociales. En el marco de la influencia en redes sociales, diversos autores han abordado problemas de maximización de la influencia, cuyo objetivo es identificar el conjunto más pequeño de usuarios capaces de influir en el resto de la red gracias a sus conexiones. Este objetivo está directamente relacionado con la búsqueda de conjuntos dominantes, ya que estos permiten alcanzar a todos los usuarios mediante un modelo de difusión de tipo cascada con un único paso (véase [3, 4, 5]).

Además de su relevancia aplicada, los problemas de dominación son también de gran interés desde el punto de vista teórico y algorítmico. En general, se trata de problemas computacionalmente complejos que han motivado el desarrollo de técnicas avanzadas de optimización, heurísticas y otros algoritmos aproximados. Esta doble vertiente, teórica y práctica, ha consolidado a los problemas de dominación como una herramienta fundamental para el análisis y modelado de sistemas complejos en estructuras de red.

2.2. Grafos

Modificar una heurística en base al estudio de la estructura en la que se presenta la información potencia la capacidad del algoritmo para tomar decisiones más informadas, lo que puede traducirse en una mejora notable en la calidad de las soluciones obtenidas y en una reducción del tiempo computacional necesario para alcanzarlas.

En esta tesis, la estructura en la que se presentan los datos es siempre un grafo, una estructura matemática ampliamente utilizada para modelar relaciones entre objetos. Formalmente, un grafo se define como $G = (V, E)$, donde:

- V es el conjunto de nodos o vértices, que representan los elementos del sistema.
- E es el conjunto de enlaces o aristas, que representan las conexiones entre pares de nodos.

Cada enlace une dos nodos, indicando que existe una relación directa entre ellos. Por ejemplo, en una red social, los nodos podrían representar personas y las aristas representar relaciones de amistad.

Los grafos pueden clasificarse según diferentes características:

- **Dirigidos**: si los enlaces tienen una dirección, es decir, $(u, v) \in E$ implica que hay una conexión de u hacia v, pero no necesariamente de v hacia u. Este tipo de enlaces se conocen comúnmente como arcos.
- **No dirigidos**: si los enlaces son bidireccionales, es decir, $(u, v) \in E$ implica también que $(v, u) \in E$.
- **Ponderados**: si a cada arista se le asigna un peso o valor numérico, que puede representar distancia, coste, tiempo, etc.
- **No ponderados**: si todas las aristas se consideran equivalentes, sin valores asignados, o con un mismo peso asignado a cada una de ellas.

Comprender la estructura y propiedades básicas de un grafo es imprescindible para el correcto entendimiento de los conceptos que se presentan a continuación.

En particular, durante la investigación realizada se ha descubierto que, en la mayoría de problemas de dominación sobre grafos, incluir siempre en la solución los nodos soporte y excluir los nodos hoja supone una gran mejora en tiempo y calidad. Para poder entender por qué esto es así, es necesario introducir el concepto de nodos soporte y nodos hoja.

Definimos *LN* $\subseteq V$ como el conjunto de nodos hoja, es decir, aquellos que están conectados únicamente a un nodo (su grado es igual a 1). Los nodos a los que están conectados los nodos hoja se denominan *nodos soporte*, y se representan como *SN*. En la Figura 2.1 se muestra un ejemplo de esto.

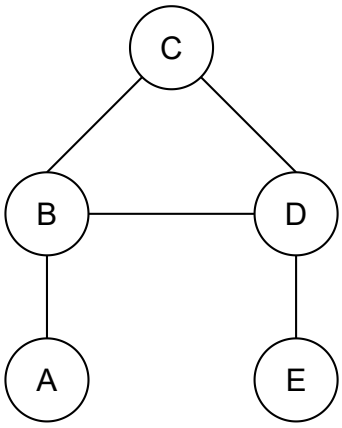


Figura 2.1: Ejemplo de grafo con 5 vértices y 5 aristas.

La Figura 2.1 representa un grafo compuesto por cinco vértices, $V = \{A, B, C, D, E\}$, y cinco aristas, $E = \{(A, B), (B, C), (B, D), (C, D), (D, E)\}$. En este grafo podemos observar dos nodos hoja A y E y sus correspondientes nodos soporte B y D.

Incluir en el Conjunto Dominante un nodo hoja proporciona una solución igual o peor, ya que dominará solamente a su único vecino. Sin embargo, incluyendo en el Conjunto Dominante un nodo soporte, este será capaz de dominar al nodo hoja asociado y al resto de vecinos que tenga. Por tanto, se concluye que toda solución óptima, en grafos sin ponderar y sin restricciones que lo impidan, incluirá necesariamente a los nodos soporte en el Conjunto Dominante.

2.3. Minimum Dominating Set Problem

El *Minimum Dominating Set Problem* (MDSP) constituye una de las variantes fundamentales dentro de la amplia familia de problemas de dominación en grafos. Su formulación básica, que busca determinar el conjunto dominante de menor tamaño posible, sirve como punto de partida para el desarrollo de múltiples extensiones y generalizaciones.

En el MDSP, un Conjunto Dominante en un grafo $G = (V, E)$ es un subconjunto de vértices $D \subseteq V$ tal que todo vértice $v \in V \setminus D$ es adyacente al menos a un vértice en D. Es decir, todos los vértices que no pertenecen al conjunto están conectados directamente con, al menos, un vértice que sí pertenece a él. Más formalmente:

$$\forall v \in V \setminus S, \quad N(v) \cap S \neq \emptyset,$$

donde $N(v)$ representa el conjunto de vecinos del vértice v, es decir, $N(v) = \{u \in V : (u, v) \in E\}$.

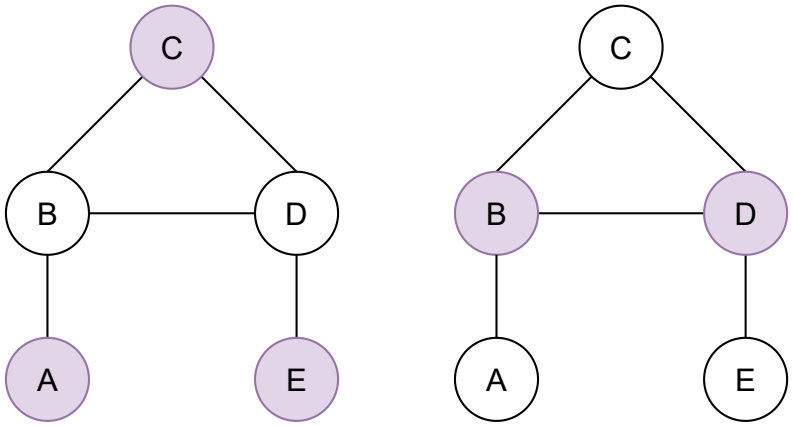
Formalmente, el MDSP puede expresarse como el siguiente problema de optimización:

$$S^\star = \underset{\substack{S \subseteq V \\ \forall v \in V \setminus S,\, N(v) \cap S \neq \emptyset}}{\arg\min} |S| \tag{2.1}$$

donde $S^\star$ representa la solución óptima del problema y $|S|$ es la cardinalidad del conjunto de dominación.

La Figura 2.2 muestra dos ejemplos de soluciones para el grafo representado en la Figura 2.1. En ella se muestran dos tipos de conjuntos dominantes,

donde los nodos seleccionados aparecen resaltados en morado. Por un lado, el conjunto formado por los vértices seleccionados en la solución $S_1 = \{A, C, E\}$, representado en la Figura 2.2a, es una solución factible que representa un Conjunto Dominante pero no es mínimo: existe un conjunto como S_2 que también resuelve el problema con menor tamaño. Por otro lado, la solución $S_2 = \{B, D\}$, mostrado en la Figura 2.2b, es un conjunto mínimo dominante, siendo una solución óptima para este ejemplo. Este conjunto representa la solución con el menor número posible de vértices que cumple la condición de dominación en el grafo.



(a) $S_1 = \{A, C, E\}$. (b) $S_2 = \{B, D\}$.

Figura 2.2: Dos soluciones que suponen un Conjunto Dominante para este grafo ejemplo.

El MDSP ha sido ampliamente estudiado debido a su relevancia práctica, véase [6, 7, 8]. Algunas aplicaciones incluyen la identificación de conjuntos representativos en redes sociales, la cobertura eficiente de áreas geográficas en redes de sensores, o la selección de nodos en redes de comunicación para minimizar costes de infraestructura. La simplicidad del modelo, junto con su aplicabilidad, lo convierten en una herramienta fundamental dentro de la teoría de grafos aplicada.

2.4. Monitor Placement Problem

Podemos definir el objetivo del *Monitor Placement Problem* (MPP) como el de asegurar la supervisión completa de una red, la cual suele representarse

mediante un grafo no dirigido $G = (V, E)$, donde V, con $|V| = n$, representa el conjunto de vértices en la red, y E, con $|E| = m$, el conjunto de aristas que conectan pares de vértices (u, v).

El MPP, constituye una variante aplicada del clásico *Minimum Vertex Cover Problem* (MVCP), en el que se busca cubrir todos los enlaces de una red desplegando el menor número posible de monitores. Sin embargo, más allá de su conexión con los problemas de cubrimiento, el MPP presenta una relación conceptual y estructural con la familia de problemas de dominación en grafos. MPP puede interpretarse como un problema de dominación particular, donde el objetivo no es cubrir vértices, sino cubrir enlaces (aristas).

Una diferencia clave con respecto a los problemas clásicos de dominación radica en la incorporación de prioridades en los enlaces no cubiertos. Esto introduce un componente de optimización bi-criterio en el MPP: por un lado, se minimiza el número de monitores desplegados, y por otro, se minimiza la penalización total asociada a los enlaces descubiertos. En situaciones en las que no sea posible obtener una solución completa dentro del límite de tiempo, el objetivo se redefine hacia la potenciación de la cobertura de enlaces dentro del tiempo disponible. Es decir, se prioriza encontrar una configuración donde los monitores desplegados cubran la mayor cantidad posible de enlaces en la red, aunque dicha cobertura no sea total. En tales casos, no todos los enlaces poseen el mismo nivel de importancia, por lo que resulta fundamental establecer un sistema de prioridades que determine qué enlaces deben ser monitorizados con preferencia [9].

Desde una perspectiva práctica, este tipo de formulaciones resulta especialmente útil en redes de comunicaciones, donde la cobertura total puede no ser alcanzable en tiempo real y debe priorizarse en función de la criticidad de los enlaces. En entornos reales, el margen temporal para ejecutar el MPP suele ser significativamente más reducido que en el caso del MVCP clásico. Esta limitación se justifica en la necesidad de actuar de forma inmediata ante comportamientos anómalos en la red, con el propósito de habilitar o deshabilitar de manera oportuna aquellos vértices que puedan estar generando dichos eventos anómalos. Así, el MPP puede verse como una extensión práctica de los modelos de dominación clásicos, adaptada a entornos dinámicos, jerárquicos y restringidos por recursos.

Una red se considera completamente monitorizada si, y solo si, todos sus enlaces están cubiertos por al menos un monitor. De esta forma, la solución óptima es aquella que garantiza dicha cobertura total, minimizando al mismo tiempo el impacto sobre el rendimiento general del sistema. Así, el MPP se plan-

tea como la búsqueda de un conjunto mínimo de vértices en los cuales ubicar monitores, con el objetivo de supervisar la totalidad de la red. Para simplificar el modelo, se asume que cualquier vértice de la red es apto para albergar un monitor, sin restricciones funcionales o estructurales.

En cuanto al concepto de prioridad asociada a cada enlace, este depende de la naturaleza de la red considerada. Por ejemplo, la importancia de un enlace puede estar determinada por el volumen de tráfico que soporta, su capacidad de transmisión, o su papel estratégico dentro de la topología de la red. La asignación de prioridades, por tanto, no es uniforme, sino que debe basarse en criterios específicos del contexto operativo. Para un análisis detallado sobre metodologías de asignación de prioridades en enlaces de red, consúltese [10, 11].

La función objetivo del MPP está compuesta por dos componentes: el número de monitores desplegados y la penalización acumulada correspondiente a los enlaces no supervisados. Una solución S (con $S \subseteq V$) representa el conjunto de vértices en los que se decide instalar un monitor. Dada una solución S, la evaluación de la función objetivo del MPP se define como:

$$\mathit{MPP}(S) = |S| + \sum_{(u,v)\in E'} p(u,v) \tag{2.2}$$

donde $E' = \{(u,v) \in E : u \notin S \wedge v \notin S\}$ representa el conjunto de aristas de la red que no están cubiertas por ningún monitor y $p(u,v)$ representa la penalización de no supervisar el enlace (u,v).

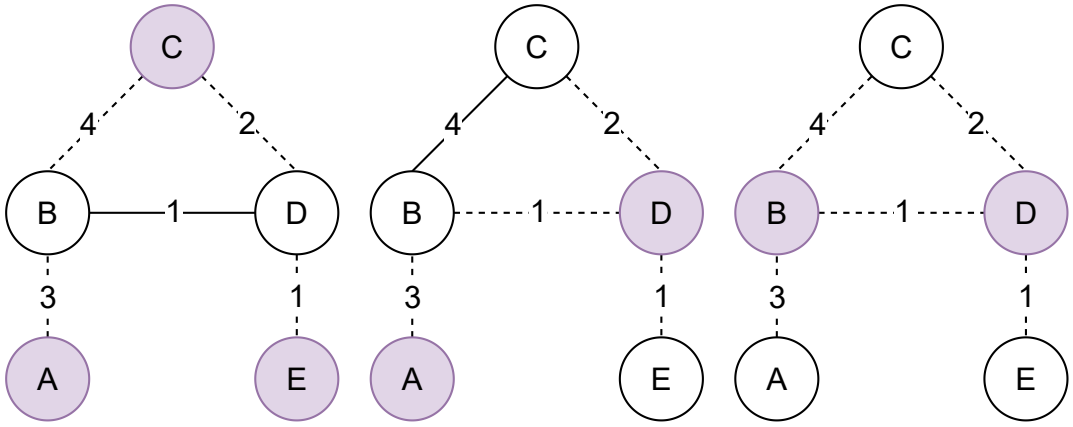
El objetivo del MPP consiste en encontrar la solución $S^\star$ que minimice dicha función:

$$S^\star = \arg\min_{S\in\mathbb{S}} \mathit{MPP}(S) \tag{2.3}$$

siendo $\mathbb{S}$ el espacio de soluciones del problema, es decir, el conjunto de todos los subconjuntos posibles de vértices seleccionados. Cabe señalar que cualquier subconjunto de V constituye una solución factible. En particular, si $|S| = |V|$, se asigna un monitor a cada vértice, lo que implica que el primer término de la función alcanza su valor máximo mientras que el segundo es cero (todos los enlaces están cubiertos). En el extremo opuesto, si $|S| = 0$, ningún monitor es desplegado; entonces, el primer término es cero y el segundo adopta su valor máximo (ningún enlace está cubierto).

La función objetivo del MPP equilibra dos componentes fundamentales: por un lado, la cantidad de monitores instalados en la red, y por otro, la penalización total asociada a los enlaces que permanecen sin supervisión. Aunque pueden considerarse diferentes estrategias para modelar dichas penalizaciones, en este trabajo se adopta el enfoque propuesto en [8], que emplea una función de penalización lineal y estática basada en una medida de distancia (véase la Sección 4.4 para más detalles sobre las penalizaciones empleadas en las instancias analizadas).

En la Figura 2.3 se muestran tres soluciones diferentes al ejemplo representado en la Figura 2.1. En cada una de ellas, los vértices donde se despliega un monitor aparecen resaltados, y las aristas cubiertas por al menos un monitor se indican mediante líneas discontinuas.



(a) $S_1 = \{A, C, E\}$. (b) $S_2 = \{A, D\}$. (c) $S_3 = \{B, D\}$.

Figura 2.3: Tres posibles soluciones para la instancia representada en la Figura 2.1.

En la Figura 2.3a, se muestra la solución S_1, compuesta por el conjunto de vértices $\{A, C, E\}$. Esta selección permite cubrir cuatro de los 5 enlaces de la red, por lo que el valor de la función objetivo es $\mathit{MPP}(S_1) = 3 + 1 = 4$. La Figura 2.3b presenta la solución $S_2 = \{A, D\}$, que pretende ser mejor desplegando únicamente dos monitores. Sin embargo, esta segunda solución deja el enlace (B, C) sin cubrir. En este caso, la penalización asociada a ese enlace es de 4, por lo que el valor de la función objetivo asciende a $\mathit{MPP}(S_2) = 2 + 4 = 6$. Finalmente, la Figura 2.3c muestra la solución $S_3 = \{B, D\}$ que es capaz de monitorizar todos los enlaces de la red, por lo que no tiene ninguna penalización. El valor de la función objetivo en este caso es $\mathit{MPP}(S_3) = 2$.

De la comparación entre estas soluciones se concluye que S_3 es la mejor alternativa, ya que proporciona el menor valor de la función objetivo. Entre las otras dos soluciones, que tienen penalización ya que no cubren todos los enlaces, es importante destacar que, aunque S_1 despliega más monitores que S_2, logra una penalización significativamente menor, resultando en una solución más eficiente. En el contexto del MPP, una solución que logra cubrir todos los enlaces siempre se considera preferible a otra que deja alguno sin supervisar. Sin embargo, debido a restricciones temporales u operativas, no siempre es posible garantizar la cobertura completa de la red.

2.5. Weighted Total Domination Problem

Habiendo introducido previamente el concepto de dominación en grafos, resulta natural extender dicha noción al caso de dominación total. Esta variante impone una condición adicional sobre los vértices del Conjunto Dominante: no sólo deben dominar al resto del grafo, sino también entre ellos mismos.

Sea $G = (V, E)$ un grafo no dirigido. Un conjunto de vértices $S \subseteq V$ se denomina conjunto de dominación total si todo vértice $v \in V$ (incluyendo los vértices de S) tiene al menos un vecino en S, es decir,

$$\forall v \in V, \quad N(v) \cap S \neq \emptyset,$$

donde $N(v)$ denota el conjunto de vecinos de v en G. En otras palabras, cada vértice del grafo es adyacente a al menos un vértice del conjunto S.

A diferencia de la dominación clásica, donde los vértices del Conjunto Dominante no necesitan estar mutuamente conectados, en la dominación total se requiere explícitamente que todos los vértices (incluidos los del propio Conjunto Dominante) estén dominados. Este matiz convierte la dominación total en una versión más restrictiva y, por tanto, más compleja desde el punto de vista computacional.

El *Weighted Total Domination Problem* (WTDP) consiste en encontrar un conjunto de dominación total $S \subseteq V$ tal que se minimice una función objetivo basada en los pesos de los vértices que pertenecen al Conjunto Dominante, las aristas del subgrafo inducido por S y las conexiones necesarias para garantizar la dominación total del grafo. Esta versión ponderada permite modelar situaciones reales donde tanto la selección de nodos como las conexiones entre ellos implican costes específicos.

Consideremos un grafo ponderado en vértices y aristas $G = (V, \omega_V, E, \omega_E)$, donde V y E representan los conjuntos de vértices y aristas, respectivamente. La función de peso en los vértices $\omega_V : V \to \mathbb{R}^+$ asigna un valor positivo real a cada vértice, y la función de peso en las aristas $\omega_E : E \to \mathbb{R}^+$ hace lo propio para cada arista. Es decir, $\omega_V(v)$ y $\omega_E(e)$ representan los pesos asociados al vértice $v \in V$ y a la arista $e \in E$, respectivamente. Para simplificar la notación, usaremos ω en lugar de ω_V y ω_E cuando no haya ambigüedad.

El WTDP consiste en encontrar un conjunto de dominación total $S \subseteq V$. Más formalmente:

$$\textit{WTDP}(S) = \sum_{u \in S} \omega(u) + \sum_{e \in E(S)} \omega(e) + \sum_{v \in V \setminus S} \min \{\omega(v, u) : u \in N(v) \cap S\}, \quad (2.4)$$

donde $E(S)$ representa el conjunto de aristas inducido por los vértices del conjunto S, y $N(v)$ denota el conjunto de vecinos del vértice v.

La función objetivo del WTDP (Ecuación 2.4) suma:

- Peso de los vértices del Conjunto Dominante.
- Peso de las aristas internas (aristas del subgrafo inducido por S).
- Peso mínimo de las aristas externas (el peso de la arista con menor peso que conecta cada nodo no seleccionado con uno del Conjunto Dominante).

En este trabajo se adopta el enfoque de minimizar la suma total de estos tres valores, sin priorizar ninguno de ellos sobre los demás.

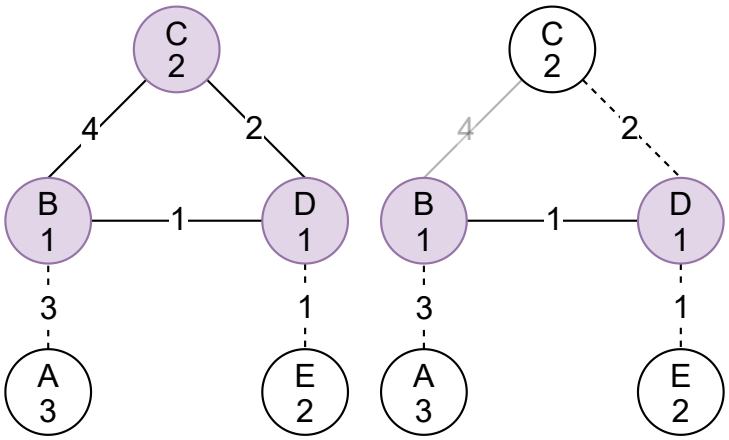
El objetivo del WTDP consiste en encontrar la solución $S^\star$ que minimice dicha función, que formalmente se puede representar como:

$$S^\star = \arg\min_{S \in \mathbb{S}} \textit{WTDP}(S) \quad (2.5)$$

siendo $\mathbb{S}$ el espacio de soluciones del problema, es decir, el conjunto de todos los subconjuntos posibles de vértices seleccionados.

En la Figura 2.4b se presenta una solución de WTDP para la Figura 2.1, donde se han asignado pesos tanto a los vértices como a las aristas. Los pesos de los vértices se muestran dentro de cada nodo debajo de la letra que les identifica, y los pesos de las aristas se indican en las mismas. Las aristas no

involucradas en el cálculo de la función objetivo han sido representadas con menor visibilidad en la figura para facilitar la interpretación. La Figura 2.4a muestra una solución factible $S = \{\mathrm{A}, \mathrm{B}, \mathrm{F}, \mathrm{H}, \mathrm{I}\}$ del WTDP. Los vértices pertenecientes a la solución están resaltados con fondo sólido.



(a) $S_1 = \{B, C, D\}$. (b) $S_2 = \{B, D\}$.

Figura 2.4: Ejemplo de solución del WTDP.

La evaluación de la función objetivo se realiza como:

- $\mathit{WTDP}(S_1) = (1 + 2 + 1) + (4 + 1 + 2) + (3 + 1) = 15$
- $\mathit{WTDP}(S_2) = (1 + 1) + (1) + (3 + 2 + 1) = 9$

El coste total de la S_1 es, por tanto, 15, mientras que el de la S_2 es 9. En el contexto de WTDP, S_2 representa una mejor solución para este grafo ejemplo.

La mayoría de las aplicaciones prácticas del problema de dominación total o de dominación ponderada se pueden modelar a través del WTDP [12, 13, 14]. Por ejemplo, puede emplearse para ubicar un número limitado de dispositivos en una red de comunicación, de forma que cada punto (incluyendo los transmisores) esté adyacente a un dispositivo monitor. El objetivo es asegurar que toda la red esté conectada directamente a un punto monitorizado.

En este contexto, tanto la instalación de dispositivos como la transmisión de datos generan costes. Así, el problema busca minimizar el coste total de la red sin imponer restricciones sobre el número de dispositivos a ubicar. Esta formulación permite obtener soluciones óptimas en términos de coste operativo, algo especialmente relevante en redes donde los gastos de mantenimiento y operación son elevados.

En relación con el ejemplo ilustrado en la Figura 2.4, el grafo representa una red con seis dispositivos y doce conexiones. Para garantizar la seguridad y cobertura de toda la red, se requiere ubicar transmisores de manera que cada dispositivo esté conectado directamente a alguno de ellos. El peso de cada nodo refleja el coste de instalación de un transmisor, mientras que el peso de cada arista representa el coste de transmisión entre dispositivos. La solución mostrada minimiza el coste total necesario para garantizar la cobertura y conectividad de toda la red.

2.6. Parallel Framework for Scatter Search

La mayoría de las metaheurísticas han sido concebidas originalmente como procedimientos secuenciales de exploración del espacio de soluciones. Esta orientación inicial responde al hecho de que, en el momento en que estas técnicas comenzaron a desarrollarse e implementarse en el ámbito de la optimización, el uso de hardware con múltiples procesadores no era común. Sin embargo, en la actualidad, la mayoría de los dispositivos modernos están equipados con arquitecturas multiprocesador.

A pesar de esta evolución tecnológica, muchos algoritmos siguen funcionando de forma secuencial, lo que impide aprovechar plenamente el potencial del hardware disponible. Esta falta de paralelismo inherente puede traducirse en una utilización ineficiente de los recursos computacionales y, en consecuencia, en un rendimiento subóptimo.

Diseñar procedimientos de búsqueda que exploten eficazmente el paralelismo no es una tarea trivial. La coordinación y sincronización entre procesos en un entorno multiprocesador puede ser compleja y propensa a errores, generando situaciones como condiciones de carrera, interbloqueos (*deadlocks*) o dificultades en el mantenimiento y depuración del software. Además, la sobrecarga derivada del reparto de tareas y la coordinación entre procesos puede llegar a contrarrestar las ventajas del paralelismo, especialmente en programas pequeños o con baja carga computacional.

En este contexto, los avances en tecnologías metaheurísticas han impulsado el desarrollo de diseños paralelos con el objetivo de reducir el esfuerzo computacional o ampliar la capacidad de exploración del espacio de soluciones [15, 16, 17]. El objetivo último de toda metodología de resolución es ofrecer un marco general de implementación que sea adaptable a problemas concretos. Esta es precisamente la motivación principal de diseñar *Parallel Framework for*

Scatter Search (PFSS), que se propone como avance metodológico de esta Tesis Doctoral.

La estructura propia de SS lo convierte en un enfoque particularmente adecuado para su paralelización, tal como han evidenciado algunos trabajos recientes [18, 19, 20]. Se han propuesto incluso variantes generales de SS adaptadas a clases específicas de problemas, como la optimización multiobjetivo [21]. Asimismo, existen versiones paralelas de SS enfocadas a problemas concretos de optimización [22]. No obstante, hasta donde se tiene conocimiento, no se ha abordado la formulación de un conjunto de directrices generales que permitan implementar SS en entornos paralelos desde una perspectiva amplia, sin ceñirse a un único problema específico.

Por tanto, el principal objetivo de la propuesta metodológica PFSS es establecer un marco común y un conjunto de pautas que faciliten el diseño de implementaciones paralelas de SS aplicables a una amplia gama de problemas de optimización combinatoria.

El diseño de algoritmos paralelos puede orientarse a diferentes fines: reducir el esfuerzo computacional necesario o ampliar la región del espacio de soluciones que se explora. El primer enfoque consiste en distribuir las tareas computacionales entre varios procesadores, con el propósito de obtener los mismos resultados que un algoritmo secuencial en menor tiempo. El segundo enfoque emplea los procesadores disponibles para explorar nuevas regiones del espacio de búsqueda, con el fin de encontrar mejores soluciones dentro del mismo tiempo límite que tendría una versión secuencial.

Existen propuestas recientes en la literatura que ilustran ambos enfoques, como por ejemplo un algoritmo de búsqueda local diseñado para ejecutarse sobre CUDA en el contexto del problema de la mochila por unión de conjuntos [23], o un algoritmo de optimización aritmética binaria paralelo orientado a la selección de características [24].

PFSS propone tres versiones paralelas distintas de SS, cada una de ellas orientada a uno o ambos objetivos mencionados anteriormente.

Cabe subrayar que el objetivo de esta investigación no es alcanzar el nuevo estado del arte en los problemas tratados, sino proporcionar a la comunidad científica una estructura general para paralelizar SS, que sea fácilmente adaptable a cualquier problema de optimización combinatoria.

Capítulo 3

Metodología

Los algoritmos metaheurísticos se han consolidado como herramientas eficaces para abordar problemas complejos de optimización combinatoria. En numerosas situaciones del mundo real, las metaheurísticas ofrecen una alternativa eficiente capaz de proporcionar soluciones de alta calidad en tiempos de cómputo reducidos. Entre los métodos más representativos dentro de esta categoría se encuentran IG, GRASP, VNS y SS, que son los protagonistas de este capítulo.

3.1. Metaheurísticas

Una metaheurística es *"un marco algorítmico de alto nivel e independiente del problema que proporciona un conjunto de directrices o estrategias para desarrollar algoritmos heurísticos de optimización"* [25]. A diferencia de las heurísticas específicas, las metaheurísticas son independientes del dominio y están diseñadas para escapar de óptimos locales, equilibrando exploración y explotación mediante el uso de mecanismos como la aleatoriedad, la memoria o la cooperación entre soluciones.

En las siguientes secciones se describen únicamente aquellas heurísticas y metaheurísticas que han sido empleadas en el desarrollo de este trabajo de Tesis Doctoral. La selección de estas técnicas responde a su adecuación al problema abordado y a su eficacia comprobada en escenarios similares en la literatura. Si bien existe una amplia variedad de enfoques en el ámbito de la optimización, esta exposición se limita exclusivamente a aquellos métodos que

han sido implementados, analizados y validados en el contexto específico de esta investigación.

3.2. Heurísticas diseñadas

Como ya se ha mencionado en la Sección 1.1, en el ámbito de la optimización, una heurística es una técnica de resolución de problemas que busca obtener soluciones satisfactorias en tiempos de cómputo razonables, especialmente en problemas donde encontrar la solución óptima resulta inviable desde el punto de vista computacional. A diferencia de los algoritmos exactos, que garantizan la obtención de una solución óptima, las heurísticas no ofrecen dicha garantía, pero permiten alcanzar soluciones de buena calidad con un coste computacional mucho menor. Estas técnicas se basan en reglas empíricas o conocimiento específico del problema para guiar la búsqueda de soluciones viables dentro del espacio de soluciones.

En esta Tesis Doctoral se han desarrollado y analizado diversas heurísticas aplicadas a problemas de dominación sobre grafos. Concretamente, se han propuesto diferentes procedimientos constructivos y métodos de búsqueda local, tanto independientes como integrados dentro de metaheurísticas. Estas técnicas forman parte fundamental del proceso de resolución y constituyen la base para el diseño eficiente de los algoritmos planteados.

Las próximas secciones están dedicadas a describir, por un lado, los procedimientos constructivos diseñados para generar soluciones iniciales (Sección 3.2.1) y, por otro, las estrategias de búsqueda local (Seccion 3.2.2) propuestas para mejorar dichas soluciones.

3.2.1. Constructivos voraces

Los constructivos voraces representan una de las estrategias más utilizadas para generar soluciones iniciales en problemas de optimización combinatoria. Su funcionamiento se basa en la construcción progresiva de una solución, seleccionando en cada paso el elemento que parece más prometedor según un criterio heurístico, denominado función voraz. Esta elección, aunque no garantiza encontrar la solución óptima, permite obtener soluciones factibles de calidad razonable en tiempos de cómputo reducidos, lo cual resulta fundamental en

problemas complejos o de gran tamaño, ya que sirve como solución inicial para procedimientos como VNS o IG.

Los distintos métodos propuestos se diferencian tanto en el enfoque adoptado (constructivo o destructivo) como en el diseño de la función voraz utilizada para guiar la construcción de la solución.

Por un lado, se han planteado dos enfoques basados en procedimientos voraces adaptados a la estructura del grafo. El primero, parte de una solución parcial formada únicamente por los nodos soporte y va añadiendo iterativamente los vértices que más contribuyen a dominar nuevos nodos, según una función voraz que mide el número de nodos aún no dominados que quedarán cubiertos tras su inclusión. Este procedimiento termina cuando la solución es un conjunto dominante. El segundo, parte de una solución que contiene todos los nodos salvo los nodos hoja y elimina, de manera iterativa, los nodos cuya retirada tenga menor impacto. Esto se evalúa gracias a la función objetivo diseñada, que calcula el nodo que tiene al vecino con un menor número de vecinos en el conjunto dominante. El nodo eliminado será aquel que maximice este valor, tratando de asegurar que el hecho de eliminar al nodo del conjunto dominante afecta lo mínimo posible al grafo.

Por otro lado, se han desarrollado dos procedimientos constructivos generales, sin asumir conocimiento específico del grafo. El primero, sigue el esquema clásico de construcción voraz, comenzando desde una solución vacía y seleccionando en cada iteración el nodo que permite minimizar la penalización asociada a los enlaces aún no cubiertos. El segundo, parte de una solución completa con todos los nodos seleccionados, pero sin tener en cuenta los nodos hoja, y elimina iterativamente los nodos menos relevantes (aquellos con menor grado) siempre que la cobertura de enlaces no se vea comprometida.

Estos enfoques constructivos permiten, no solo generar soluciones iniciales válidas y prometedoras para las metaheurísticas, sino también estudiar el impacto del conocimiento estructural del grafo y del diseño de funciones voraces específicas en la calidad de las soluciones obtenidas.

3.2.2. Búsquedas locales

La búsqueda local es una técnica ampliamente utilizada en el ámbito de la optimización combinatoria para mejorar soluciones previamente generadas, generalmente por procedimientos constructivos. Su principio fundamental consiste en explorar el vecindario de una solución actual mediante la aplicación de

operadores de movimiento, con el objetivo de encontrar soluciones de mejor calidad dentro de una región cercana del espacio de búsqueda.

A diferencia de los métodos constructivos, que parten de una solución vacía o parcial y la van completando, las búsquedas locales comienzan con una solución factible y aplican transformaciones sobre ella, manteniendo la factibilidad en cada paso. Esta estrategia permite explotar regiones prometedoras del espacio de soluciones, mejorando progresivamente la calidad.

La definición de una búsqueda local se apoya en tres elementos esenciales: el operador de movimiento (que define cómo se transforma una solución en otra), el vecindario inducido por ese operador (conjunto de soluciones alcanzables mediante un solo movimiento), y la estrategia de exploración del vecindario (por ejemplo, *Best improvement* o *First improvement*). Estos componentes deben ser cuidadosamente diseñados en función de la naturaleza del problema abordado y del equilibrio deseado entre intensificación y eficiencia computacional.

Todas las búsquedas locales implementadas en los trabajos de esta Tesis Doctoral tienen en común la estrategia de exploración del vecindario, siendo esta *First improvement*. Se trata de una estrategia de exploración del vecindario en la que se recorre el conjunto de soluciones vecinas y, si la solución resultante mejora con respecto a la solución original, se actualiza la mejor solución encontrada. Esta estrategia evita evaluar todas las soluciones del vecindario en cada iteración, lo que permite reducir significativamente el tiempo de cómputo frente a otras estrategias más exhaustivas como *Best improvement*, en la que se explora completamente el vecindario para seleccionar la mejor solución. Aunque *First improvement* puede no encontrar siempre la mejor mejora posible, suele conducir a resultados de calidad comparable en un tiempo computacional significativamente menor, especialmente si el orden de exploración del vecindario se aleatoriza para favorecer la diversidad [26].

Clasificando las búsquedas locales diseñadas por su operador de movimiento encontramos dos distintas en los trabajos de la Tesis Doctoral:

- *Exchange1x1*: Este operador de movimiento se trata del intercambio simple (intercambiar un nodo que está en la solución por uno que no lo está). La exploración se realiza de forma aleatoria, y se consideran dos optimizaciones adicionales de esta búsqueda local: una predicción del valor de la función objetivo sin realizar el movimiento y una reducción del vecindario a aquellos movimientos que garantizan la factibilidad. Ambas optimizaciones buscan reducir el tiempo de cómputo sin afectar negativamente a la calidad de la solución. En esta propuesta, al intercambiar un nodo que es-

tá dentro de la solución con uno que está fuera, puede llevar a pensar que no se está realizando ningún cambio sobre la solución, siendo la función objetivo el número de elementos que hay en la misma. Sin embargo, tras cada movimiento se aplica un proceso llamado *clean*, que es capaz de eliminar aquellos nodos que son redundantes en la solución, es decir, que pueden ser eliminados y la solución conserva la factibilidad.

- *Exchange2x1*: Operador de intercambio más complejo, que elimina dos nodos de la solución e inserta uno nuevo, buscando minimizar la cantidad de nodos que forma el conjunto dominante. Se han desarrollado dos estrategias distintas: una búsqueda local exhaustiva, que explora de forma aleatoria todo el vecindario, y una búsqueda local inteligente, que selecciona cuidadosamente los nodos a eliminar e insertar basándose en propiedades estructurales del grafo. Esta última estrategia permite explorar movimientos prometedores con menor esfuerzo computacional.

En conjunto, estas estrategias muestran un compromiso entre calidad de la solución y eficiencia computacional, ajustando la complejidad de la búsqueda local a las características del problema y a los objetivos de cada metaheurística.

3.3. Greedy Randomized Adaptive Search Procedure

La metodología *Greedy Randomized Adaptive Search Procedure* (GRASP) es una metaheurística multiarranque presentada en [27], aunque no fue formalmente definida hasta [28]. Consta de dos fases principales: una fase de construcción y otra de mejora. La primera tiene como objetivo generar soluciones de alta calidad y diversidad desde cero [29], y en la definición de la metaheurística se describe de forma precisa el procedimiento, siendo un pilar fundamental de la misma. Por otro lado, la segunda fase, que tiene como objetivo encontrar un óptimo local, se trata de un procedimiento de mejora en el que se puede usar desde una búsqueda local simple hasta un procedimiento más elaborado como *Tabu Search*.

La idea de GRASP radica en introducir aleatoriedad durante la fase de construcción para incrementar la diversidad de la búsqueda. A diferencia de una construcción completamente voraz, cada iteración de GRASP produce una solución distinta, lo que permite explorar diferentes regiones del espacio de búsqueda. El pseudocódigo del esquema GRASP se presenta en el Algoritmo 1.

Algorithm 1 $\mathrm{GRASP}(I, \alpha, \Delta)$

```
1: S_b ← ∅
2: for i ∈ 1 ... Δ do
3:     S ← Construccion(I, α)
4:     S' ← Mejora(S)
5:     if EsMejor(S', S_b) then
6:         S_b ← S'
7:     end if
8: end for
9: return S_b
```

El método requiere tres parámetros de entrada: I, la instancia del problema a resolver; α, que controla el equilibrio entre aleatoriedad y voracidad en la construcción; y Δ, el número de iteraciones de GRASP.

El algoritmo construye e intenta mejorar Δ soluciones (líneas 2-8). En cada iteración, se construye una solución desde cero (línea 3). Luego, se aplica una mejora para encontrar una solución de mayor calidad (línea 4). Si la nueva solución es mejor que la mejor encontrada hasta el momento (línea 5), se actualiza (línea 6). El proceso termina devolviendo la mejor solución encontrada (línea 9).

3.3.1. Fase de construcción

Tal y como se ha mencionado previamente, en el contexto de GRASP, la fase de construcción es responsable de generar soluciones iniciales que sirvan como punto de partida para la posterior mejora. Existen diversos enfoques para diseñar esta fase, cada uno con un equilibrio diferente entre exploración (diversidad) y explotación (calidad). Es importante mencionar que en la construcción de GRASP el primer elemento de la solución se selecciona siempre de forma aleatoria.

Una característica fundamental de la fase de construcción en GRASP es la selección aleatoria del primer elemento de la solución. Esta decisión responde a un objetivo central de la metaheurística: garantizar una adecuada diversificación del espacio de búsqueda. La inclusión de aleatoriedad desde el inicio permite generar múltiples trayectorias constructivas diferentes entre iteraciones del algoritmo. Si se seleccionara siempre el mejor elemento según un criterio voraz en la primera posición, todas las soluciones generadas comenzarían su

construcción desde el mismo punto, reduciendo así la variedad de soluciones exploradas.

Uno de los enfoques más usados es el modelo Constructivo*Greedy Random* (CGR), el cual utiliza una lista de candidatos restringida *Restricted Candidate List* (RCL) construida a partir de una función voraz. Esta lista incluye los elementos cuyo valor están por debajo de un umbral calculado de la siguiente forma:

$$\mu = g_{\text{min}} + \alpha_{GR} \cdot (g_{\text{max}} - g_{\text{min}}) \tag{3.1}$$

donde α_{GR} se trata de un parámetro que controla el grado de aleatoriedad (es decir, el equivalente a α en la propuesta orginal de GRASP) . Un valor bajo de α_{GR} genera una construcción más voraz, mientras que un valor alto introduce mayor diversidad aleatoria. Si $\alpha_{\text{GR}} = 0$, la selección es totalmente voraz, ya que, en ese caso, los únicos candidatos que forman la RCL son los candidatos que tienen como valor de la función voraz g_{min}. Por otro lado, si $\alpha_{\text{GR}} = 1$, los candidatos a formar parte de la RCL son todos los elementos disponibles, ya que $\mu = g_{max}$, por lo que estaríamos hablando de una selección de candidatos totalmente aleatoria.

Nótese que esta será la forma y el procedimiento a seguir en caso de que el objetivo sea el de minimizar el valor de la función voraz. En cada iteración, se selecciona al azar entre los candidatos de la RCL un elemento de la RCL para ser añadido a la solución.

La Figura 3.1 representa la selección del siguiente candidato a formar parte de la solución siguiendo el enfoque CGR. Los elementos que forman parte de la RCL en cada caso tienen el fondo morado. Por otro lado, el valor de la función voraz al introducir cada elemento se encuentra dentro del mismo. De esta forma, el valor de $g_{min} = 3$, mientras que el de $g_{max} = 16$. Las tres listas de candidatos a formar parte de la solución están formadas por los mismos elementos pero en escenarios distintos ya que el conjunto de valores tomará $\alpha_{GR} = \{0.25, 0.5, 0.75\}$. El valor del umbral μ se marca con una línea que corta la lista de elementos y ha sido calculado siguiendo la fórmula previamente descrita en Ecuación 3.1. Siendo así, poniendo como ejemplo cuando $\alpha_{GR} = 0.25$, la fórmula sustituyendo valores sería: $\mu = 3 + 0.25 \cdot (16 - 3)$. El valor obtenido al resolver las operaciones es $\mu = 6.25$. Los elementos que formarán parte de la RCL serán aquellos que tengan un valor de la función voraz igual o inferior al umbral μ en un escenario en que se busca minimizar el valor de la función voraz. Finalmente, el candidato seleccionado entre los que forman la RCL se selecciona de forma aleatoria, resultando en este caso en aquel con valor de la función voraz de 6, marcado en la figura como el candidato con borde de doble línea. Esta lógica se sigue con los otros dos ejemplos de listas de candidatos en

los que cambia el valor de α_{GR} y, por tanto, el valor del umbral. Esto repercute directamente en los candidatos que formarán parte de la RCL.

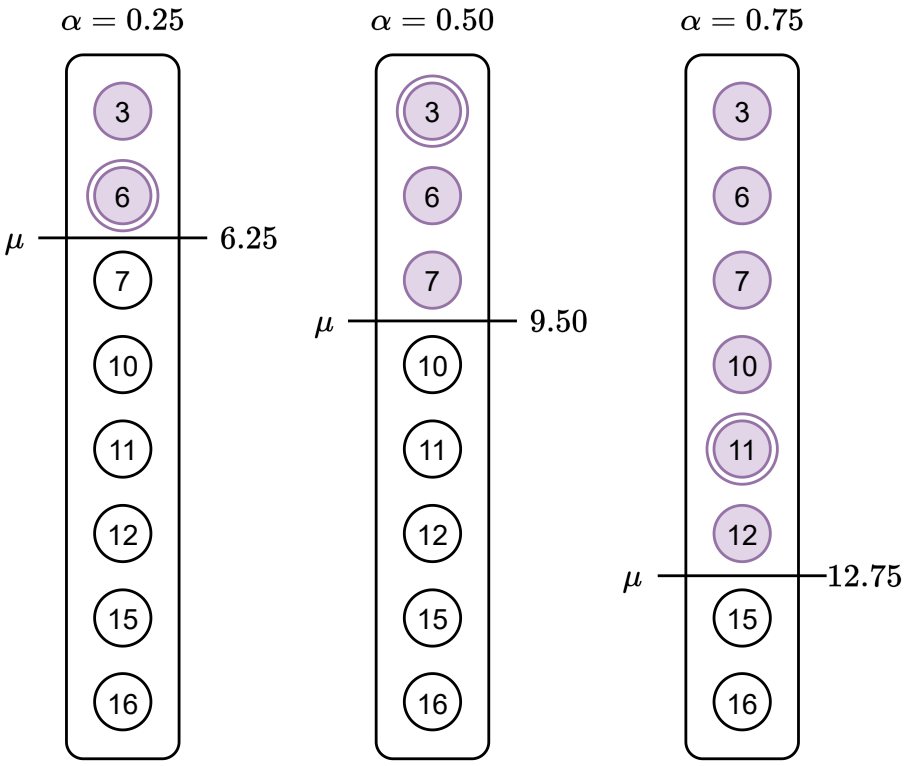


Figura 3.1: Representación del funcionamiento la RCL en la construcción CGR de GRASP.

Por otro lado, el modelo Constructivo*Random Greedy* (CRG) invierte las fases del esquema tradicional de GRASP. En este caso, se selecciona aleatoriamente un subconjunto de candidatos de tamaño proporcional a un parámetro α_{RG} y se introduce en la RCL. Posteriormente, se elige el mejor elemento de la RCL según la función voraz. Esta variante ofrece una manera alternativa de combinar aleatoriedad y criterio voraz, favoreciendo la exploración al inicio y una selección dirigida posteriormente.

3.3.2. Biased GRASP

GRASP ha sido objeto de diversas extensiones con el objetivo de mejorar su rendimiento y adaptabilidad a distintos problemas combinatorios. Además de

la versión original, existen otras como *Reactive* GRASP [30], que introduce una adaptación dinámica del parámetro α en función de los resultados obtenidos durante la búsqueda, permitiendo al algoritmo ajustar su comportamiento entre exploración y explotación.

Por otra parte, existe otra variante llamada *Biased Randomized* GRASP que reemplaza la selección uniforme de los candidatos que forman parte de la RCL por una selección sesgada [31], utilizando distribuciones probabilísticas que favorecen elementos más prometedores, incrementando así la eficiencia de la búsqueda sin perder diversidad.

Tanto GRASP estándar como sus variantes son procedimientos sin memoria, ya que no consideran información de iteraciones anteriores. Sin embargo, conservar información de iteraciones anteriores podría mejorar la selección durante la construcción de nuevas soluciones.

3.4. Variable Neighborhood Search

Una de las estrategias metaheurísticas más relevantes en la última década para la resolución de problemas de optimización combinatoria es el enfoque conocido como *Variable Neighborhood Search* (VNS), que fue definido originalmente en [32]. Esta metodología se ha consolidado como una alternativa eficaz y versátil gracias a su capacidad para equilibrar la intensificación y la diversificación en el proceso de búsqueda. A diferencia de otras técnicas que exploran una vecindad fija, VNS se basa en la idea de modificar sistemáticamente la estructura de la vecindad durante la exploración del espacio de soluciones, lo que permite escapar de óptimos locales y alcanzar soluciones de mayor calidad. Su simplicidad conceptual, combinada con una sólida eficacia práctica, ha permitido que VNS se aplique exitosamente en una amplia gama de problemas, tanto académicos como del mundo real. Las vecindades pueden ser exploradas mediante tres estrategias: determinista (*Variable Neighborhood Descent*), estocástica (*Reduced VNS*) o mixta (*Basic VNS*). También se han propuesto otras variantes, como *General VNS*, *Variable Neighborhood Decomposition Search*, *Variable Formulation Search*, entre otros. Para una descripción más detallada, véase [33].

Para poder entender cómo funcionan esta metaheurística y sus variantes, debemos primero definir el esquema básico de VNS *Basic Variable Neighborhood Search* (BVNS), que combina la intensificación proporcionada por la mejora con la diversificación introducida por la fase *Shake*. Encontrar un equi-

librio entre ambas estrategias es clave para alcanzar soluciones de calidad. El Algoritmo 2 muestra el pseudocódigo del esquema propuesto.

Algorithm 2 $\mathrm{BVNS}(S, k_{\max}, \delta)$

```
1:  for i ∈ 1...δ do
2:      k ← 1
3:      while k ≤ k_max do
4:          S' ← Shake(S, k)
5:          S'' ← Mejora(S')
6:          if EsMejor(S'', S) then
7:              k ← 1
8:              S ← S''
9:          else
10:             k ← k + 1
11:         end if
12:     end while
13: end for
14: return S
```

El método requiere tres parámetros de entrada: una solución inicial S generada previamente, la vecindad más lejana a explorar $k_{\max}$ y el número máximo de iteraciones permitidas δ. BVNS suele ejecutarse un número fijo de iteraciones o durante un tiempo de cómputo determinado. En este caso, se establece un número fijo de iteraciones δ (líneas 1-13). En cada iteración, el algoritmo comienza con el primer entorno (línea 2) y continúa hasta alcanzar el entorno $k_{\max}$ (líneas 3-12). Para cada entorno, se genera una solución perturbada mediante el procedimiento *Shake* en la línea 4. Esta solución S' es posteriormente mejorada a través de la fase de mejora, generando la solución S'' (línea 5). Finalmente, BVNS lleva a cabo la fase de cambio de entorno: si la solución mejorada S'' mejora a la mejor solución encontrada hasta el momento (línea 6), el algoritmo reinicia el proceso desde el primer entorno (línea 7) y actualiza la mejor solución (línea 8). En caso contrario, se incrementa el índice de entorno para explorar uno diferente (línea 10). El algoritmo finaliza tras completar δ iteraciones, devolviendo la mejor solución hallada (línea 14).

3.4.1. Jump VNS

Jump Variable Neighborhood Search (JVNS), es una variante que combina la búsqueda determinista del procedimiento de mejora con la exploración

aleatoria de la fase de (*Shake*). El objetivo es equilibrar intensificación y diversificación. *Jump VNS* fue originalmente presentado en [34] como una extensión del esquema BVNS. La diferencia principal entre JVNS y BVNS es la manera de realizar el cambio de vecindad. Mientras que el BVNS explora secuencialmente las vecindades desde $k = 1$ hasta $k_{\max}$ incrementando una unidad en cada paso, *Jump VNS* cambia de vecindades de forma más radical. Este enfoque es útil cuando el cambio de BVNS es insuficiente para escapar de óptimos locales, debido a que las vecindades que están cerca entre sí tienden a generar soluciones similares.

Para mejorar la eficiencia y evitar la exploración de soluciones equivalentes, JVNS introduce un nuevo parámetro de entrada k_{step}, que controla el incremento de la variable k para cambiar de vecindad. El pseudocódigo del procedimiento JVNS se muestra en el Algoritmo 3.

Algorithm 3 $\mathrm{JVNS}(S, k_{max}, k_{step}, \Delta)$

```
1:  for i ∈ 1...Δ do
2:      k ← k_step
3:      while k ≤ k_max do
4:          S' ← Shake(S_b, k)
5:          S'' ← Mejora(S')
6:          if EsMejor(S'', S) then
7:              S ← S''
8:              k ← k_step
9:          else
10:             k ← k + k_step
11:         end if
12:     end while
13: end for
14: return S
```

El método requiere cuatro parámetros de entrada: la solución inicial S; la vecindad más lejana a explorar $k_{\max}$; el salto entre vecindades k_{step}; y el número de iteraciones del VNS ejecutadas, Δ. El procedimiento realiza Δ iteraciones completas de JVNS (líneas 1-13). En cada iteración, se inicializa la vecindad actual con el valor k_{step}, y se exploran sucesivas vecindades hasta alcanzar $k_{\max}$ (líneas 3-12). Para cada vecindad, se aplica el procedimiento *Shake* (línea 4), encargado de introducir diversidad en la búsqueda, generando una nueva solución S'. Como esta solución no tiene por qué ser un óptimo local, se aplica un método de mejora, obteniendo una solución mejorada S'' (línea 5).

Luego, se evalúa si S'' mejora la mejor solución encontrada hasta el mo-

mento. En caso afirmativo (línea 6), se actualiza S (línea 7) y se reinicia la vecindad al valor inicial k_{step} (línea 8). Si no hay mejora, se cambia de vecindad en pasos de k_{step} (línea 10). Al finalizar las Δ iteraciones, el algoritmo devuelve la mejor solución obtenida S (línea 14).

3.4.2. Multi-start VNS

En el marco de VNS, el procedimiento *Shake* introduce cambios aleatorios en la solución actual, lo que permite al algoritmo escapar de óptimos locales y explorar diferentes regiones del espacio de soluciones. La versión *multi-start* de VNS extiende esta idea mediante la incorporación de múltiples soluciones iniciales. En lugar de partir de una única solución, se generan diversas soluciones iniciales que se consideran de forma independiente, ejecutando un procedimiento VNS para cada una de ellas.

La motivación principal detrás de la estrategia *Multi-start VNS* es evitar el estancamiento en óptimos locales, promoviendo una exploración más amplia del espacio de búsqueda. No obstante, cabe destacar que un algoritmo VNS con una fase de *Shake* mejorada puede alcanzar un grado de exploración comparable al de la versión multi-start. Un diseño eficaz del procedimiento de *Shake* permite al algoritmo incrementar su capacidad de diversificación y, en consecuencia, identificar soluciones de alta calidad.

Sin embargo, la elección entre la versión estándar de VNS y su enfoque *multi-start* sigue siendo objeto de debate. La eficacia de cada uno depende de diversos factores, como la naturaleza del problema, las instancias evaluadas o la calidad de las soluciones iniciales generadas. Por tanto, la decisión entre VNS y *Multi-start* VNS continúa siendo discutida dentro de la comunidad de optimización.

3.4.3. Shake

La fase de *Shake* es determinante en el algoritmo VNS y tiene como objetivo permitir al algoritmo escapar de óptimos locales cuando la fase de mejora se estanca. Este componente introduce diversidad en la búsqueda al cambiar la vecindad explorada. Para ello, se emplea un parámetro k que controla la magnitud del cambio de vecindad. Dicho parámetro se incrementa cuando no se encuentran mejoras, con el fin de ampliar la exploración hacia regiones más alejadas del espacio de soluciones en el que se está buscando; y se reduce al

hallar mejoras, permitiendo así intensificar la búsqueda en regiones prometedoras.

Los cambios realizados sobre la solución para cambiar la vecindad de exploración pueden seguir un criterio heurístico (*Intensified Shake*) o totalmente aleatorio (*Random Shake*), y los movimientos pueden ser desde un movimiento de intercambio hasta un movimiento de adición o de eliminación; esto dependerá del problema abordado. Nótese que seleccionar correctamente el movimiento para explorar distintas vecindades de acuerdo a las necesidades de cada problema que se trata puede ser vital en el buen funcionamiento de la fase *Shake*.

Intensified Shake es una estrategia que, en lugar de seleccionar elementos de la solución aleatoriamente para modificar la solución actual (como hace el enfoque clásico), selecciona inteligentemente los elementos más prometedores con base en un criterio heurístico. El objetivo es dirigir la búsqueda hacia regiones del espacio de soluciones que se consideran más prometedoras, sin perder completamente la capacidad de explorar nuevas soluciones.

3.5. Iterated Greedy

Iterated Greedy (IG) es una metaheurística propuesta originalmente para resolver problemas de planificación de tareas [35] y que desde entonces ha evolucionado, demostrando su eficacia en la resolución de una amplia gama de problemas difíciles de optimización combinatoria. El éxito de IG radica en una idea sencilla pero efectiva: destruir parcialmente la solución e intentar reconstruirla, permitiendo escapar de óptimos locales.

IG necesita una solución inicial desde la cual comenzar la búsqueda. Aunque la literatura sugiere que incluso una solución generada de manera aleatoria puede ser suficiente para que IG proporcione soluciones de alta calidad, investigaciones recientes han demostrado que iniciar desde una región prometedora del espacio de búsqueda tiende a mejorar el rendimiento de la mayoría de los algoritmos metaheurísticos. Posteriormente esta solución será transformada destruyéndola y reconstruyéndola de forma parcial para llegar a otras vecindades de búsqueda. El Algoritmo 4 presenta el pseudocódigo general del marco IG.

El algoritmo recibe tres parámetros de entrada: S, una solución que representa un óptimo local; β, el porcentaje de elementos eliminados durante la fase de destrucción; y Δ, el número máximo de iteraciones permitidas sin mejora.

Algorithm 4 IG(S, β, Δ)

```
1:  δ ← 0
2:  while δ < Δ do
3:      S' ← Destruccion(S, β)
4:      S'' ← Reconstruccion(S')
5:      S''' ← Mejora(S'')
6:      if EsMejor(S''', S) then
7:          S ← S'''
8:          δ ← 0
9:      else
10:         δ = δ + 1
11:     end if
12: end while
13: return S
```

IG comienza generando una solución inicial. Posteriormente, se aplica una mejora local sobre dicha solución inicial, obteniéndose la mejor solución encontrada hasta el momento, y que se pasa al algoritmo como parámetro de entrada S. El algoritmo continúa iterando hasta alcanzar el criterio de parada (líneas 2–12). Un posible criterio de parada podría ser el representado en el pseudocódigo, donde se define como un número máximo de iteraciones sin encontrar una solución mejor, controlado por el parámetro Δ, o alguno diferente como, por ejemplo, un tiempo máximo de ejecución.

En cada iteración, la solución actual S es parcialmente destruida, generando una solución S' (línea 3). Posteriormente, la factibilidad se restablece mediante un procedimiento de reconstrucción, obteniéndose una nueva solución S'' (línea 4). Esta solución no es necesariamente un óptimo local, por lo que se vuelve a aplicar una fase de mejora, obteniendo una nueva solucion S''' (línea 5). Al final de cada iteración, se compara si la nueva solución generada S''' es mejor que la que ya teníamos S (línea 6). Si S''' mejora la mejor solución conocida, se actualiza S (línea 7) y se reinicia el contador de iteraciones sin mejora (línea 8); en caso contrario, dicho contador se incrementa (línea 10). El proceso finaliza cuando se alcanza el criterio de parada, devolviendo la mejor solución encontrada en el procedimiento (línea 13).

Para representar el funcionamiento de las siguientes fases que forman parte de IG se ha diseñado la Figura 3.2. Los elementos que forman parte de la solución están representados como círculos con color morado, frente a los elementos que no forman parte de la misma, los que no tienen ningún color asociado. Los elementos representados con la línea punteada representan ele-

mentos que en la fase anterior formaban parte de la solución pero han sido eliminados.

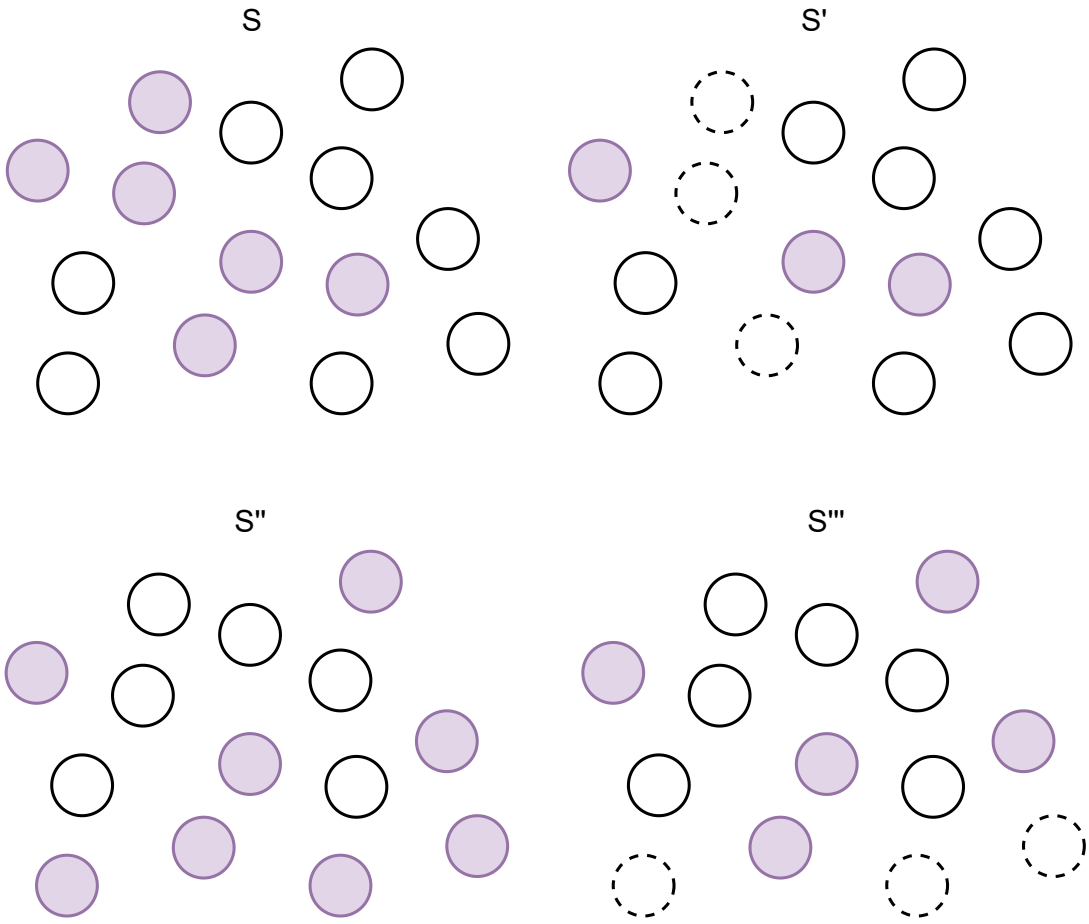


Figura 3.2: Representación de la evolución de la solución a medida que se aplican las distintas fases de *Iterated Greedy* sobre ella.

En primer lugar, IG recibe una solución que contiene 6 elementos S. Al comenzar con el procedimiento, se aplica la fase de destrucción sobre S, obteniendo la solución representada como S'. Este nuevo conjunto de elementos representa una solución no factible, por lo que se debe que reconstruir. El resultado de reconstruir la solución es S'', que contiene un total de 8 elementos. Por último, a esta solución se le aplica una mejora, resultando en S''', formada por un total de 5 elementos y que es factible. Este ejemplo muestra un caso en el que la solución consigue mejorar tras la iteración de IG, por lo que δ volvería a tomar el valor 0.

3.5.1. Fase de destrucción

La fase de destrucción del algoritmo IG constituye una de las fases responsables de diversificar la búsqueda, al eliminar una parte de la solución. Esta destrucción se debe adaptar a los términos del problema, ya que dependiendo de la forma de representar la solución puede conllevar: eliminar elementos de la solución, cambiar la etiqueta de elementos de la solución, entre otras. La cantidad de elementos que se modifican está determinada por el parámetro de entrada β. En nuestros trabajos, con el objetivo de garantizar la escalabilidad del algoritmo, el valor de β se expresa como un porcentaje del tamaño de la solución (es decir, $\beta \in [0, 1]$).

La fase de destrucción puede seguir un criterio aleatorio o un criterio voraz. La primera alternativa está orientada a la diversificación y permite al algoritmo explorar distintas regiones del espacio de búsqueda. En cambio, la segunda se centra en la intensificación, intentando mejorar la solución en la región actual.

3.5.2. Fase de reconstrucción

La fase de reconstrucción tiene como objetivo obtener de nuevo una solución factible tras haber sido destruida parcialmente en la fase descrita en la Sección 3.5.1. Para ello, se pueden nuevamente considerar dos enfoques distintos.

Por un lado, si el objetivo de esta fase es favorecer la diversificación, puede optarse por añadir elementos de forma aleatoria a la solución hasta alcanzar una solución factible. Por otro lado, puede aplicarse un criterio voraz que minimice el valor de la función objetivo de la solución.

3.6. Scatter Search

Scatter Search (SS) es una metaheurística poblacional que fue originalmente propuesta por [36]. Desde entonces, numerosos trabajos han abordado esta técnica, como el de [37], en el que se proporciona una descripción estructurada de la metodología. SS se basa en la idea de que los diseños y estrategias sistemáticos para explorar el espacio de búsqueda tienden a ofrecer mejores resultados que aquellos procedimientos que dependen en gran parte de elementos aleatorios.

Esta metaheurística se compone de cinco fases que permiten construir y mantener un conjunto de soluciones, denominado *RefSet*. La Figura 3.3 muestra una representación gráfica de las fases de esta metodología.

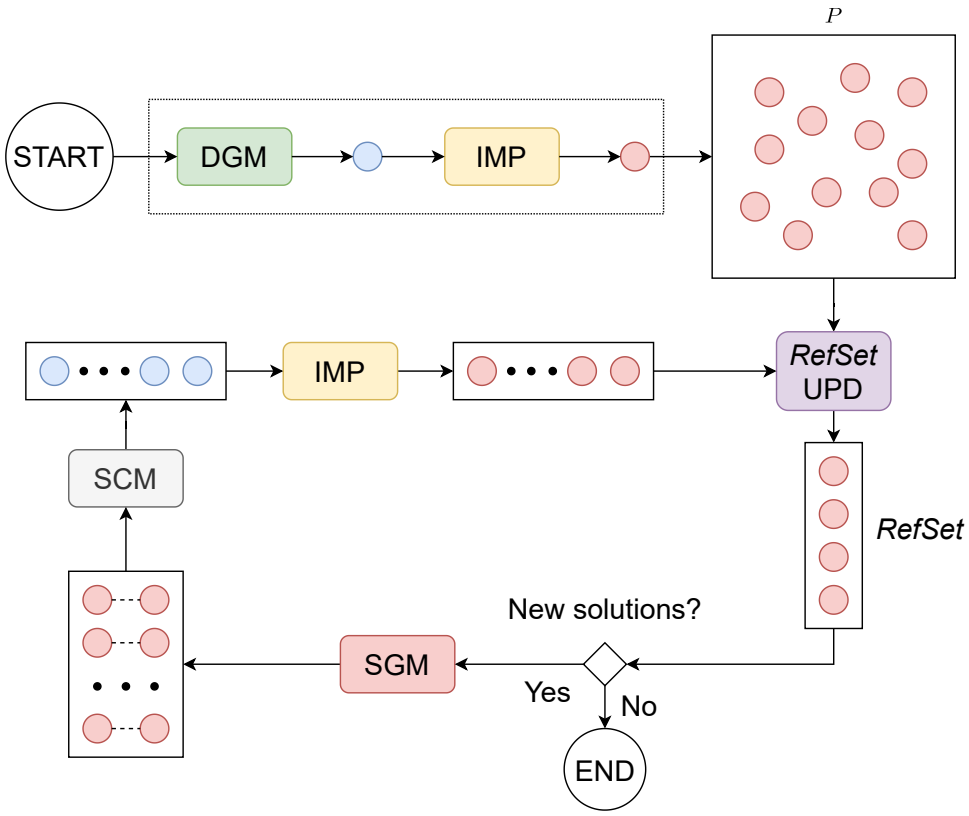


Figura 3.3: Esquema básico de la metaheurística *Scatter Search*.

El algoritmo comienza construyendo una población inicial P, la cual debe estar formada por soluciones de alta calidad y distintas entre sí. Para ello, el método *Diversification Generation Method* (DGM) crea soluciones iniciales diversas que posteriormente son mejoradas mediante *Improvement Method* (IMP). Las soluciones generadas son añadidas a la población P, repitiéndose este proceso hasta alcanzar un número predeterminado de soluciones iniciales.

Una vez construida la población P, el método *Reference Set Update* (*RefSet* UPD) selecciona un subconjunto de b soluciones, conformando el conjunto *RefSet*, el cual se mantiene durante toda la búsqueda. Este conjunto se forma con las $b/2$ mejores soluciones de P y las $b/2$ soluciones más diversas respecto a las ya seleccionadas. Un criterio habitual de parada de este procedimiento es que no se incorporen nuevas soluciones al *RefSet* después de una iteración del

bucle principal.

En cada iteración, el *RefSet* se utiliza como entrada para el método *Subset Generation Method* (SGM), que produce subconjuntos de soluciones (por ejemplo, pares) que serán combinados. Las implementaciones básicas de este método generan todos los pares posibles del *RefSet*; sin embargo, también es posible considerar subconjuntos de orden superior o reglas específicas de emparejamiento, según el problema que se trata de resolver.

Posteriormente, *Solution Combination Method* (SCM) crea nuevas soluciones combinando las soluciones generadas por SGM. Estas nuevas soluciones son mejoradas utilizando el método IMP y luego evaluadas por el método *RefSet* UPD. Una regla común para actualizar el *RefSet* consiste en añadir una solución al *RefSet* únicamente si mejora a la peor solución del conjunto. Para mantener la diversidad, dicha solución sustituye a la solución más parecida y de menor calidad dentro del *RefSet*. El bucle principal de SS termina cuando no se añaden nuevas soluciones al conjunto.

Una vez alcanzada esta condición, la búsqueda puede finalizar o reiniciarse mediante una reconstrucción del *RefSet*, conservando la mejor solución obtenida hasta el momento y añadiendo las $b - 1$ restantes de la misma manera que en la fase inicial. En este caso, la búsqueda finaliza tras un número predefinido de pasos de reconstrucción.

De los cinco métodos que conforman SS, dos de ellos *RefSet* UPD y SGM se tratan de procedimientos que pueden aplicarse directamente a cualquier problema de optimización combinatoria [38]. Por el contrario, DGM, IMP y SCM suelen depender del problema específico y requieren un diseño adaptado al contexto particular en el que se está trabajando.

3.6.1. Parallel Scatter Search

Esta variante de SS que se propone como avance metodológico de esta Tesis Doctoral tiene como objetivo aprovechar todos los procesadores de los ordenadores actuales en el proceso de búsqueda, tal y como se explica en la Sección 2.6. Los procedimientos diseñados pretenden ser aplicables a una amplia gama de problemas de optimización combinatoria.

Primero, es importante diferenciar entre paralelismo y concurrencia. El primero se refiere a la ejecución simultánea de múltiples tareas, mientras que el segundo hace referencia al concepto de avanzar en múltiples tareas al mismo tiempo, sin que necesariamente se ejecuten de forma simultánea. El diseño de

un algoritmo paralelo tiene como objetivo maximizar el paralelismo, aunque no siempre es posible debido a limitaciones del hardware.

Al diseñar un algoritmo paralelo [39], el uso correcto de los hilos resulta un mecanismo fundamental para lograr el paralelismo. Un hilo es una unidad de ejecución ligera e independiente dentro de un proceso que permite que múltiples tareas se ejecuten concurrentemente en un procesador multinúcleo o en un entorno multiprocesador. El uso de múltiples hilos permite que un algoritmo continúe ejecutándose incluso cuando algunas de sus fases son costosas en tiempo. Además, los hilos posibilitan el paralelismo al dividir un algoritmo en unidades de trabajo más pequeñas que pueden ejecutarse de manera concurrente, lo que puede acelerar la ejecución en un entorno multiprocesador.

Los modelos de memoria en el paralelismo desempeñan un papel crucial al determinar cómo se comparte, accede y sincroniza la información entre múltiples hilos o procesos en un entorno de computación paralela. Estos modelos definen las reglas y garantías que rigen la consistencia y visibilidad de la memoria. En la propuesta metodológica presentada en esta Tesis Doctoral, se utiliza un modelo de memoria compartida [40], donde todos los hilos comparten el mismo espacio de memoria. En este modelo, el desarrollador es responsable de determinar cómo se hacen visibles las actualizaciones realizadas por un hilo para los demás, evitando condiciones de carrera.

En esta propuesta, no se vincula el trabajo a una tecnología específica, permitiendo a la comunidad investigadora que desee aplicarla a su investigación adaptar los diseños a su lenguaje de programación y herramienta de paralelización de preferencia. Específicamente, para cada estrategia algorítmica propuesta, se asume que se dispone de un conjunto de hilos, cada uno capaz de ejecutar cualquiera de las tareas consideradas. El número de hilos disponibles depende únicamente de las especificaciones del hardware, lo que hace que los diseños paralelos propuestos sean escalables. Si hay más tareas que hilos disponibles, algunas tareas deberán esperar hasta que un hilo quede disponible para continuar con el proceso.

Cabe señalar que, en los diseños propuestos, la reducción del tiempo no será directamente proporcional al número de hilos disponibles. Esto se debe principalmente a que la creación y gestión de hilos, así como los cambios de contexto, conllevan una sobrecarga que debe tenerse en cuenta al diseñar algoritmos paralelos.

A continuación se enumeran y describen las tres propuestas de diseño paralelo de SS.

Hoarding Parallel Scatter Search

La primera propuesta, *Hoarding Parallel Scatter Search* (HPSS), tiene como objetivo reducir el tiempo total de ejecución de SS mediante la paralelización de aquellas fases que pueden ejecutarse simultáneamente sin necesidad de comunicación.

La generación de la población inicial P, compuesta por $|P|$ ejecuciones independientes de DGM seguidas de IMP, no requiere compartir datos ni información.

Dado que IMP se aplica a soluciones generadas por DGM, IMP depende de DGM y estos métodos no pueden ejecutarse simultáneamente durante la generación de P. La paralelización de esta etapa en HPSS consiste en distribuir la construcción de cada solución entre los hilos disponibles. Con este enfoque, se espera que el tiempo necesario para generar la población inicial se reduzca en un factor cercano a T (dependiendo de la sobrecarga), donde T es el número de hilos disponibles.

Una vez generada la población, el siguiente paso consiste en construir el *RefSet*, mezclando soluciones de alta calidad y soluciones diversas. Este paso no es adecuado para la paralelización, ya que la selección de la siguiente solución que va a formar parte del *RefSet* depende de las soluciones ya presentes en el conjunto.

Al seleccionar la siguiente solución de alta calidad, es necesario verificar que no es igual a una solución que se encuentre ya en el *RefSet*. Si lo que queremos es añadir soluciones distintas, también requerimos la información de las soluciones que ya forman parte del *RefSet* ya que la medida de diversidad se calcula normalmente con una métrica de distancia entre la solución candidata y las ya presentes en el *RefSet*. Por tanto, la lógica de *RefSet* UPD no permite la adición simultánea de múltiples soluciones.

La única parte que puede paralelizarse es la evaluación de la distancia entre cada solución candidata y el *RefSet* en construcción. Sin embargo, esto suele ser un proceso rápido y la sobrecarga asociada con la gestión de hilos podría ser más costosa en tiempo que el propio proceso.

A continuación, se aplica SGM, un procedimiento relativamente simple que consiste en generar todos los pares de soluciones que se van a combinar para crear nuevas soluciones. El tiempo requerido para este paso suele ser despreciable, por lo que no es un buen candidato para su paralelización.

El método de combinación de soluciones es una de los fases críticas de

SS. Afortunadamente, cada combinación de soluciones es independiente de las demás, y no es necesario realizar sincronización entre ellas. Por lo tanto, SCM es un excelente candidato para paralelización.

Lo mismo aplica al método de mejora: la ejecución de IMP sobre cada solución resultante de SCM puede realizarse de manera independiente en los hilos disponibles. Si SCM se paraleliza de forma independiente a IMP, IMP tendría que esperar hasta que todas las soluciones hayan sido combinadas para comenzar su paralelización.

Para evitar tiempos de espera innecesarios, la paralelización de ambas fases se realiza conjuntamente. Cada combinación se lanza en un hilo independiente y, cada vez que se crea una nueva solución a partir de una combinación de soluciones procedentes del *RefSet*, se aplica el método de mejora. En otras palabras, la combinación y la mejora se ejecutan concurrentemente, minimizando el tiempo necesario para generar nuevas soluciones.

Cada una de las soluciones creadas por las combinaciones tras SCM e IMP, se evalúa como candidata para entrar al *RefSet*. Esta evaluación no puede realizarse en paralelo, ya que el criterio de aceptación requiere compararla con las soluciones que forman parte del *RefSet*. Por lo tanto, este paso se ejecuta de forma secuencial y las soluciones mejoradas se encolan hasta que se hayan evaluado las candidatas anteriores.

La Figura 3.4 muestra el esquema de HPSS basado en el esquema original de SS representado en la Figura 3.3. Las tareas que se ejecutan en paralelo están representadas sobre un fondo rayado. HPSS comienza lanzando una nueva tarea para cada solución, donde cada tarea ejecutará un paso independiente de DGM e IMP, generando una nueva solución para la población inicial.

Una vez creada la población inicial, el *RefSet* se construye siguiendo un enfoque secuencial. Luego, mientras se incluyen nuevas soluciones en el *RefSet*, el SGM genera secuencialmente todos los pares de soluciones a combinar. Cada subconjunto (típicamente un par) de soluciones se combina mediante el SCM en paralelo, y para cada solución resultante de la combinación, se lanza un nuevo procedimiento de mejora.

Finalmente, todas las soluciones mejoradas se encolan para ser evaluadas secuencialmente como candidatas a ingresar en el *RefSet*.

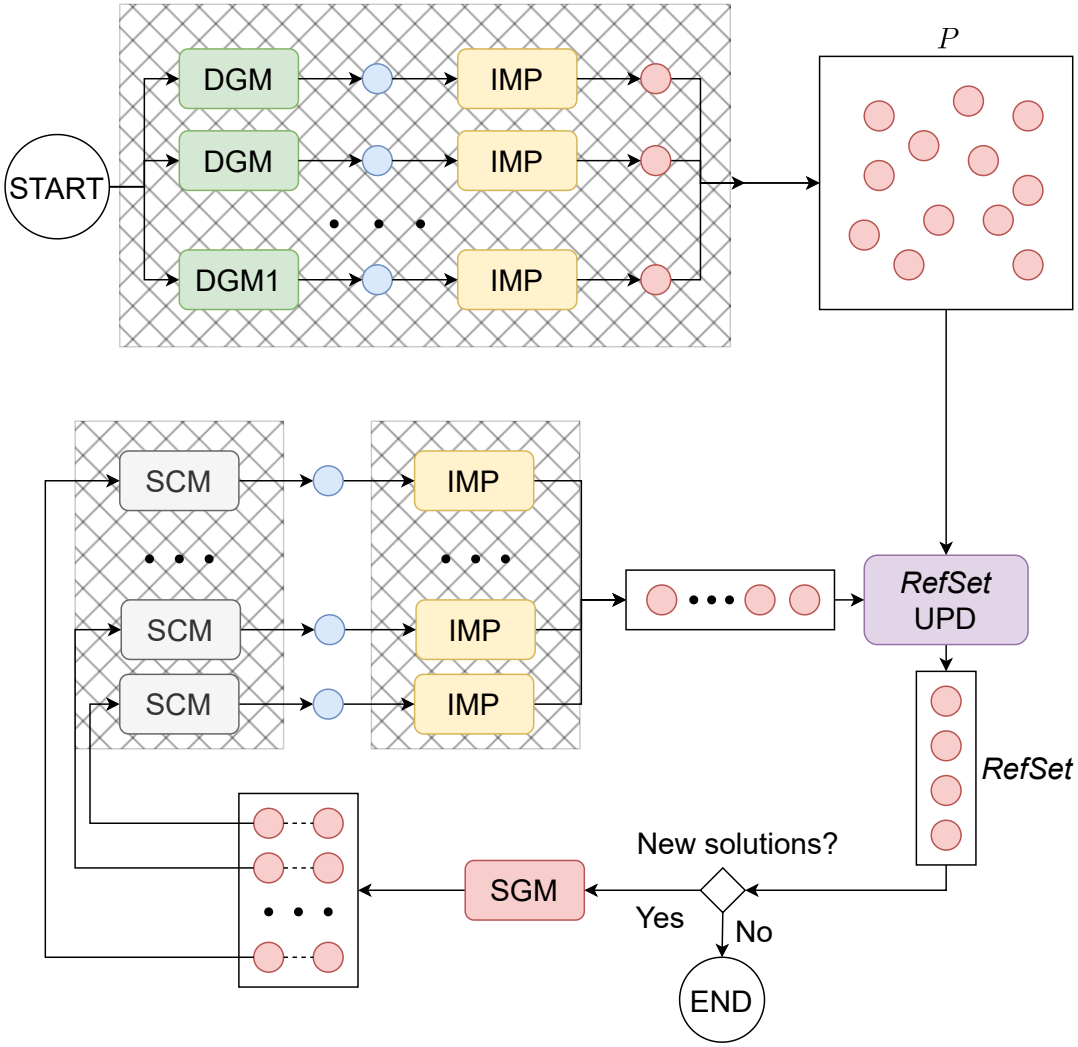


Figura 3.4: Esquema de *Hoarding Parallel Scatter Search*.

Agglomerative Parallel Scatter Search

La segunda propuesta de paralelización, denominada *Agglomerative Parallel Scatter Search* (APSS), tiene como objetivo explorar regiones más amplias del espacio de búsqueda en comparación con su versión secuencial. La mayoría de los trabajos publicados sobre SS proponen múltiples procedimientos constructivos, métodos de mejora y mecanismos de combinación. Los investigadores suelen seleccionan la mejor alternativa en cada fase mediante experimentación empírica, evaluando el rendimiento promedio o global. Sin embargo,

esto implica que otras alternativas que han obtenido peores resultados genéricos se descarten, pudiendo ofrecer un buen rendimiento en algunas instancias particulares.

El diseño de APSS parte de la siguiente hipótesis: ¿mejora el rendimiento del algoritmo SS si se incorporan todos los procedimientos constructivos, métodos de mejora y mecanismos de combinación propuestos, en lugar de seleccionar solo los mejores? Resulta inviable comprobar esta hipótesis en una implementación secuencial, debido al elevado coste computacional que implicaría evaluar todas las combinaciones posibles en tiempos razonables. No obstante, gracias a la capacidad de procesamiento paralelo, es factible integrar todos estos componentes en una única implementación, aprovechando las ventajas individuales de cada uno. Si bien esto no necesariamente reducirá el tiempo de ejecución (e incluso podría incrementarlo ligeramente, ya que cada fase tardaría tanto como la propuesta más lenta de esa misma fase), se incrementa la diversidad del espacio explorado, lo que podría derivar en soluciones de mejor calidad.

El aumento en el tiempo de cómputo puede atribuirse a que, en el diseño secuencial, la elección de cada componente también considera el coste computacional. Por ello, alternativas costosas pueden ser descartadas. En cambio, en APSS se permite incluirlas, aunque ello implique un mayor tiempo de ejecución. La Figura 3.5 presenta el esquema general de APSS.

Se asume que existen C procedimientos constructivos ($\mathit{DGM}_1 \ldots \mathit{DGM}_C$), I métodos de mejora ($\mathit{IMP}_1 \ldots \mathit{IMP}_I$) y S mecanismos de combinación de soluciones ($\mathit{SCM}_1 \ldots \mathit{SCM}_S$). En la versión secuencial, cada solución de la población inicial se genera mediante un DGM específico y posteriormente se mejora con un único IMP. APSS, en cambio, utiliza el paralelismo para generar una solución con cada DGM disponible y mejorar cada una con todos los IMP propuestos. Así, en cada iteración, se crean $C \times I$ soluciones, que se agregan a la población inicial. Gracias al paralelismo, el tiempo para generar estas soluciones puede ser comparable al necesario para generar una sola en la versión secuencial.

El conjunto *RefSet* se construye de forma secuencial, y el mecanismo de combinación (SGM) genera los pares de soluciones habituales en el enfoque SS. APSS combina todos los SCM disponibles con todos los IMP para generar nuevas soluciones. Es decir, cada par de soluciones se somete a todos los métodos de combinación y mejora, resultando en $S \times I$ soluciones candidatas para ser añadidas al *RefSet*. Como la actualización del *RefSet* debe hacerse de forma secuencial, estas soluciones se encolan para su evaluación. Aunque

los procesos de combinación y mejora se ejecutan en paralelo, el número de soluciones generadas es considerablemente mayor, lo que puede implicar un tiempo de ejecución total más elevado que el de la versión secuencial. APSS se detiene cuando no se añaden nuevas soluciones al conjunto *RefSet*, al igual que en SS y HPSS.

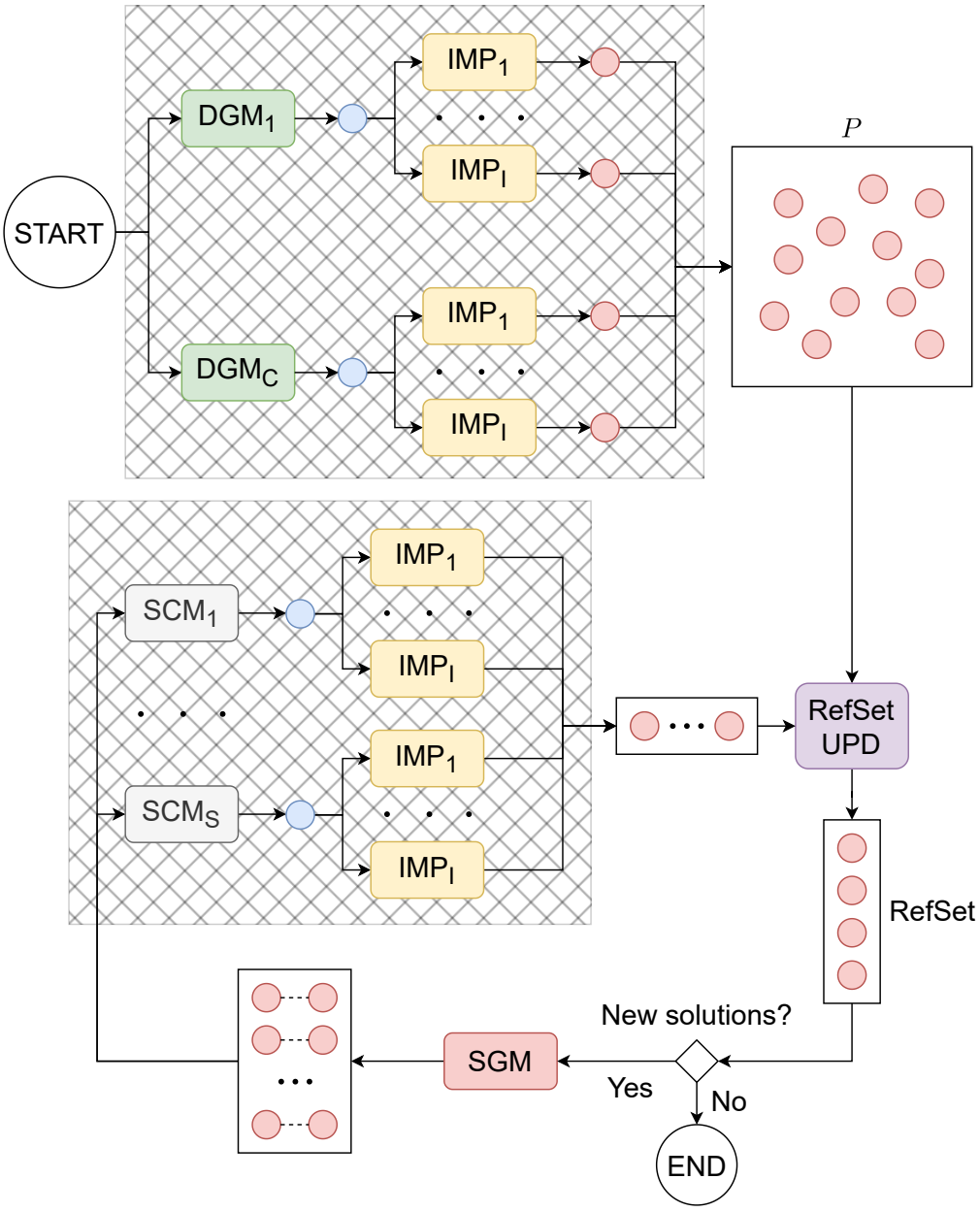


Figura 3.5: Esquema de *Agglomerative Parallel Scatter Search*.

Cooperative Parallel Scatter Search

La tercera propuesta, llamada *Cooperative Parallel Scatter Search* (CPSS), busca un equilibrio entre la reducción del tiempo de cómputo de HPSS y el incremento en la exploración del espacio de búsqueda ofrecido por APSS. La idea es fomentar la cooperación entre los distintos componentes del algoritmo para generar soluciones de alta calidad y diversidad.

CPSS se basa en APSS, pero incorpora un paso intermedio denominado *Select Best and Diverse* (SBD). Dado un conjunto de soluciones, este procedimiento selecciona la mejor, evaluando la función objetivo, y la más diversa. La diversidad se define como la distancia mínima respecto a las demás soluciones: la solución más diversa es aquella cuya distancia a la solución más parecida es la mejor. Dado un conjunto S de soluciones generadas por algún DGM, IMP o SCM, se definen:

$$s^b = \arg\min_{s \in S} f(s)$$

$$s^d = \arg\max_{s \in S} \min_{r \in S} d(s, r)$$

donde $f(s)$ representa la función objetivo y $d(s, r)$ una métrica de distancia entre las soluciones s y r, que depende del problema y la representación de la solución.

La Figura 3.6 muestra el esquema propuesto para el enfoque CPSS. El algoritmo comienza con la generación de soluciones iniciales de forma paralela utilizando todos los DGM disponibles. Este conjunto se introduce en el módulo SBD, que selecciona dos soluciones: s^b y s^d. Estas se utilizan como entrada para todos los métodos de mejora disponibles y se ejecutan en paralelo, generando un nuevo conjunto de soluciones. Estas también son evaluadas por SBD, que selecciona nuevamente la mejor y la más diversa para que pasen a formar parte de la población inicial.

Cabe destacar que CPSS no aplica los métodos de mejora a todas las soluciones generadas por los DGM, sino solo a aquellas seleccionadas por SBD, lo que reduce considerablemente el esfuerzo computacional respecto a APSS, manteniendo al mismo tiempo un balance entre intensificación y diversificación.

El proceso de combinación y mejora en CPSS imita la construcción del conjunto P. Cada conjunto de soluciones generadas por un SCM se pasa al módulo SBD, que extrae dos soluciones. Estas se mejoran con todos los IMP en paralelo, y las soluciones resultantes se proponen para ser añadidas al *RefSet*, el cual se actualiza secuencialmente.

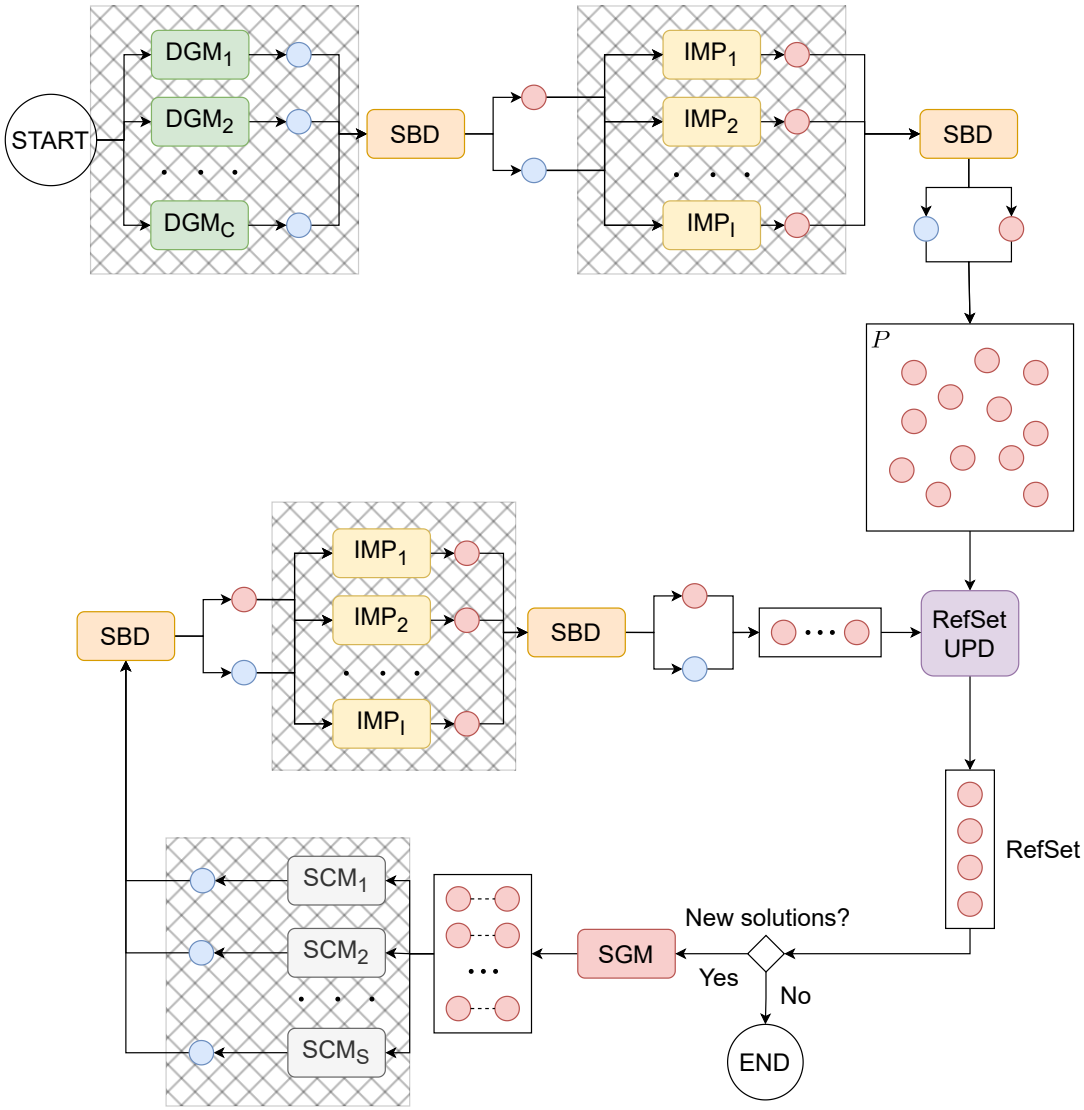


Figura 3.6: Esquema de *Cooperative Parallel Scatter Search*.

A diferencia de APSS, el número de soluciones evaluadas en esta etapa es menor, lo que debe reducir el tiempo total de ejecución. Así, CPSS conserva todos los métodos de combinación y mejora propuestos, pero los aplica a un subconjunto reducido de soluciones. El objetivo es lograr resultados de una calidad comparable a APSS, pero con menor coste computacional. Al igual que en las demás variantes, el algoritmo finaliza cuando no se incorporan nuevas soluciones al conjunto *RefSet*.

Capítulo 4

Análisis conjunto de los resultados

En este capítulo se analizan los resultados obtenidos para los distintos problemas de optimización abordados en esta tesis. Se presenta un resumen de cada propuesta desarrollada y, para cada problema, se reportan los valores promedio de la función objetivo junto con otras medidas relevantes como el tiempo de cómputo.

4.1. Métricas de evaluación

Con el objetivo de realizar una evaluación rigurosa y homogénea de las propuestas desarrolladas para resolver los problemas de dominación considerados, es fundamental establecer un conjunto de métricas comunes. Estas métricas permitirán comparar de manera justa el rendimiento de los algoritmos tanto entre sí como respecto a métodos existentes en la literatura. Asimismo, se garantiza la coherencia en el análisis de resultados, al aplicar los mismos criterios en todos los experimentos realizados.

Todas las tablas incluyen las siguientes métricas: Promedio, valor promedio de la función objetivo obtenido por cada algoritmo; Tiempo, tiempo medio de cómputo requerido (en segundos); Desv., desviación porcentual media respecto a la mejor solución encontrada; y #Mejores, número de veces que el algoritmo alcanza la mejor solución del experimento. En todas las tablas, los mejores resultados sobre cada métrica aparecen en negrita.

Los experimentos de todos los problemas en los que se han trabajado se dividen en dos fases diferenciadas: una fase preliminar y una fase final. La fase

preliminar tiene como objetivo ajustar los parámetros de entrada del algoritmo y determinar la mejor estrategia para guiar la búsqueda. Por su parte, la fase final está diseñada para llevar a cabo una comparación directa con algoritmos del estado del arte, con el propósito de evaluar la aportación de la propuesta presentada en cada trabajo.

4.2. Reproducibilidad

En el contexto de la investigación científica actual, la reproducibilidad se ha convertido en un pilar fundamental para validar y comparar los resultados presentados en los artículos académicos. En particular, en el área de la optimización y los algoritmos metaheurísticos, donde la aleatoriedad y la configuración experimental influyen en los resultados que se obtienen, garantizar la posibilidad de reproducir los experimentos se considera una buena práctica esencial.

La reproducibilidad facilita la transferencia de conocimiento, promueve la transparencia y contribuye al progreso científico. Las revistas de alto impacto exigen cada vez más que los autores proporcionen detalles sobre la configuración experimental, los conjuntos de datos utilizados y, cuando sea posible, el código fuente desarrollado.

Con el fin de fomentar la reproducibilidad de los experimentos presentados en este trabajo, se han seguido una serie de principios fundamentales. En primer lugar, se reportan el lenguaje de programación en el que fue implementado el algoritmo y la máquina en la que se ha ejecutado en cada sección de resultados del problema. Además, siempre se especifican los parámetros seleccionados para la configuración de la propuesta. Por último, para facilitar la tarea de futuros investigadores, se facilita el acceso a las instancias usadas y el código fuente desarrollado. Los enlaces para encontrar toda esta información de cada problema son:

- MDSP: https://grafo.etsii.urjc.es/MDSP
- MPP: https://grafo.etsii.urjc.es/MonitorPlacement
- WTDP: https://grafo.etsii.urjc.es/wtdp
- PFSS: https://grafo.etsii.urjc.es/ParallelScatterSearch

4.3. Resultados obtenidos del Minimum Dominating Set Problem

En esta sección se presentan y analizan los resultados obtenidos en las pruebas computacionales realizadas para resolver el MDSP con el algoritmo basado en la metaheurística IG. Se resolvieron un total de 124 instancias con una máquina que tiene un procesador AMD EPYC 7282 a 2.8 GHz con 16 GB de memoria RAM. Las heurísticas fueron implementadas en Java 11, mientras que el enfoque exacto se desarrolló utilizando el software Gurobi 9.

El conjunto de instancias se divide en dos subconjuntos diferenciados: por un lado, se consideraron 28 instancias tomadas directamente de la literatura con el objetivo de realizar una comparación justa con trabajos previos; por otro, se generaron 96 instancias adicionales, de mayor complejidad, con el propósito de evaluar el rendimiento y los límites tanto del enfoque exacto como del metaheurístico.

Gracias a los experimentos preminilares realizados, la configuración final del algoritmo IG para el problema del *Minimum Dominating Set* queda ajustada de la siguiente manera: la solución inicial se genera mediante el procedimiento GIP, que se trata de un constructivo voraz que ofrece soluciones de alta calidad y tiempos de cómputo muy reducidos. En la solución, se incluyen siempre los nodos soporte desde el inicio, ya que, en promedio, dominan el 63.5 % de los nodos. Esta solución se mejora mediante una búsqueda local eficiente (ELS), que acelera el proceso en promedio un 66 % respecto a una búsqueda local que itere sobre todas las posibles soluciones. En la fase de destrucción y reconstrucción de IG, se aplica una destrucción aleatoria y una reconstrucción voraz, balanceando exploración y explotación. El porcentaje de destrucción se establece en $\beta = 0.2$, ya que logra la mejor calidad con menor tiempo. Finalmente, el criterio de parada se fija en $\Delta = 200$ iteraciones sin mejora, dado que no se observa mejora más allá de ese umbral.

Los algoritmos considerados en los experimentos finales son: el propuesto (IG) con la configuración establecida gracias a los experimentos preliminares; el algoritmo *Randomised Local Search* (RLS) [1]; la resolución del modelo exacto mediante Gurobi (*Integer Linear Programming* (ILP)); y tres variantes de algoritmos de colonias de hormigas con búsqueda local: *Ant Colony Optimization Local Search* (ACO-LS) [41], *Ant Colony Optimization Pre-Processing Local Search* (ACO-PP-LS) (con fase de preprocesamiento) [42] y *Ant Colony Optimization Local Search Start* (ACO-LS-S) (con selección de vértices y feromonas inicializadas a partir de soluciones voraces). Para garantizar una comparación

justa, se han empleado los mismos parámetros indicados en sus respectivos trabajos originales, tal como se detalla en [1]. En todos los casos se ha considerado el tiempo total de cómputo, incluyendo la generación de la solución inicial.

Algoritmo	Promedio	Tiempo (s)	Desv. (%)	#Mejores
IG	1304.68	164.38	0.04	20
ILP	**1303.11**	**0.25**	**0.00**	**28**
RLS	1309.35	600.00	0.16	20
ACO-LS	1550.66	600.00	11.82	13
ACO-PP-LS	1545.96	600.00	11.51	14
ACO-LS-S	1341.33	600.00	1.33	15

Tabla 4.1: Comparación entre IG, RLS, ILP y tres variantes ACO sobre el conjunto de 28 instancias propuestas en [1].

Los resultados de la Tabla 4.1 indican que ACO-LS y ACO-PP-LS son las variantes menos competitivas. RLS mejora estos resultados en términos de desviación y número de mejores soluciones, confirmando su superioridad dentro de las propuestas previas. Sin embargo, IG supera claramente a RLS tanto en calidad de las soluciones como en tiempo de ejecución, siendo hasta cuatro veces más rápido en promedio. La capacidad de ILP para resolver todas las instancias en menos de un segundo sugiere que se trata de instancias estructuralmente simples, con múltiples soluciones buenas.

Para evaluar el rendimiento en contextos más exigentes, se ha generado un nuevo conjunto de 96 instancias formadas por grafos aleatorios más diversos y carentes de estructura. Los resultados se muestran en la Tabla 4.2

Algoritmo	Promedio	Tiempo (s)	Desv. (%)	#Mejores
IG	**10.75**	**23.88**	**1.23**	**88**
ILP	11.94	1799.77	10.18	43
RLS	12.53	600.00	15.07	14
ACO-LS	14.12	600.00	33.19	0
ACO-PP-LS	14.11	600.00	33.17	0
ACO-LS-S	14.12	600.00	33.18	0

Tabla 4.2: Resultados obtenidos por los seis algoritmos sobre las 96 nuevas instancias aleatorias.

Tal y como era previsible, ILP alcanza su límite computacional en este nuevo conjunto de instancias, siendo incapaz de demostrar optimalidad en la mayoría de los casos dentro del tiempo límite (1800s), aunque consigue encontrar la mejor solución conocida en 43 ocasiones. Por otro lado, IG demuestra su robustez al alcanzar 88 de las 96 mejores soluciones, con una desviación media del 1.23 %, lo que demuestra que, incluso en los casos donde no obtiene la mejor solución, se mantiene muy próximo a ella.

RLS, el mejor algoritmo previo, presenta una desviación del 15.07 % y tan solo alcanza 14 soluciones óptimas, requiriendo además tiempos significativamente mayores. Por tanto, IG se consolida como una propuesta altamente competitiva tanto en calidad como en eficiencia computacional. Por su parte, las tres variantes de ACO no muestran diferencias relevantes entre sí y sus resultados corroboran la complejidad de las nuevas instancias, siendo únicamente IG y RLS capaces de enfrentarlas con éxito en tiempos razonables.

4.4. Resultados obtenidos del Monitor Placement Problem

En esta sección se presentan y analizan los resultados obtenidos al aplicar la propuesta desarrollada para resolver el MPP. El objetivo es evaluar la eficacia y eficiencia del algoritmo propuesto en comparación con el estado del arte. Para ello, se ha replicado el entorno experimental utilizado en trabajos previos relevantes, asegurando así la validez y comparabilidad de los resultados obtenidos.

Todos los algoritmos han sido implementados en Java 11 y las pruebas experimentales se han ejecutado en una máquina con procesador AMD EPYC 7282 a 2.8 GHz y 72 GB de memoria RAM. Se ha empleado el mismo conjunto de instancias utilizado en la literatura, concretamente en el trabajo donde se describe el que era el mejor algoritmo para abordar el MPP [8].

Del conjunto original de instancias, no se han podido considerar $\mathrm{delaunay\_n19}$ y $\mathrm{delaunay\_n20}$ debido a limitaciones de memoria, ya que la máquina empleada no dispone de suficiente capacidad para almacenarlas en RAM en tiempo de ejecución, a pesar de que el algoritmo está técnicamente capacitado para resolverlas. No obstante, los resultados obtenidos con las instancias de mayor tamaño demuestran que la propuesta mejora su desempeño a medida que aumenta la complejidad de la instancia.

El conjunto final considerado consta de 37 instancias,de las cuales 10 fueron usadas para el ajuste de parámetros del algoritmo realizado durante los experimentos preliminares.

El algoritmo se ajustó gracias a la ejecución de distintos experimentos preliminares para encontrar la mejor combinación de parámetros. Se determinó que los mejores valores se obtenían seleccionando como procedimiento constructivo *GreedyDestructive* junto con la búsqueda local ILS (una búsqueda local inteligente propuesta en este trabajo que realiza menos movimientos), debido a su buen equilibrio entre calidad de solución y bajo tiempo de cómputo. Se evaluaron dos métodos para la fase *shake* del BVNS: *RandomShake* e *IntensifiedShake*, variando los parámetros $k_{\text{max}} \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ y $\delta \in \{1, 10, 20, 30, 40\}$, donde k_{max} representa el tamaño máximo de la vecindad como proporción del número de nodos y δ el número de iteraciones completas del BVNS. Para *RandomShake*, los valores óptimos fueron $k_{\text{max}} = 0.1$ y $\delta = 20$, mientras que para *IntensifiedShake* se seleccionaron $k_{\text{max}} = 0.2$ y $\delta = 20$, balanceando calidad y eficiencia. Comparando ambos, *IntensifiedShake* fue superior en todas las métricas, siendo por tanto el método *shake* elegido en la versión final del BVNS.

Una vez definidos todos los parámetros del algoritmo BVNS, el siguiente experimento se centra en evaluar su rendimiento comparando los resultados obtenidos con un algoritmo evolutivo híbrido, denominado (LS + PI) EA. Este algoritmo representa una de las estrategias más utilizadas en la literatura [8], al combinar la búsqueda local con heurísticas basadas en evolución. En este experimento final se considera el conjunto completo de 37 instancias, agrupadas por tipo.

La Tabla 4.3 presenta los resultados obtenidos al aplicar ambos algoritmos sobre todas las instancias de prueba. Cabe señalar que la columna Mediana reporta la mediana de la función objetivo obtenida tras 100 ejecuciones por instancia y T. Prom. (s) reporta el tiempo medio que tarda el algoritmo en llegar al valor, utilizando la misma metodología experimental descrita en [8] para poder comparar los resultados de forma justa.

Los resultados mostrados en la Tabla 4.3 evidencian que el algoritmo BVNS logra encontrar la mejor solución en las 37 instancias evaluadas, es decir, obtiene el mejor valor para cada tipo de instancia. En cuanto al tiempo de ejecución, se destaca la instancia de tipo frb, donde BVNS es aproximadamente 445 veces más rápido que (LS + PI) EA. Respecto a la desviación media, BVNS presenta un valor constante de 0.0 % en todos los casos, mientras que el algoritmo de referencia alcanza una desviación media aproximada del 17.0 %, con máximos en las instancias $\mathrm{delaunay}$ (40.47 %) e $\mathrm{internet\text{-}as}$ (33.22 %), y un mínimo en las

Tipo de instancia	Algoritmo	Mediana	T. Prom. (s)	Desv. (%)	#Mejores
delaunay	**BVNS**	**41311.72**	**272.21**	**0.00**	**9**
	(LS + PI) EA	100602.33	475.78	40.47	0
frb	**BVNS**	**584.67**	**0.56**	**0.00**	**15**
	(LS + PI) EA	588.67	252.56	0.66	0
internet-as	**BVNS**	**5701.00**	**19.95**	**0.00**	**1**
	(LS + PI) EA	7595.00	179.18	33.22	0
NREN	**BVNS**	**422.00**	**0.05**	**0.00**	**1**
	(LS + PI) EA	452.00	2.02	7.11	0
p2p-Gnutella	**BVNS**	**6001.67**	**6.49**	**0.00**	**9**
	(LS + PI) EA	7966.67	71.95	24.58	0
tech	**BVNS**	**1540.50**	**1.98**	**0.00**	**2**
	(LS + PI) EA	1658.50	32.08	7.33	0

Tabla 4.3: Comparación entre el algoritmo evolutivo híbrido (LS + PI) EA y el algoritmo propuesto BVNS para cada tipo de instancia.

de tipo frb (0.66 %). Es importante resaltar que, precisamente en estas últimas, la diferencia en tiempo de ejecución entre ambos algoritmos es la más significativa. Las mayores diferencias en desviación se producen en las instancias más complejas, lo que pone de manifiesto la escalabilidad del algoritmo propuesto. En base a este análisis, se concluye que BVNS es una alternativa competitiva para resolver el problema MPP, ofreciendo soluciones de alta calidad con un esfuerzo computacional reducido, incluso en los escenarios más exigentes.

4.5. Resultados obtenidos del Weighted Total Domination Problem

A continuación, se presentan los resultados experimentales obtenidos en el estudio del WTDP, con el objetivo de evaluar el rendimiento del algoritmo propuesto frente a los principales métodos existentes en la literatura. Todos los experimentos se han llevado a cabo en un procesador AMD Ryzen 9 5950X (3.4 GHz) con 128 GB de memoria RAM, utilizando Java 17 como lenguaje de programación. El conjunto de instancias utilizado se divide en dos subconjuntos diferenciados. Por un lado, se consideran 180 instancias derivadas directamente del estado del arte del problema WTDP, con tamaños que varían entre 20

y 125 nodos. Este conjunto se subdivide en dos grupos, según la literatura: MA (45 instancias) y NEW (135 instancias). Por otro lado, se propone un nuevo conjunto de 135 instancias más complejas, denominado CSM, con tamaños que oscilan entre 200 y 500 nodos, con el objetivo de analizar los límites de los algoritmos comparados.

Los experimentos se dividen, como siempre, en dos fases distintas: experimentación preliminar y experimentación final. La primera tiene como propósito seleccionar la mejor configuración para los algoritmos propuestos, mientras que la segunda se centra en evaluar la calidad del algoritmo propuesto en comparación con el algoritmo de referencia del estado del arte para el WTDP.

En la fase de experimentación preliminar, se seleccionaron los mejores parámetros para la configuración final del algoritmo propuesto. El procedimiento constructivo elegido fue BGC con $\alpha = 0.0$, ya que proporciona las mejores soluciones en comparación con otros valores de α. Asimismo, se selecciona el criterio de parada en el que detiene la búsqueda cuando es imposible añadir un nodo sin que la solución resultante sea de peor calidad que la mejor solución encontrada hasta el momento, en lugar de detenerse al alcanzar una solución factible. Para mejorar las soluciones construidas, se seleccionó la búsqueda local RLS, dado que reduce significativamente el tiempo de cómputo manteniendo la calidad. En cuanto al procedimiento JVNS, se estableció $k_{\mathsf{max}} = 0.4$ para la versión estándar y $k_{\mathsf{max}} = 0.3$ para la variante multiarranque, aunque finalmente se optó por la versión estándar debido a su mejor equilibrio entre calidad y tiempo. Además, el número de iteraciones se fijó en $\Delta = 60$, ya que el aumento de valor de este parámetro no aporta mejoras significativas y solo incrementa el tiempo de ejecución. Esta configuración final combina BGC + RLS como punto de partida para JVNS, maximizando la eficacia del enfoque propuesto.

Los siguientes experimentos se dedican a realizar una comparación exhaustiva del algoritmo propuesto JVNS con los métodos más destacados de la literatura. En particular, el mejor enfoque para el problema WTDP es un Algoritmo Genético llamado *Genetic Algorithm 1* (GA1), en el cual la población inicial se genera mediante un algoritmo GRASP tradicional, y cada individuo se mejora mediante una búsqueda local [43]. En dicho estudio, también se plantea un enfoque exacto, pero se concluye que GA1 es el mejor método para resolver el WTDP. Cabe destacar que en esta sección se emplea el conjunto completo de instancias en todos los experimentos.

El primer experimento compara la calidad de la solución inicial generada por JVNS y GA1. Mientras que el trabajo anterior utiliza un GRASP tradicional para construir la solución inicial, JVNS emplea un método Biased GRASP. Los

resultados del algoritmo anterior han sido extraídos directamente del manuscrito original. La Tabla 4.4 presenta los resultados obtenidos.

Algoritmo	Promedio	Tiempo (s)	Desv. (%)	#Mejores
Biased GRASP	**523.20**	**2.89**	**0.07**	**123**
GRASP	539.96	2.95	2.69	55

Tabla 4.4: Comparación de las soluciones iniciales de JVNS y GA1, generadas mediante Biased GRASP y GRASP respectivamente.

Ambos algoritmos requieren tiempos computacionales similares. Sin embargo, Biased GRASP demuestra una clara superioridad al alcanzar la mejor solución en 123 de las 135 instancias, con una desviación media de solo 0.07 %. Esta desviación indica que incluso en las instancias donde no alcanza el mejor valor del experimento, se mantiene muy cercano. Aunque el GRASP tradicional también presenta un buen rendimiento, solo logra la mejor solución en 55 instancias.

La mejor contribución exacta de la literatura propone cuatro métodos exactos aplicados al conjunto de instancias MA, menos complejas que las del conjunto NEW. Evaluamos JVNS sobre dichas instancias para observar qué tan cerca se encuentra de los óptimos conocidos. Los resultados se muestran en la Tabla 4.5.

Algoritmo	Promedio	Tiempo (s)	Desv. (%)	#Mejores
JVNS	95.35	**3.48**	0.03	44
f1	95.51	449.08	0.11	42
f1+	**95.33**	157.28	**0.00**	**45**
f2	**95.33**	151.22	**0.00**	**45**
f2+	**95.33**	51.04	**0.00**	**45**

Tabla 4.5: Comparación de JVNS con métodos exactos sobre las instancias MA.

El mejor método exacto es f2+, que alcanza todos los valores óptimos en tiempos razonables. JVNS, por su parte, logra resultados similares en calidad en apenas 3 segundos promedio, con una desviación de tan solo 0.03 % y 44 soluciones óptimas de 45.

A continuación, comparamos JVNS con el mejor algoritmo heurístico previo, GA1, sobre el conjunto de instancias NEW, originalmente diseñado para

evidenciar la necesidad de soluciones no exactas. Los resultados se presentan en la Tabla 4.6.

Número de nodos	Algoritmo	Promedio	Tiempo (s)	Desv. (%)	#Mejores
75	**JVNS**	**421.78**	**4.43**	**0.02**	**43**
	GA1	423.87	9.91	0.34	39
100	**JVNS**	**525.53**	**11.14**	**0.05**	**42**
	GA1	526.51	23.13	0.23	36
125	**JVNS**	**611.31**	**22.83**	**0.12**	**39**
	GA1	611.80	47.76	0.20	37

Tabla 4.6: Comparación entre JVNS y GA1 sobre el conjunto NEW.

Divididos por tamaño, los resultados muestran que JVNS requiere la mitad del tiempo computacional que GA1, alcanzando más soluciones óptimas y menor desviación en cada subconjunto.

Dado que los conjuntos MA y NEW son resueltos por ambos algoritmos sin suponer un alto tiempo de cómputo, proponemos un nuevo conjunto de instancias más complejas, denominado CSM, con tamaños entre 200 y 500 nodos. Ante la imposibilidad de obtener el código original de GA1, se implementó cuidadosamente conforme a su descripción. Los resultados se presentan en la Tabla 4.7.

Número de nodos	Algoritmo	Promedio	Tiempo (s)	Desv. (%)	#Mejores
200	**JVNS**	**871.40**	**100.73**	**0.00**	**45**
	GA1	937.08	962.77	6.72	1
350	**JVNS**	**1332.04**	**616.34**	**0.00**	**45**
	GA1	1794.06	1824.58	29.79	0
500	**JVNS**	**1776.40**	**1164.68**	**0.00**	**45**
	GA1	2539.57	1900.48	37.56	0

Tabla 4.7: Comparación entre JVNS y GA1 en el conjunto desafiante CSM.

Los resultados demuestran que JVNS escala adecuadamente ante instancias más complejas, superando a GA1 tanto en calidad como en eficiencia. A pesar del aumento en los tiempos de cómputo, JVNS requiere aproximadamente la mitad del tiempo que GA1, logrando todas las soluciones óptimas sin

desviación. Por el contrario, GA1 muestra un rendimiento decreciente a medida que aumentan los nodos.

4.6. Resultados obtenidos del Parallel Framework for Scatter Seach

En esta sección se presentan los resultados experimentales obtenidos al aplicar las diferentes variantes propuestas para paralelizar la metaheurística SS sobre las propuestas basadas en SS sobre tres problemas combinatorios. El objetivo principal de esta sección es analizar el comportamiento de cada propuesta de paralelización en cuanto a la calidad de las soluciones y eficiencia computacional.

Con el fin de realizar una comparación justa, todos los algoritmos han sido implementados en Java 21 y los experimentos se llevaron a cabo en un procesador AMD Ryzen 9 5950X con 32 hilos y 128 GB de memoria RAM. Para hacer la comparación lo más robusta posible, se ejecutaron todas las variantes 10 veces. Para cada experimento, se reportan las siguientes métricas adicionales a las explicadas al comienzo de la Sección 4.6.1:

- Mejor: el mejor valor de la función objetivo obtenido en todas las ejecuciones.
- Desv. Típica: la desviación estándar del valor de la función objetivo.
- T. Prom. (s): el tiempo promedio, en segundos, requerido por cada ejecución.

En este trabajo no se realizó un ajuste de los parámetros de los algoritmos, ya que se asumió que dicho ajuste fue llevado a cabo en los trabajos originales, y que los mejores valores de parámetros son los reportados en los artículos correspondientes. Por tanto, los valores específicos de los parámetros utilizados para cada algoritmo son exactamente los mismos que los indicados en los trabajos originales.

En las siguientes secciones se van a mostrar los resultados obtenidos al paralelizar las mejores propuestas algorítmicas, que representan el estado del arte de cada uno de los problemas, con las estrategias desarrolladas en este trabajo.

4.6.1. Capacitated Dispersion Problem

El primer problema usado para realizar este análisis se trata de *Capacitated Dispersion Problem* (CDP). En este experimento, se utilizan instancias con 50, 150 y 500 elementos propuestas por [44]. Los resultados se presentan en tres tablas, una para cada tamaño del problema.

La Tabla 4.8 resume los resultados obtenidos para las instancias más pequeñas.

$n = 50$	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	105.05	104.83	0.00069	$\leq$**0.01**	18	0.07
HPSS	104.94	104.75	**0.00058**	$\leq$**0.01**	16	0.27
APSS	**105.08**	**105.03**	0.00105	0.02	**19**	**0.03**
CPSS	**105.08**	104.98	0.00076	0.02	**19**	0.06

Tabla 4.8: Resumen de resultados para instancias con 50 elementos del problema CDP.

Las instancias con 50 elementos son demasiado pequeñas para que se aprecien diferencias significativas entre los métodos analizados. Todas las variantes ofrecen un rendimiento prácticamente equivalente, por lo que la paralelización no aporta ventajas notables en este caso.

La Tabla 4.9 presenta los resultados obtenidos en instancias de tamaño intermedio.

$n = 150$	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	112.80	112.18	0.00294	0.10	7	0.30
HPSS	112.73	112.12	**0.00087**	**0.01**	6	0.34
APSS	**113.02**	**112.59**	0.01263	0.24	**16**	**0.10**
CPSS	112.97	112.41	0.00804	0.22	13	0.12

Tabla 4.9: Resumen de resultados para instancias con 150 elementos del problema CDP.

En este conjunto, ninguna variante muestra grandes diferencias entre ejecuciones. HPSS produce soluciones de calidad similar a las de *Sequential Scatter Search* (SSS) pero en un tiempo diez veces menor. Las versiones paralelas

que tienen como objetivo mejorar la calidad, APSS y CPSS, obtienen mayor número de mejores soluciones que SSS, aunque con un aumento moderado en el tiempo de ejecución. Como era de esperar, los resultados confirman que HPSS se centra en la eficiencia, mientras que APSS y CPSS aumentan la exploración sacrificando la carga computacional requerida.

Finalmente, la Tabla 4.10 presenta los resultados para el conjunto más exigente.

$n = 500$	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	21.61	20.69	0.04731	3.89	17	2.70
HPSS	21.54	20.53	**0.00701**	**0.34**	18	2.76
APSS	**22.19**	**21.12**	0.27494	7.39	**28**	**1.11**
CPSS	22.14	21.03	0.21332	7.10	26	1.13

Tabla 4.10: Resumen de resultados para instancias con 500 elementos del problema CDP.

Las instancias con 500 elementos permiten evaluar de manera más clara el comportamiento de las versiones paralelas. HPSS mantiene una calidad de soluciones similar a la de SSS, pero con un tiempo de ejecución diez veces menor. Tanto APSS como CPSS superan a SSS en calidad, aunque requieren de un esfuerzo computacional mayor. En este contexto, la mejora en calidad justifica el coste de tiempo adicional.

4.6.2. Max-Cut Problem

El segundo problema es el *Max Cut Problem* (MCP). En este experimento, se consideran dos conjuntos de instancias, derivados del trabajo original sobre SS secuencial propuesto por [45]. Los resultados se comparan en un total de 2 tablas, una por cada conjunto de instancias. El total de instancias asciende a 84 y se dividen de la siguiente manera:

- C1: Compuesto por 54 instancias generadas mediante un generador de grafos descrito en [46], con tamaños que oscilan entre 800 y 3000 nodos. Las instancias incluyen distintos tipos de grafos, con pesos de -1, 0 o 1. Este conjunto fue posteriormente utilizado en [47, 48].
- C2: Formado por 30 instancias de baja densidad, también con pesos de

-1, 0 o 1, y tamaños entre 128 y 2744 nodos. Estas instancias fueron propuestas en [48].

La primera Tabla 4.11 resume los resultados obtenidos al comparar las variantes del algoritmo SS sobre el primer conjunto de instancias.

C1	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	5121.07	5106.35	30.34	910.60	1	1.03
HPSS	5114.83	5097.15	13.53	**408.59**	1	1.26
APSS	**5149.91**	**5132.47**	17.92	1283.99	**41**	**0.07**
CPSS	5106.52	5089.24	**1.82**	1760.44	16	1.12

Tabla 4.11: Resumen de resultados para el C1 de instancias del MCP (54 instancias).

En primer lugar, si se analiza la variante HPSS, diseñada para reducir el tiempo de cómputo respecto a SSS, se observa que consigue reducir a la mitad el tiempo requerido, manteniendo una calidad de soluciones similar. Las pequeñas diferencias en cuanto a calidad se deben a que se tratan de algoritmos no deterministas, ya que incorporan componentes aleatorios durante la búsqueda. APSS destaca claramente en calidad, alcanzando 41 de las 54 mejores soluciones y una desviación casi nula. Esto se debe a que incluye todas las estrategias constructivas, de búsqueda local y de combinación del enfoque original, lo que permite una exploración más profunda del espacio de soluciones.

Sin embargo, este diseño implica un coste computacional elevado, resultando en tiempos de ejecución superiores a los de SSS. Por su parte, CPSS consigue mejores resultados que SSS y HPSS, aunque sin alcanzar el rendimiento de APSS. Sorprendentemente, no presenta mejoras significativas en tiempo respecto a APSS. Esto se debe a que algunos de los procesos más costosos del algoritmo original deben ejecutarse de forma síncrona en CPSS, penalizando su rendimiento.

La Tabla 4.12 muestra los resultados para el segundo conjunto. Nuevamente, HPSS destaca por reducir significativamente el tiempo de cómputo en comparación con SSS, sin reducir la calidad de los resultados obtenidos. APSS tiene un comportamiento similar al observado sobre el C1, logrando 20 de las 30 mejores soluciones con una ligera penalización temporal. CPSS, sin embargo, no consigue obtener buenos resultasos para este conjunto específico, ya que presenta altos tiempos de ejecución sin grandes mejoras en calidad.

Estos resultados refuerzan la idea de que la elección de la variante paralela

C2	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	1105.00	**1099.67**	**3.32**	656.69	16	0.31
HPSS	1101.47	1094.89	8.97	**270.39**	12	0.60
APSS	**1105.20**	1097.13	4.75	698.40	**20**	**0.11**
CPSS	1081.80	1073.79	13.91	1139.54	13	1.17

Tabla 4.12: Resumen de resultados para el C2 de instancias del MCP (30 instancias).

debe hacerse en función del tipo de instancia, del problema estudiado y del equilibrio deseado entre calidad y tiempo de cómputo.

4.6.3. Profile Minimization Problem

Los experimentos realizados en este apartado se han llevado a cabo utilizando uno de los grupos de instancias empleados en [49]. En dicho trabajo, se identifican tres conjuntos principales: D4-trees, K-Graphs y HB (*Harwell Boeing Sparse Matrix Collection* [50]). En este trabajo, solo se ha considerado la colección HB, ya que los otros dos grupos presentan instancias triviales que la versión secuencial de SS puede resolver sin problemas, lo que no da lugar a estudiar correctamente el efecto de las versiones paralelas propuestas.

El conjunto HB se ha dividido en cuatro subconjuntos de instancias, teniendo en cuenta el tamaño de los grafos de entrada:

- Instancias con $|V| < 100$: 14 instancias, con número de vértices entre 24 y 96, y entre 34 y 2145 vértices.
- Instancias con $100 \leq |V| < 250$: 13 instancias, con vértices entre 100 y 245, y entre 179 y 1758 vértices.
- Instancias con $250 \leq |V| < 500$: 17 instancias, con vértices entre 256 y 494, y entre 586 y 2712 vértices.
- Instancias con $|V| \geq 500$: 17 instancias, con vértices entre 503 y 960, y entre 1282 y 3422 vértices.

Los resultados de los experimentos se presentan en cuatro tablas, una por cada grupo de instancias.

En la Tabla 4.13 se muestran los resultados obtenidos sobre las instancias con menos de 100 vértices.

$\|V\| < 100$	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	460.14	462.51	2.24	0.93	10	0.64
HPSS	460.50	465.86	2.88	**0.13**	9	0.27
APSS	**459.21**	**460.87**	**1.11**	2.36	**12**	**0.10**
CPSS	459.57	462.08	1.68	3.54	10	0.15

Tabla 4.13: Resultados obtenidos para las instancias HB con menos de 100 vértices.

En cuanto al tiempo de ejecución, HPSS logra soluciones en un tiempo de ejecución notablemente menor y con una calidad similar a la obtenida por SSS. APSS destaca como el enfoque más eficaz, encontrando el mejor valor en 12 de las 14 instancias y registrando el menor valor promedio y la menor desviación porcentual respecto al mejor resultado. CPSS, aunque no supera a SSS en número de mejores soluciones, consigue reducir la desviación media.

La Tabla 4.14 muestra los resultados para este segundo conjunto.

$100 \leq \|V\| < 250$	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	2132.31	2156.35	15.98	18.71	6	0.31
HPSS	2140.31	2178.86	31.29	**2.75**	3	0.73
APSS	**2124.85**	**2147.50**	**15.25**	46.14	**9**	**0.09**
CPSS	2125.77	2150.49	16.24	66.58	6	0.15

Tabla 4.14: Resultados obtenidos para las instancias HB con entre 100 y 249 nodos.

HPSS destaca de nuevo por su eficiencia temporal, aunque con una ligera pérdida de calidad en las soluciones. Esta variante mantiene una desviación media inferior al 1 %, lo que la convierte en una buena opción cuando se requiere rapidez sin sacrificar en exceso la calidad. APSS vuelve a ofrecer el mejor rendimiento. En este caso, CPSS también supera a SSS en número de mejores soluciones y desviación media, aunque a costa de un mayor tiempo computacional.

En la Tabla 4.15 se reflejan los resultados obtenidos al ejecutar las 4 variantes de SS sobre las instancias con entre 250 y 500 vértices.

$250 \leq \|V\| < 500$	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	6890.29	7067.24	102.01	117.22	4	1.01
HPSS	6985.41	7164.05	120.79	**16.45**	2	1.65
APSS	**6847.12**	**6978.65**	71.54	322.30	**10**	0.93
CPSS	6867.76	6983.64	**71.49**	405.74	6	**0.61**

Tabla 4.15: Resultados obtenidos para las instancias HB con entre 250 y 499 vértices.

Se observan diferencias claras en el rendimiento temporal entre SSS y HPSS. APSS vuelve a destacar como la mejor opción, logrando el mayor número de mejores soluciones y baja desviación. Este experimento demuestra que CPSS mejora notablemente frente a SSS y HPSS a medida que aumenta la complejidad de las instancias.

Finalmente, la Tabla 4.16 muestra los valores obtenidos por los algoritmos sobre las instancias más grandes del conjunto, que está formado por instancias de más de 500 vértices.

$\|V\| \geq 500$	Mejor	Promedio	Desv. Típica	T. Prom. (s)	#Mejores	Desv. (%)
SSS	16992.88	17315.11	187.75	854.42	3	0.47
HPSS	17153.35	17554.63	292.61	**114.35**	4	1.84
APSS	**16981.76**	**17222.32**	**149.95**	1447.62	6	**0.52**
CPSS	16984.53	17283.02	205.63	1718.21	**7**	0.57

Tabla 4.16: Resultados obtenidos para las instancias HB con 500 vértices o más.

Esta tabla valida la hipótesis sobre las ventajas del paralelismo. CPSS logra el mayor número de mejores soluciones, aunque APSS mantiene el mejor promedio general. HPSS, pese a no ser el mejor en calidad, mantiene tiempos de cómputo notablemente inferiores a SSS, consolidándose como alternativa eficiente en contextos donde el tiempo es crítico.

Capítulo 5

Conclusiones y trabajo futuro

En este capítulo se exponen las conclusiones obtenidas al analizar los resultados obtenidos en cada problema. El capítulo consta de cinco secciones, cuatro de ellas discutiendo las conclusiones de cada problema en el que se ha trabajado en esta tesis por separado y una última en la que se presenta la línea de investigación por la que se va a continuar tras la finalización de la tesis.

5.1. Conclusiones sobre Minimum Dominating Set Problem

El MDSP se trata de un problema combinatorio con aplicaciones prácticas como la monitorización de redes informáticas y la maximización de influencia en redes sociales. El enfoque propuesto se basa en una metaheurística de tipo IG formado por un constructivo voraz que genera una solución inicial, una búsqueda local eficiente para mejorar la solución de partida y estrategias de destrucción y reconstrucción para potenciar la exploración.

El algoritmo demuestra un alto rendimiento sobre 28 instancias de referencia y sobre los nuevos grafos generados, alcanzando los mejores valores conocidos en la mayoría de los casos con un bajo coste computacional. La fase de destrucción permite escapar de óptimos locales al explorar regiones vecinas prometedoras. Sin embargo, su eficacia puede verse limitada cuando el óptimo global se encuentra alejado de la región actual de búsqueda.

El conocimiento de los grafos y los problemas de dominación se integra en el diseño del algoritmo: los nodos soporte se incluyen obligatoriamente en la

solución, mientras que los nodos hoja se excluyen, lo cual reduce significativamente el espacio de búsqueda cuando las instancias son grandes.

Finalmente, la búsqueda local eficiente reduce el número de movimientos necesarios para alcanzar un óptimo local en comparación con un enfoque exhaustivo, mejorando así la eficiencia computacional.

5.2. Conclusiones sobre Monitor Placement Problem

El MPP se resuelve mediante el uso de la metaheurística BVNS, proponiendo una búsqueda local inteligente que aprovecha la identificación de regiones prometedoras del espacio de búsqueda para minimizar el tiempo de cómputo sin deteriorar la calidad de las soluciones obtenidas.

Se proponen dos constructivos voraces para generar una solución inicial prometedora. Ambos siguen enfoques opuestos: por un lado, *GreedyConstructive* parte desde cero y construye una solución ubicando monitores en distintos vértices de la red; por otro lado, *GreedyDestructive* parte de una solución con monitores en todos los vértices y elimina iterativamente aquellos que no son necesarios, manteniendo la cobertura total de la red.

Finalmente, se ha implementado un *intensified shake* que permite seleccionar de forma inteligente los nodos dentro de la fase de *shake* del BVNS, guiando así la búsqueda hacia regiones más prometedoras del espacio de soluciones.

Los resultados experimentales demuestran que cada componente del algoritmo propuesto tiene un efecto positivo en los resultados finales, posicionando a BVNS como un método competitivo para resolver el MPP.

5.3. Conclusiones sobre Weighted Total Domination Problem

Las principales contribuciones del artículo son de carácter metodológico. Se propone un procedimiento metaheurístico basado en la metodología JVNS, con el objetivo de obtener soluciones de alta calidad para instancias de gran

tamaño en tiempos de cómputo razonables. Además, se investigan estrategias de búsqueda eficientes, comparando el diseño básico del JVNS con una versión *multi-start*.

También se analizan diferentes estrategias para la construcción de soluciones iniciales de alta calidad, proponiendo dos algoritmos heurísticos con funciones voraces distintas, así como un algoritmo basado en la variante *Biased GRASP* de la metaheurística *Greedy Randomized Adaptive Search Procedure*. Se comparan empíricamente dos criterios de parada con el fin de evaluar su impacto en la calidad de las soluciones construidas. Además, se estudian tres variantes diferentes de búsqueda local basados en movimientos de intercambio, con el objetivo de alcanzar un equilibrio entre la calidad de la solución y el tiempo de cómputo requerido.

En resumen, esta investigación realiza contribuciones significativas en el campo de la optimización mediante metaheurísticas, especialmente en el contexto del WTDP. Los hallazgos obtenidos constituyen una base valiosa para futuros desarrollos en esta área, con aplicaciones potenciales en redes sociales y de comunicación, entre otros dominios.

5.4. Conclusiones sobre Parallel Framework for Scatter Search

Se proponen tres diseños paralelos de la metaheurística SS, cada uno enfocado en un objetivo distinto. Por un lado, el HPSS busca reducir el esfuerzo computacional del SS tradicional sin sacrificar la calidad de las soluciones. Por otro lado, el APSS incluye distintas estrategias de construcción y de búsqueda con el fin de explorar una mayor parte del espacio de búsqueda, aumentando la calidad de las soluciones sacrificando tiempo de cómputo. Finalmente, el CPSS se configura para equilibrar eficiencia y eficacia mediante la colaboración entre distintos componentes de búsqueda.

Los diseños propuestos han sido evaluados en tres problemas con representaciones de solución diferentes: el *Profile Minimization Problem*, el *Max-Cut Problem* y el *Capacitated Dispersion Problem*. Los resultados experimentales evidencian las ventajas y desventajas de cada enfoque paralelo. En particular, HPSS destaca como una alternativa veloz, por lo que resulta una gran alternativa cuando el tiempo de cómputo es limitado. Cuando no existe dicha restricción, tanto APSS como CPSS obtienen resultados competitivos en términos de

calidad, superando los resultados obtenidos por la versión secuencial reimplementada en este trabajo siguiendo las directrices de los artículos originales. La principal diferencia entre APSS y CPSS radica en los métodos constructivos y de búsqueda local que forman parte del algoritmo, lo que permite una amplia gama de configuraciones. Los experimentos demuestran que, cuanto mayor es la instancia, mejor es el rendimiento de los diseños paralelos. En consecuencia, el tamaño del problema puede considerarse un criterio clave para determinar la viabilidad práctica de desarrollar una versión paralela del SS.

5.5. Trabajo futuro

A lo largo de esta Tesis Doctoral se ha abordado el estudio de diversos problemas relacionados con la dominación en grafos, incluyendo el MDSP, el WTDP y MPP. Los enfoques propuestos, basados fundamentalmente en metaheurísticas y estrategias híbridas de búsqueda, han demostrado ser eficaces y competitivos. No obstante, aún quedan múltiples direcciones de investigación abiertas que pueden enriquecer y extender el trabajo realizado.

Una primera línea de trabajo futuro consiste en aplicar los conocimientos adquiridos, gracias a la investigación realizada durante la Tesis Doctoral de estos problemas de dominación, sobre nuevas variantes del problema de dominación como *Roman Domination Problem*, *k-Dominating Set Problem*, *Connected Dominating Set Problem* o *Independent Dominating Set Problem*.

Otra dirección prometedora se centra en la generalización y reutilización de procedimientos diseñados que han resultado eficientes, como *IntensifiedShake* o la elección de nodos soporte para formar parte de la solución inicial siempre en problemas de optimización combinatoria distintos, pero estructuralmente similares.

Además, sería interesante seguir con la línea de la metodología propuesta en esta tesis, adaptando metaheurísticas consolidadas a una variante paralelizada, creando así una base sobre la que otros investigadores puedan trabajar, evolucionando a un modelo paralelo que permita aprovechar las características de las máquinas de hoy en día.

Estas líneas futuras no solo prolongan la investigación realizada en esta Tesis Doctoral, sino que consolidan el papel de la dominación en grafos como una herramienta clave en la resolución de problemas relevantes en ciencia de datos, redes y optimización.

Parte II

Publicaciones: artículos publicados, aceptados y enviados

Capítulo 6

Variable neighborhood search approach with intensified shake for monitor placement

Los detalles del artículo y la revista en la que está publicado son:

- Casado, A., Mladenović, N., Sánchez-Oro, J., & Duarte, A. (2023). Variable neighborhood search approach with intensified shake for monitor placement. Networks, 81(3), 319-333. https://doi.org/10.1002/net.22134
- Estado: **Publicado**
- Factor de impacto (JCR 2023): 2.100
- Posición relativa: 36/54
- Cuartil: Q3

Received: 12 November 2021 | Revised: 25 October 2022 | Accepted: 8 November 2022

DOI: 10.1002/net.22134

RESEARCH ARTICLE

WILEY

Variable neighborhood search approach with intensified shake for monitor placement

Alejandra Casado[1] | **Nenad Mladenović**[2] | **Jesús Sánchez-Oro**[1] | **Abraham Duarte**[1]

[1]Department of Computer Sciences, Universidad Rey Juan Carlos, Móstoles, Spain
[2]Department of Industrial Engineering and Research Center on Digital Supply Chain and Operations Management, Khalifa University, Abu Dhabi, UAE

Correspondence
Abraham Duarte, Department of Computer Sciences, Universidad Rey Juan Carlos, Móstoles, Spain.
Email: abraham.duarte@urjc.es

Funding information
Ministerio de Ciencia, Innovación y Universidades, Grant/Award Number: PGC2018-095322-B-C22; Comunidad de Madrid, Grant/Award Number: S2018/TCS-4566; Fondos Estructurales, Grant/Award Number: Y2018/EMT-5062; Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan, Grant/Award Number: AP08856034.

Abstract

Several problems are emerging in the context of communication networks and most of them must be solved in reduced computing time since they affect to critical tasks. In this research, the monitor placement problem is tackled. This problem tries to cover the communications of an entire network by locating a monitor in specific nodes of the network, in such a way that every link remains surveyed. In case that a solution cannot be generated in the allowed computing time, a penalty will be assumed for each link uncovered. The problem is addressed by considering the variable neighborhood search framework, proposing a novel constructive method, an intelligent local search to optimize the improvement phase, and an intensified shake to guide the search to more promising solutions. The proposed algorithm is compared with a hybrid search evolutionary algorithm over a set of instances derived from real-life networks to prove its performance.

KEYWORDS

intensified shake, metaheuristics, monitor placement, variable neighborhood search

1 | INTRODUCTION

Communication networks are, nowadays, an essential part of every activity in both personal and professional points of view. Among their services we can highlight streaming, security, banking, shopping, and so forth. Originally, the security of those networks was not a requirement when designing them. However, the increase in the network attacks has made security a key factor in the network design of every company.

Although a successful attack to the network of a company may result in an important economic crisis, it must be noted that there are some critical services that must be continuously monitored, such as transport, hospitals, or defense. The failure in these kinds of services usually results in severe damages to people which may result in humanitarian crisis [1].

The number of attacks is continuously increasing, as well as the privacy on the Internet is becoming more and more relevant every year. As a result, companies and institutions need to improve the security system of their networks in order to minimize the possibilities of being attacked, thus increasing the network protection. It is worth mentioning that a fast reaction to an attack is a key factor in successfully protecting it [14].

One of the most common attacks is denial of service (DoS) and distributed denial of service (DDoS), since a successful attack can completely disable an Internet provider. The failure become critical if more services directly depend on the services under attack, which can result in a cascade failure, harming a large number of clients [6].

The early detection of potential threats such as malware spreading, loss of certain nodes in the network, unauthorized accesses to the network, among others, has a positive impact not only in the security, but also in the network performance. Therefore, it is desirable to perform detection of failures in real time to mitigate the negative effect of the attack.

One of the most important stages when increasing network protection consists of guaranteeing the surveillance of the complete network. If the network is correctly monitored, the system will be able to provide a faster reaction to an attack. This surveillance is commonly performed by deploying a monitor, which is a special hardware, in certain nodes of the network to analyze all the communications that go through them. This analysis provides the network administrators with real-time information about possible threats, which may help them to protect specific nodes.

The most simple approach to perform a complete network analysis consists of deploying a monitor in every node of the network, guaranteeing that all the communications going through the network are captured. Unfortunately, this is not possible since the cost of deploying a monitor in a node is usually high. Additionally, deploying a monitor implies an overhead in the network performance since all the traffic must be captured. Therefore, the number of monitors deployed in a network must be minimized while maintaining the complete network analyzed.

The monitor placement problem (MPP) has been widely studied in the literature, resulting in different variants which consider new aspects of the network surveillance. Although traditional approaches consider static networks, where changes are not common, in this work we are focused in networks which are in constant evolution, so the deploying of monitors must be optimized to achieve real-time analysis of the network without negatively affecting to its performance.

This problem can be reduced to a vertex covering problem, which is $\mathcal{NP}$-complete [13] and, therefore, exact approaches are not able to solve real-life instances due to their complexity. The problem remains $\mathcal{NP}$-complete for approximations within a factor smaller than 1.36 [7].

Therefore, this family of problems has been mainly tackled from a heuristic perspective. First approaches were focused on evolutionary algorithms for the maximum independent set [2] and the minimum vertex cover [9]. Later, a branch and bound and different heuristics were proposed for solving the minimum vertex cover when considering random graphs [15,16]. More approaches were then proposed for the same problem: genetic algorithms [12], simulated annealing [11], or a hierarchical Bayesian algorithm [22], among others. Milanovic [17] proposed a new genetic algorithm for solving the generalized vertex cover problem considering networks with weights in both, nodes and links. More recently, Chandu [5] designed a parallel evolutionary algorithm that leverages the hardware architecture by considering Apache Hadoop for distributing computation.

Evolutionary algorithms have been widely studied in the field of monitor placement, from a population injection method [18] to a hybrid evolutionary algorithm devoted to solve a dynamic variant of the problem [19]. As far as we know, the best algorithm in the literature for the MPP is a hybrid search heuristic [20]. This procedure is an enhancement of the population injection method originally presented in [18], whose main contribution is the increase in the diversity of the search.

In this work, we address the MPP with a variable neighborhood search (VNS) algorithm, which consists of a novel constructive method, an intelligent local search to optimize the improvement phase, and an intensified shake to guide the search to more promising solutions. The remaining of the paper is structured as follows. Section 2 formally describes the problem tackled in this research, Section 3 presents the algorithmic proposal to solve the MPP, Section 4 shows the experimental analysis performed to validate the proposal, comparing it with a hybrid search evolutionary algorithm and, finally, Section 5 draws some conclusions derived from this research.

2 | PROBLEM DEFINITION

The objective of the MPP is to monitor a complete network, which is usually modeled as an undirected graph $G = (V, E)$, where V, with $|V| = n$, is the set of nodes in the network, and E, with $|E| = m$, consists of pairs of nodes (u, v), with $u, v \in V$, indicating that there is a connection between nodes u and v.

Before defining the problem under consideration, it is necessary to introduce the concept of vertex cover. Specifically, a vertex cover Λ of a network is a subset of vertices $\Lambda \subseteq V$ which satisfies the constraint that, for every edge $(u, v) \in E$, either $u \in \Lambda$ or $v \in \Lambda$. Then, the minimal vertex cover is the one with the minimum size. More formally, the minimum vertex cover problem is defined as follows:

$$MVCP(G) = \min_{\Lambda \in C} |\Lambda|,$$

where C is the set of all possible vertex covers for the network G. Since a network can have several different minimum vertex covers (naturally, with the same value of objective function), we refer to the MVCP as the problem of finding one of those vertex covers.

The MPP is a particular variant of the MVCP but with direct application in communication networks. In this context, there are some additional features that must be satisfied in order to solve the MPP. The first one is related to the computing time required to solve the problem. In real-life networks, the time window available to solve the MPP is considerably smaller than in the classic MVCP. The rationale behind this consideration is the immediacy with which any anomalous network behavior must be solved, with the aim of activating or deactivating any node producing the anomaly in the monitored network. In case that the time feature cannot be satisfied then, the budgeted time should be used for finding a solution which monitors the largest number of connections in the network. In this case, not all the connections are equally relevant, so it is necessary to prioritize the links to be monitored [3].

A network is considered to be fully monitored if and only if all the links are covered by, at least, one monitor. The optimal solution is the one that covers the complete network while minimizing the impact in the performance. Therefore, the MPP looks for monitoring the complete network but placing the minimum number of monitors in it. For the sake of simplicity, we consider that a monitor can be placed in any node of the network.

The definition of priority for a link in the network completely depends on its nature. For instance, it can be related to the traffic flow through the link, its bandwidth, the relevance of the link in the real network, among others. We refer the reader to [21,25] for a detailed analysis of setting network priorities.

The inclusion of link priorities prevents the MPP to be reformulated as the MVCP. However, as stated in [20], the priorities can be easily included in the problem model by considering a priority function $p : E \rightarrow \mathbb{N}$ that assigns a penalty to each uncovered link in a given solution. Therefore, two solutions with the same number of monitors can be compared by computing the penalty due to the uncovered links.

The objective function for the MPP is then conformed by two different elements: the number of monitors that are deployed, and the accumulated penalty due to the uncovered links. A solution S (with $S \subseteq V$) for the MPP contains the nodes in which a monitor should be deployed. Given a solution S, the objective function of the MPP is evaluated as:

$$MPP(S) = |S| + \sum_{(u,v)\in E'} p(u,v),$$

where $E' = \{(u,v) \in E : u \notin S \wedge v \notin S\}$, that is, the set of all edges of the network which are not covered by any monitor. Then, the objective of the MPP is to find the solution $S^\star$ with the minimum value of the objective function. More formally,

$$S^\star = \arg\min_{S\in\mathbb{S}} MPP(S),$$

where $\mathbb{S}$ represents the solution space of the problem, consisting in all the feasible solutions for the MPP. It is worth mentioning that any subset of nodes conforms a feasible solution. In particular, $|S| = |V|$ means that a monitor is deployed in every single node. Then, the first addend takes on the largest value while the second addend is zero. Similarly, $|S| = 0$ means that no monitor is deployed. Therefore, the first addend is zero, while the second one acquires its maximum value.

The objective function balances both features: the number of monitors included in the solution and the penalties associated with those links which are not covered. Although there are several types of penalties that can be considered for the network, we follow the same approach as in [20], which considers a static linear distance penalty function (see Section 4 for the penalty included in the considered instances).

Figure 1 shows a network instance consisting of 5 nodes, $V = \{1,2,3,4,5\}$ and 5 edges, $E = \{(1,2),(1,4),(2,3),(2,5),(3,5)\}$. The number close to each edge indicates the penalty assumed if this link is not covered. Figure 2 represents three different solutions for Figure 1. In all of them, the nodes where a monitor is deployed are colored, while the links which are

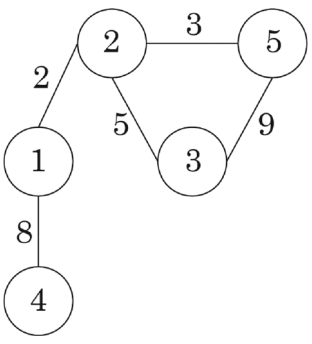


FIGURE 1 Example instance with 5 nodes and 5 edges

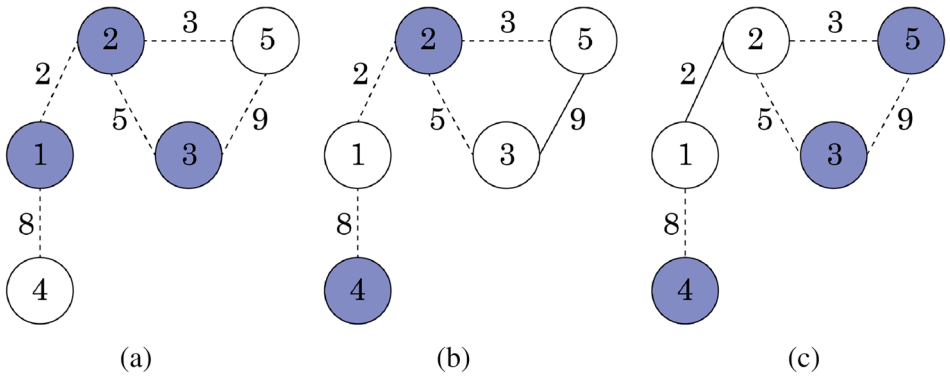


FIGURE 2 Three possible solutions for the instance depicted in Figure 1. (A) Solution $S_1 = \{1, 2, 3\}$, (B) solution $S_2 = \{2, 4\}$, and (C) solution $S_3 = \{3, 4, 5\}$

covered by, at least, one monitor, are represented with a dashed line. Figure 2A depicts solution S_1, selecting nodes 1, 2, and 3, which are able to cover the entire network. Therefore, the objective function value is evaluated as $MPP(S_1) = 3 + 0 = 3$, since there are 3 nodes in the solution and there is no penalty (all links are covered). If we now analyze Figure 2B, there are two selected nodes (2 and 4) instead of three, but the link $(3, 5)$ is not covered by any deployed monitor. Then, the objective function value is $MPP(S_2) = 2 + 9 = 11$. Finally, solution S_3 deploys a monitor in nodes 3, 4, and 5, but the link (1,2) is still uncovered, resulting in an objective function value of $MPP(S_3) = 3 + 2 = 5$. Regarding these results, the best solution is S_1, which provides the smallest objective function value. It is interesting to analyze that S_3 outperforms S_2 even having selected more nodes to deploy a monitor. In the context of MPP, a solution with all the links covered is always better than a solution with at least one link uncovered but, due to time constraints, it is not always possible to cover all the nodes.

3 | ALGORITHMIC APPROACH

In contrast to most of the previous works, which followed approaches based on evolutionary algorithms, this paper tries a novel strategy where a trajectory-based metaheuristic is proposed. While population-based metaheuristics, such as evolutionary algorithms, maintain a population of solutions which are combined to continue improving, trajectory-based metaheuristics work with a single solution which is iteratively improved following different stages.

Specifically, this work is focused on VNS as metaheuristic framework for solving the MPP. The success of VNS relies on systematic changes of neighborhoods with the aim of obtaining high quality solutions in small computing times. In the last decades, VNS has been in continuous evolution, giving rise to a large variety of schemes, which are usually classified depending on the neighborhood exploration. In particular, the neighborhood proposed can be explored by following three different criteria: deterministic (variable neighborhood descent), stochastic (reduced VNS), or mixed (basic VNS). Additional variants have also been presented: General VNS, variable neighborhood decomposition search, variable formulation search, among others. See [10] for further details.

This work considers the basic VNS (BVNS) scheme, which combines the intensification of the local improvement stage with the diversification introduced by the perturbation method. Finding a balance between both, intensification and diversification, will eventually lead to high quality solutions. Algorithm 1 shows the pseudocode of the proposed BVNS.

The method requires three input parameters: the initial solution S (see Section 3.2 for more details), the maximum neighborhood to be explored during the search $k_{\max}$, and the maximum number of iterations performed δ. BVNS is usually executed either a certain number of iterations or for a fixed computing time. In this case, we fix the number of iterations δ (steps 1–13). In each iteration, the algorithm starts with the first neighborhood (step 2). Then, BVNS iterates until reaching the largest predefined neighborhood $k_{\max}$ (steps 3–12). For each neighborhood, the solution is perturbed following the shake procedure described in Section 3.4 (step 4). The perturbed solution S' is then improved using the method described in Section 3.3 to find a local optimum in the current neighborhood (step 5). Finally, BVNS performs the neighborhood change stage. In particular, if the improved solution S'' outperforms the best solution found so far (step 6), the method restarts the search from the first neighborhood (step 7), updating the best solution found (step 8). Otherwise, the algorithm continues with the next neighborhood (step 10). BVNS ends when performing δ iterations, returning the best solution found during the search (step 14).

Algorithm 1. *BVNS*$(S, k_{\max}, \delta)$

for $i \in 1 \dots \delta$ **do**
  $k \leftarrow 1$
  while $k \leq k_{\max}$ **do**
    $S' \leftarrow Shake(S, k)$
    $S'' \leftarrow Improve(S')$
    if $MPP(S'') < MPP(S)$ **then**
      $k \leftarrow 1$
      $S \leftarrow S''$
    else
      $k \leftarrow k + 1$
    end if
  end while
end for
return S

3.1 | Support and leaf nodes

Before describing in detail each part of the proposed algorithm, we need to introduce the concept of support nodes, which allows us to reduce the search space. Specifically, there are some nodes that are always in the optimal solution due to the structure of the graph. We denote these nodes as support nodes.

Given a network $G = (V, E)$, we denote $LN \subseteq V$ as the set of leaf nodes which are connected to a single node in the graph (i.e., the degree of the node is equal to 1). The nodes to which the leaf nodes are connected are named as support nodes SN. Considering the example depicted in Figure 1, $LN = \{4\}$ and $SN = \{1\}$, respectively. Then, if we need to monitor all the links in a network, there are only two options to cover the link in which a leaf node is involved: deploy a monitor in the leaf node, or deploy it in its support node. It trivially holds that deploying a monitor in the leaf node always result in an equal or worse solution, since it only covers the link in which the leaf node is involved. Therefore, one optimal solution will always contain a monitor in every support node.

3.2 | Initial solution

VNS requires an initial solution to start the search. This solution can be generated either at random or using a more elaborated procedure. Although the basis of VNS indicates that the initial solution is not relevant for the algorithm, it has been experimentally tested that a high quality initial solution usually helps the algorithm to provide better solutions in smaller computing times [24]. Therefore, in this work, we propose two greedy constructive procedures to provide BVNS a good starting point in the search space.

The first constructive procedure follows a traditional greedy approach which starts from an empty solution and, iteratively, adds elements to it until reaching a feasible solution. Algorithm 2 shows the pseudocode of the proposed constructive procedure.

Algorithm 2. *GreedyConstructive*(G)

$S \leftarrow \emptyset$
DeploySupportNodes(S, G)
$UL \leftarrow \{(u, v) \in E : u \notin S \wedge v \notin S\}$
$C \leftarrow V \setminus (LN \cup SN)$
while $UL \neq \emptyset$ **do**
  $c \leftarrow \arg\max_{u \in C} \sum_{\substack{(u,v) \in E \\ v \in C}} p(u, v)$
  $S \leftarrow S \cup \{c\}$
  $C \leftarrow C \setminus \{c\}$
  $UL \leftarrow UL \setminus \{(c, x) : \forall x \in N(c)\}$
end while
return S

The algorithm starts from an empty solution S (step 1) and deploy a monitor in every support node (step 2). Then, the list of uncovered links UL is created with every edge in the graph which is not covered by any monitor (step 3). The candidate nodes to host a monitor are those which are neither a leaf node nor a support node (step 4). The method now iterates until covering all the links in the network (steps 5–10). In each iteration, the next candidate node is selected as the one which is able to minimize the penalty that affects to the solution under construction (step 6). In other words, we only consider (by summing up the corresponding penalty) edges whose both endpoints belong to C.

The selected candidate c is then added to the solution (step 7), updating the set of candidates (step 8), removing from the set of uncovered links UL all edges in which c is involved (step 9). Notice that $N(c)$ refers to the adjacent nodes to c. The method ends returning a solution S where all the links are covered (step 11).

The second constructive method, named *GreedyDestructive*, follows a destructive approach. In particular, if all links need to be covered in a network, it is expected that a large number of nodes would finally be selected. Therefore, it would be interesting to start from a solution with a monitor deployed in every node but in the set of leaf nodes LN and, iteratively, remove a monitor from a node while all the links remain covered. This destructive approach is useful for those hard optimization problems in which the size of the expected solution consists of selecting the majority of available elements, since it requires less iterations than considering a traditional constructive procedure. Therefore, it can be easily adapted to several combinatorial optimization problems where the objective is to select an eventually large subset of elements.

In each iteration, a monitor can be removed from the node v if and only if it is not a support node (i.e., $v \notin SN$) and all its adjacent nodes have a monitor deployed (i.e., $u \in S\ \forall u \in N(v)$). Among all the monitors that can be removed, we need to select the most promising one. Specifically, we select the node with the smallest degree. The rationale behind this is that the smaller the degree, the smaller the number of links covered by a monitor deployed in that node. The method stops when no monitor can be removed without leaving a link uncovered. For the sake of brevity, we omit the inclusion of the corresponding pseudocode since it is symmetrical to the one reported in Algorithm 2.

3.3 | Improvement method

The local improvement phase in VNS is responsible for finding a local optimum with respect to certain neighborhood. It can be as complex as desired: from a straightforward hill-climbing local search method (see [4]) to a more elaborated metaheuristic such as tabu search, or VND, among others (see [8,23] for some successful applications of using complex metaheuristics as improvement). In this research, we propose the use of a simple yet effective local search heuristic to reach a local optimum with a relatively small computational effort, which is a critical part of the problem.

The definition of a local search method requires three main elements: the movement used in the local search, the neighborhood of solutions that can be generated with the movement, and the strategy selected to traverse that neighborhood.

First of all, it is necessary to define the movement considered for this improvement phase. Notice that the goal of this movement is to reach a solution of better quality than the original one, so we need to guarantee that the movement can produce an improvement. In the context of MPP, since the objective function value is computed as the number of deployed monitors plus the penalty, it is desirable to reduce the number of deployed monitors without increasing the penalty. To that end, we propose a movement named *Exchange2x1* which removes two deployed monitors and locate a new one. According with Section 3.1, support nodes are always in the solution, while leaf nodes are never considered in the solution. Therefore, this move can be mathematically described as follows:

$$Exchange2x1(S,u,v,w) \leftarrow (S \setminus \{u,v\}) \cup \{w\},$$

where $u, v \in (S \setminus SN)$, $w \notin S$, and $w \in (V \setminus LN)$. In order to simplify the notation, we denote as $\overline{S} = S \setminus SN$ and $\overline{V} = V \setminus LN$.

Having defined the movement used in this phase, the neighborhood of a solution S is conformed with those solutions that can be reached by performing a single *Exchange2x1* move over S. More formally:

$$N_{2x1}(S) \leftarrow \{S' \leftarrow Exchange2x1(S,u,v,w) : \forall u, v \in \overline{S} \wedge w \in (\overline{V} \setminus S)\}.$$

With the movement and neighborhood defined, it is now necessary to indicate the strategy used to traverse the neighborhood. We propose two different strategies, resulting in two independent local search methods: an exhaustive local search and an intelligent local search. The main difference between both strategies is that the latter only considers those movements that do not leave any link uncovered.

The exhaustive local search, named ELS, randomly traverses the neighborhood N_{2x1} by removing each pair of nodes and inserting a new one to replace them. Then, if the resulting solution outperforms the original one, the best solution found is updated, resulting in a first improvement strategy. Otherwise, the movement is undone, continuing the search with new

candidates. As can be expected, this strategy performs a vast number of operations, resulting in a nonefficient improvemen t phase (see Section 4 for a detailed analysis).

With the aim of providing an efficient strategy to explore the neighborhood, we propose an intelligent local search, named ILS, that carefully selects both, the two nodes to be removed and the one to be included. The goal of this strategy is to reduce the number of operations to reach a local optimum.

We first consider a preprocessing strategy in charge of identifying extra monitors whose removal does not leave any link uncovered and, therefore, they are not necessary in the solution. In general, given a solution S and a node $u \in S$, whose neighborhood $N(u) \subseteq S$, then, u can be simply removed from S.

The next step in the ILS strategy consists of selecting the nodes involved in the movement. The goal is to select only nodes that drive to an improvement in the objective function. In order to do so, the first node to be removed u is selected at random. Thus, it restricts the other two nodes involved in the movement: v, the other node to be removed; and w, the node which will be included in the solution.

We first identify those monitors whose removal do not drive to an improvement. Formally, given a solution S and any node $u \in S$, such that $|N(u) \setminus S| > 1$, independently of the election of a node v, it is impossible to find a single node w able to cover those links previously covered by u and v.

Therefore, given a solution S, a node $u \in S$ is a promising candidate to be removed if $|N(u) \setminus S| = 1$ (notice that the situation $|N(u) \setminus S| = 0$ was considered in the preprocessing strategy). The candidate to be incorporated in the solution is the only node w that remains in the set $N(u) \setminus S$. Finally, this move is actually an improvement move if we are able to identify a node $v \in (S \cap N(w))$ such as $N(v) \setminus S = \{w\}$.

ILS traverses the neighborhood performing just those moves that guarantee an improvement, thus ignoring those movements that lead to nonimproving solutions. This approach is suitable for those optimization problems in which it is possible to rank the moves evaluated in the local search method. The computational results will show the relevance of performing this study on which elements are the most promising ones, becoming a strategy to be taken into account in similar problems.

3.4 | Shake

The shaking phase in VNS is intended to allow the algorithm to escape from local optima when the local search gets stuck. This component introduces diversification in the search by perturbing the incumbent solution, resorting to a different solution in the neighborhood under exploration. To that end, a parameter k controls the perturbation size. This parameter is responsible for increasing the perturbation size when no improvement is found with the aim of exploring further regions of the search space. On the contrary, this method reduces the perturbation size when finding improvements so as to focus on promising regions of the search space.

The perturbation is performed in two stages, by using two new movements: *Drop* and *Add*. The former removes a monitor deployed at node u in solution S producing a new solution S'. In mathematical terms,

$$Drop(S, u) \leftarrow S \setminus \{u\}.$$

On the contrary, the *Add* movement, denoted as $Add(S, u)$, deploys a monitor in the node u. More formally,

$$Add(S, u) \leftarrow S \cup \{u\}.$$

The first stage of the shake procedure randomly selects k nodes where a monitor is deployed and performs the *Drop* move over them. Then, in the second stage, monitors are deployed using the *Add* move until all the links are covered. We propose two different strategies to perform the second stage, resulting in two shake procedures.

On the one hand, the first strategy, named *RandomShake*, follows the traditional VNS scheme by randomly selecting k nodes that are not in S and deploying a monitor in them with the *Add* movement. Notice that this strategy usually produces solutions in which there are more monitors than necessary since the candidate nodes to host a monitor are selected completely at random.

On the other hand, we propose a new shake method, named *IntensifiedShake*, that performs this second stage intelligently, choosing the most promising nodes to host a monitor. It is necessary to establish a criterion to decide which the next node selected to host a monitor is. In this work, we propose to use the same criterion as the one used in *GreedyConstructive*, that is, the selected nodes are those whose links that are not covered generate the largest penalty (always excluding the leaf nodes). This second strategy has not been widely explored in the literature, but it is an interesting approach since it is able to find promising regions of the search space without compromising the diversity of the search, thus leading to more promising solutions. Therefore, it is a strategy to be considered in those optimization problems where it is possible to consider a greedy criterion in the shake procedure.

4 | COMPUTATIONAL RESULTS

This section has two main goals. First of all, it is necessary to adjust the input parameters of the proposed algorithm, as well as the algorithmic strategies that conform the final method. Having configured the best variant of the proposed procedure, the second objective is to perform a competitive testing considering the state-of-the-art method for the MPP.

All the algorithms have been implemented in Java 11 and the experiments have been conducted in an AMD EPYC 7282 (2.8 GHz) and 72 GB RAM. We have considered the same set of instances as the ones presented in the related literature, where the best algorithm for the MPP is introduced [20]. From the complete set of instances, we have not been able to consider `delaunay_n19` and `delaunay_n20` due to hardware constraints, since our computer does not have enough memory to store them (although the algorithm would be able to solve them). However, the results obtained with the largest instances will show how the improvements obtained by the proposed algorithm increase with the size of the instance. The final set of instance consists of 37 instances. In order to favor future comparisons, we have made publicly available these instances at https://grafo.etsii.urjc.es/MonitorPlacement.

All the experiments report the following metrics: Avg., the average objective function value obtained by each algorithm among the considered instances; time (s), the average computing time required by each algorithm to finish, measured in seconds; Dev (%), the average deviation of the objective function value with respect to the best solution found during the experiment, evaluated as $\frac{|Best-Obj.F.|}{Best}$ for each instance; and # Best, the number of times that the algorithm reaches the best solution of the algorithm in the experiment. In all the experiments, the best result for each metric is highlighted in bold font.

In order to select the best values for each parameter, some preliminary experiments have been executed. In all of them (from Tables 1–5), 10 representative instances have been used to configure the proposed algorithm in order to avoid overfitting. Those instances are: `delaunay_n10`, `delaunay_n11`, `frb30-15-3`, `frb35-17-1`, `frb40-19-2`, `p2p-Gnutella04`, `p2p-Gnutella05`, `p2p-Gnutella06`, `tech-routers-rf`, `tech-WHOIS`.

The first preliminary experiment is intended to evaluate the performance of the proposed constructive methods, *GreedyConstructive* and *GreedyDestructive*. Since they are greedy algorithms, we have constructed a single solution for each instance using each constructive procedure, as presented in Table 1.

As suggested in Section 3.2, the destructive procedure is considerably faster than the constructive one, since the final solutions contain a monitor deployed in most of the nodes. Therefore, the number of nodes to be removed when considering a monitor in each node (*GreedyDestructive*) is rather smaller than the number of nodes to be added when considering an empty initial solution (*GreedyConstructive*). However, in terms of quality, the solutions generated with *GreedyConstructive* are slightly better than the ones created with *GreedyDestructive*, as indicated by the small deviation presented by the *GreedyDestructive* procedure. Analyzing these results, it is not possible to determine which is the best constructive procedure, so in a latter experiment the performance of the constructive procedures coupled with the local search method will be tested.

TABLE 1 Results obtained when creating a single solution for each instance by each constructive procedure

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GreedyDestructive	1844.90	**0.19**	0.52	2
GreedyConstructive	**1837.80**	1.82	**0.04**	**8**

TABLE 2 Heat map of the computing times (left) and average deviation (right) when considering different values of δ and k_{max} for *RandomShake*

	k_{max}				
δ	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
1	0.63	0.93	1.86	2.41	3.92
10	2.17	6.02	11.28	15.02	27.95
20	4.04	12.12	21.56	29.29	54.20
30	6.04	17.84	31.70	42.69	80.46
40	7.67	22.96	85.93	82.07	107.57
	k_{max}				
δ	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
1	0.23	0.23	0.22	0.22	0.22
10	0.10	0.15	0.07	0.10	0.05
20	0.02	0.10	0.05	0.09	0.05
30	0.02	0.10	0.05	0.04	0.05
40	0.02	0.09	0.05	0.02	0.05

TABLE 3 Heat map of the computing times (left) and average deviation (right) when considering different values of δ and k_{max} for *IntensifiedShake*

	k_{max}				
δ	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
1	0.20	0.26	0.57	1.15	2.04
10	0.19	0.99	3.15	6.05	12.74
20	0.20	1.89	5.75	11.79	22.25
30	0.23	2.56	8.50	17.91	33.51
40	0.24	3.22	10.93	27.10	45.64
	k_{max}				
δ	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
1	0.36	0.34	0.32	0.12	0.12
10	0.36	0.17	0.07	0.11	0.06
20	0.36	0.09	0.03	0.11	0.06
30	0.33	0.09	0.03	0.10	0.06
40	0.33	0.09	0.03	0.10	0.03

TABLE 4 Comparison of both shake strategies for the BVNS algorithm with the corresponding best values for each search parameter

Shake method	Avg.	Time (s)	Dev. (%)	#Best
IntensifiedShake	**1816.20**	**1.89**	**0.01**	**9**
RandomShake	1818.80	4.04	0.22	6

TABLE 5 Comparison of the contribution of each component of the algorithm proposed

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GreedyDestructive	1844.90	**0.19**	1.89	0
GreedyDestructive+ILS	1819.30	0.21	0.26	4
BVNS	**1816.20**	1.89	**0.00**	**10**

The second experiment evaluates the influence of considering the intelligent local search ILS instead of executing the exhaustive local search ELS, both described in Section 3.3. Figure 3 shows the comparison of the computing times for executing the ILS and ELS over the solution generated by the *GreedyConstructive* procedure to decide if it is worth considering ILS over ELS.

Notice that the figure is represented using a logarithmic scale due to the vast differences between the computing times of both local search methods. It is worth mentioning that both local search methods result in solutions of similar quality, so we only analyze computing times. Regarding these results, we can clearly see the positive impact of considering the intelligent local search. On average, it is 2500 times faster than ELS, reaching a speedup of 5200 in the most complex instances. In particular, notice that ILS does not require more than 10 s in any of the considered instances. Furthermore, in the most complex instances, namely p2p-Gnutella04, p2p-Gnutella05, and p2p-Gnutella06, the ELS has been stopped when reaching 100 000 s, highlighting the relevance of considering the intelligent local search procedure. Therefore, we consider ILS as the local search method for the remaining experiments.

The next experiment tries to settle which the most adequate constructive procedure for the MPP is. As stated above, it is not clear which the best method to generate the initial solution is, so we test the performance of both greedy procedures executed isolated and when they are coupled with ILS. Figure 4 shows the relation between computing time and average deviation when considering these four procedures.

The first conclusion that can be extracted from this experiment is that ILS considerably improves the generated solution without requiring a high computational effort in both greedy procedures. Additionally, we can see how *GreedyDestructive+ILS* is able to outperform the results provided by *GreedyConstructive+ILS*, requiring much less computing time. Therefore, *GreedyDestructive* procedure is used as the method for generating the solution used as starting point for the complete BVNS algorithm. These results indicates that it is interesting to consider a destructive approach which starts with all elements selected in those optimization problems in which high-quality solutions have most of the elements selected, since it is able to considerably reduce computing time without deteriorating the quality of the solution. Additionally, the effect of ILS with respect to ELS suggests that it is recommended to include information about the structure of the solution in the local search if it is available.

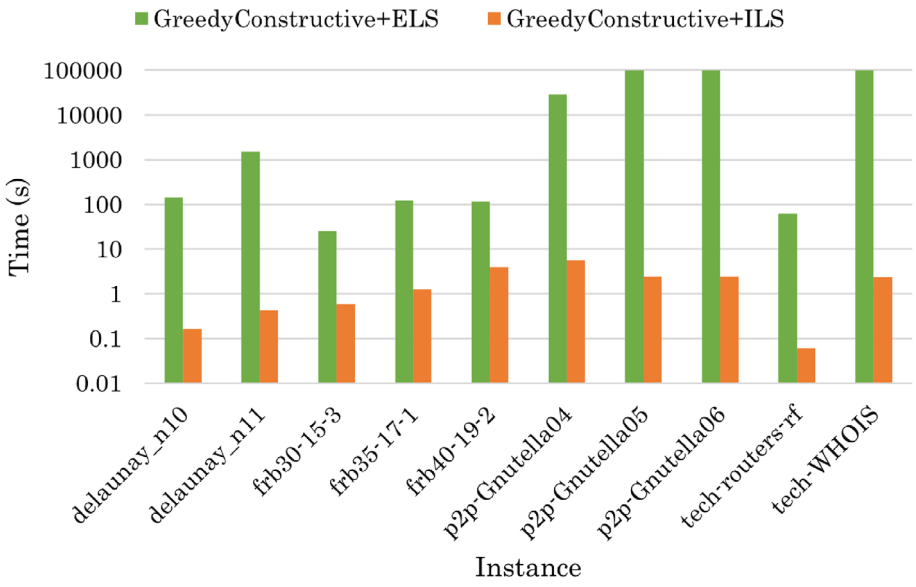


FIGURE 3 Comparison of the time required by the local search and the efficient variant proposed for each instance in the preliminary set

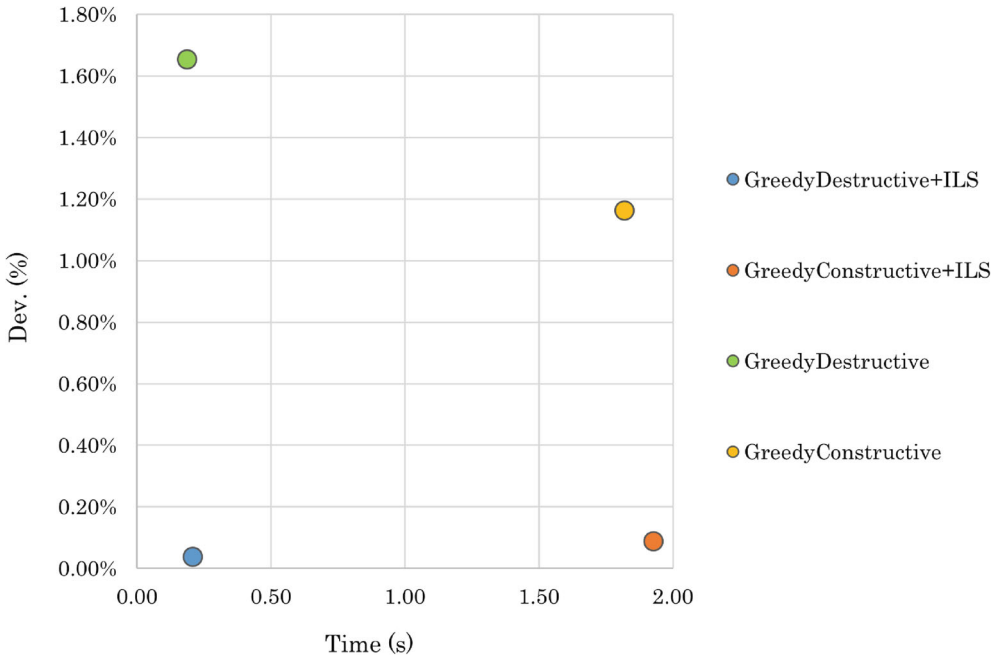


FIGURE 4 Comparison of the time and the average deviation between both construction strategies isolated and then considering ILS

Having defined the constructive procedure and the local search method, we need to tune the values for the input parameters of BVNS. The algorithm requires just two parameters: δ, which is the number of complete VNS iterations, and $k_{\max}$, the maximum neighborhood to be explored in each complete iteration. The values tested for the largest neighborhood are $k_{\max} = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, where each value represents a percentage of the number of nodes of the instance to guarantee the scalability of the proposal. We do not consider larger values of $k_{\max}$ since perturbing more than half of the solution will result in a completely different one, which is against the philosophy of the VNS framework. Regarding the number of iterations, we test $\delta = \{1, 10, 20, 30, 40\}$.

This work proposes two different shake methods, *RandomShake* and *IntensifiedShake*, and the best values for these parameters for one shake method are not necessarily the best values for the other shake method. Therefore, we have conducted two equivalent experiments varying the shake method considered. Since modifying one of the parameters might affect to the other one, we have decided to show the results obtained in two heat maps where the worst values are colored in red and the best values in green, interpolating the values between them using a color gradient. Table 2 shows the results obtained by considering *RandomShake*.

Analyzing the heat map of computing times (left), as expected, it grows with the number of iterations and maximum neighborhood explored, being two orders of magnitude slower than the fastest option ($k_{\max} = 0.1$ and $\delta = 1$) in some cases. If we simultaneously analyze the heat map of average deviation (right), we can clearly see that the best values are obtained when considering small values of $k_{\max}$, specifically $k_{\max} = 0.1$, and the number of iterations stagnates when reaching 20. Since the computing time of $k_{\max} = 0.1$ and $\delta = 20$ is also one of the smallest in the experiment, we select these parameter values when considering *RandomShake*. Table 3 now shows the same experiment but considering *IntensifiedShake*.

Analogously to the previous experiment, the computing time also gets increased with the value of $k_{\max}$ and δ, although the differences are not so remarkable. Regarding the average deviation, the best values are obtained with $k_{\max} = 0.3$, which indicates that the intensified shake is able to increase the size of the maximum neighborhood without deteriorating the quality of the generated solutions. With respect to the number of iterations, it seems that performing more than 20 iterations does not result in any improvement, so we select $k_{\max} = 0.2$ and $\delta = 20$ for *IntensifiedShake*. The rationale behind this is that, although considering the next $k_{\max}$ value reduces the deviation from 0.09 to 0.03, we do not believe that it is a significant improvement for multiplying the computing time by 3.

Having defined the best parameter values for $k_{\max}$ and δ for both *RandomShake* and *IntensifiedShake*, it is necessary to compare the performance of each shake method in the BVNS algorithm. Table 4 shows the results obtained by both variants.

As it can be seen in the results, *IntensifiedShake* requires less computing time than *RandomShake*. This behavior can be partially explained since the solution to which the local search is applied is closer to a local optimum in the case of *IntensifiedShake* than in *RandomShake*, reducing the number of iterations required to reach the local optimum. Additionally, *IntensifiedShake* is able to outperform *RandomShake* in number of best solutions and in average deviation. Notice that *IntensifiedShake* misses just one best value, remaining really close to it as derived from a deviation of 0.01%. It is worth mentioning that *RandomShake* is rather competitive in this case, but we select *IntensifiedShake* for being slightly better in all the metrics requiring considerably less computing time than *RandomShake*. In this case, the greedy selection of elements in *IntensifiedShake* is able to reduce computing time without deteriorating the objective function value, which highlights the relevance of considering this strategy for optimization problems in which it is possible to define a greedy criterion to select an element to be included in the solution.

The last preliminary experiment is designed to evaluate the contribution of each component of the final algorithm to the quality of the generated solutions. To that end, we compare the constructive method isolated, then coupled with the intelligent local search and, finally, the complete *BVNS* algorithm, configured with *GreedyDestructive*, *ILS*, *IntensifiedShake*, $k_{\max} = 0.2$, and $\delta = 20$. Table 5 shows the results obtained in this experiment.

It is worth mentioning that the efficient design of each component leads *BVNS* to require comparable computing time than the constructive procedure isolated or coupled with the local search method. Although ILS allows the algorithm to find 4 best solutions, it is the complete *BVNS* algorithm the one that is able to reach all the best solutions. However, the small deviation achieved by the combination of constructive and local search methods result in a simple yet effective algorithm to provide high quality solutions in negligible computing times. These results show the contribution of each part of the proposed algorithm to the final version.

At this point, all the components and parameters of the *BVNS* algorithm have been fixed, so the next experiment is devoted to evaluate the performance of *BVNS* when comparing it with a hybrid search evolutionary algorithm (LS+PI) EA, one of the most widely used algorithms found in the state of the art [20], which consists of an effective hybrid search heuristic that leverages the combination of the greedy local search method with evolution-based heuristics. The complete set of 37 instances is considered in this final experiment, grouping them by type. Table 6 shows the results obtained when considering the complete testbed of instances. Notice that the column Median O.F. reports the median objective function value obtained along 100 executions for each instance, considering the same experimental methodology described in [20]. It is worth mentioning that the hardware used to perform the experiments in this research is directly comparable with the one considered in the state of the art. In particular, the score given in a CPU benchmark to each processor is the same (1.9).[1]

Results in Table 6 show how *BVNS* is able to find 37 best solutions, that is, in each type of instances the algorithm reaches the best solution for every instance. If we now look at the time required by each algorithm, in the type of instances where the difference is the largest (frb), *BVNS* is 445 times faster than *(LS+PI) EA*. Looking at the average deviation obtained by each algorithm in each type of instances, it can be seen that *BVNS* value is always 0.0%, while the state-of-the-art algorithm has an average deviation of approximately 17.0% taking into account all types of instances, reaching the maximum value in `delaunay` (40.47%) and `internet-as` (33.22%) instances, and the minimum value in `frb` instances (0.66%). Notice that, in this last type of instances, the difference in time between the two algorithms is the greatest one. It is worth mentioning that the largest differences in deviation are obtained in the most complex instances, highlighting the scalability of the proposed algorithm. Analyzing these results, we can conclude that the proposed BVNS is a competitive algorithm for solving the MPP, requiring a reduced computational effort even for the most complex instances, we refer the reader to Table A1 to view individual results.

[1] https://www.cpubenchmark.net/compare/Intel-Xeon-E5-2690-v4-vs-AMD-EPYC-7282/2780vs3625

TABLE 6 Comparison of the hybrid search evolutionary algorithm (LS+PI) EA and the proposed BVNS for each instance type

Instance type	Algorithm	Median O.F.	Avg. time (s)	Dev. (%)	#Best
delaunay	***BVNS***	**41 311.72**	**272.21**	**0.00**	**9**
	(LS+PI) EA	100 602.33	475.78	40.47	0
frb	***BVNS***	**584.67**	**0.56**	**0.00**	**15**
	(LS+PI) EA	588.67	252.56	0.66	0
internet-as	***BVNS***	**5701.00**	**19.95**	**0.00**	**1**
	(LS+PI) EA	7595.00	179.18	33.22	0
NREN	***BVNS***	**422.00**	**0.05**	**0.00**	**1**
	(LS+PI) EA	452.00	2.02	7.11	0
p2p-Gnutella	***BVNS***	**6001.67**	**6.49**	**0.00**	**9**
	(LS+PI) EA	7966.67	71.95	24.58	0
tech	***BVNS***	**1540.50**	**1.98**	**0.00**	**2**
	(LS+PI) EA	1658.50	32.08	7.33	0

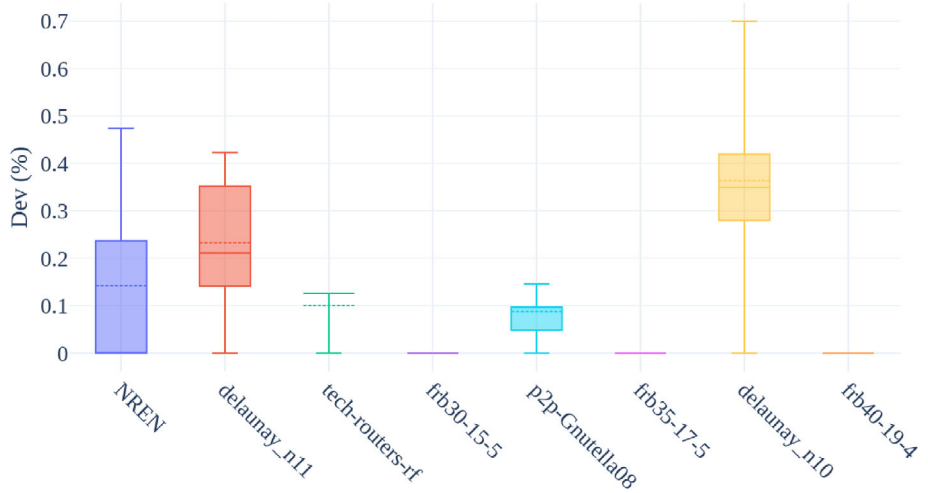


FIGURE 5 Box-plots for some representative instances to show the variability of the proposed algorithm

In order to evaluate if there are statistically significant differences between the results obtained by both algorithms, the pairwise nonparametric Wilcoxon Signed Ranks statistical test is performed. The resulting p-value smaller than 0.001 supports the hypothesis that both samples belong to different populations.

Finally, to evaluate the effect of the randomness in the proposed algorithm, Figure 5 shows the box-plot obtained for the 100 executions of BVNS. In order to present all the box-plots in the same figure, the reported values are the deviation with respect to the best value of the 100 executions per instance. The considered instances are the same as the ones presented in the best previous work [20] to facilitate the comparison. As it can be seen in the figure, most of them present minimum or even zero variability, supporting the robustness of the proposal. The largest variability is obtained in the most complex instances, where the objective function value varies in a range of five units, which is negligible when considering the magnitude of the objective function value. We refer the reader to Table A2 in the Appendix for the individual results on every instance.

5 | CONCLUSIONS

This work deals with the MPP, which tries to locate monitors in a network in such a way that all the communications are surveyed to protect the network from external attacks or failures. Since it is a critical task, it requires a fast response, so the proposed algorithm must be able to perform the monitoring without requiring large computing times. The problem is addressed by considering the BVNS framework, proposing an intelligent local search that is able to leverage the identification of promising regions of the search space to minimize the impact in the computing time without deteriorating the quality of the solutions provided.

Furthermore, two greedy procedures are proposed to produce a promising initial solution. Both methods follow opposite directions: on the one hand, the *GreedyConstructive* starts from scratch and constructs a solution by locating monitors; on the other hand, a novel approach is followed by *GreedyDestructive*, which initially considers that a monitor is deployed in every

node and iteratively removes monitors while maintaining the network covered. Finally, an intensified shake is proposed, which is able to perform an intelligent selection of the nodes inside the shake procedure of VNS with the aim of guiding the search to a more promising region of the search space. The experimental results show how every component of the proposed algorithm has a positive effect in the final results, emerging BVNS as a competitive method for solving the MPP.

Future lines of research comprehend the study of the strategies presented in this work when applying it to problems related to the monitor placement but with additional constraints, such as weighted nodes, fault tolerance, and so forth, to evaluate the adaptation of the proposed algorithm.

ACKNOWLEDGMENTS

Alejandra Casado, Jesús Sánchez-Oro, and Abraham Duarte research was funded by "Ministerio de Ciencia, Innovación y Universidades" under grant ref. PGC2018-095322-B-C22, "Comunidad de Madrid" and "Fondos Estructurales" of European Union with grant refs. S2018/TCS-4566 and Y2018/EMT-5062. Nenad Mladenović has been partially supported by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan, Grant No. AP08856034.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in MonitorPlacement at https://grafo.etsii.urjc.es/MonitorPlacement.

ORCID

Alejandra Casado https://orcid.org/0000-0003-3417-6859
Nenad Mladenović https://orcid.org/0000-0001-6655-0409
Jesús Sánchez-Oro https://orcid.org/0000-0003-1702-4941
Abraham Duarte https://orcid.org/0000-0002-4532-3124

REFERENCES

[1] G. Andersson, P. Donalek, R. Farmer, N. Hatziargyriou, I. Kamwa, P. Kundur, N. Martins, J. Paserba, P. Pourbeik, and J. Sanchez-Gasca, *Causes of the 2003 major grid blackouts in north america and europe, and recommended means to improve system dynamic performance*, IEEE Trans. Power Syst. **20** (2005), no. 4, 1922–1928.
[2] T. Back and S. Khuri, *An evolutionary heuristic for the maximum independent set problem*, Proc. 1st IEEE Conf. Evol. Comput. IEEE World Congr. Comput. Intell., IEEE, 1994, pp. 531–535.
[3] S. P. Borgatti and M. G. Everett, *A graph-theoretic perspective on centrality*, Soc. Netw. **28** (2006), no. 4, 466–484.
[4] P. Casas-Martínez, A. Casado-Ceballos, J. Sánchez-Oro, and E. G. Pardo, *Multi-objective grasp for maximizing diversity*, Electronics **10** (2021), no. 11, 1232.
[5] D. P. Chandu, *A parallel genetic algorithm for three dimensional bin packing with heterogeneous bins*. arXiv preprint arXiv:1411.4565, 2014.
[6] P. Crucitti, V. Latora, and M. Marchiori, *Model for cascading failures in complex networks*, Phys. Rev. E **69** (2004), no. 4, 045104.
[7] I. Dinur and S. Safra, *On the hardness of approximating minimum vertex cover*, Ann. Math. **162** (2005), 439–485.
[8] A. Duarte, R. Martí, F. Glover, and F. Gortazar, *Hybrid scatter tabu search for unconstrained global optimization*, Ann. Oper. Res. **183** (2011), no. 1, 95–123.
[9] I. K Evans, Evolutionary algorithms for vertex cover. Proc. Int. Conf. Evol. Program., Springer, 1998, pp. 377–386.
[10] P. Hansen, N. Mladenović, R. Todosijević, and Saı¨ d Hanafi, *Variable neighborhood search: Basics and variants*, EURO J. Comput. Optim. **5** (2017), no. 3, 423–454.
[11] N. Hatano and M. Suzuki, *Quantum annealing and other optimization methods*. arXiv preprint math-ph/0506007, 2005.
[12] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, MIT Press, Massachussetts, 1992.
[13] R. M. Karp, "*Reducibility among combinatorial problems*," *Complexity of computer computations*, Springer, New York, 1972, pp. 85–103.
[14] S. Lagraa and J. François, *Knowledge discovery of port scans from darknet*, Proc. 2017 IFIP/IEEE Symp. Integr. Netw. Serv Manag. (IM). IEEE, 2017, pp. 935–940.
[15] E. L. Lawler and D. E. Wood, *Branch-and-bound methods: A survey*, Oper. Res. **14** (1966), no. 4, 699–719.
[16] R Luling and B. Monien, *Load balancing for distributed branch & bound algorithms*, Proc. 6th Int. Parallel Process. Symp., IEEE, 1992, pp. 543–548.
[17] M. Milanovic, *Solving the generalized vertex cover problem by genetic algorithm*, Comput. Inform. **29** (2010), no. 6, 1251–1265.
[18] R. Mueller-Bady, R. Gad, M. Kappes, and I. Medina-Bulo, *Using genetic algorithms for deadline-constrained monitor selection in dynamic computer networks*, Proc. Comp. Publ. 2015 Annu. Conf. Genet. Evol. Comput, 2015, pp. 867–874.
[19] R. Mueller-Bady, M. Kappes, I. Medina-Bulo, and F. Palomo-Lozano, *Optimization of monitoring in dynamic communication networks using a hybrid evolutionary algorithm*, Proc. Genet. Evol. Comput. Conf., 2017, pp. 1200–1207.
[20] R. Mueller-Bady, M. Kappes, I. Medina-Bulo, and F. Palomo-Lozano, *An evolutionary hybrid search heuristic for monitor placement in communication networks*, J. Heurist. **25** (2019), no. 6, 861–899.
[21] M. E. J. Newman, *A measure of betweenness centrality based on random walks*, Soc. Netw. **27** (2005), no. 1, 39–54.
[22] M. Pelikan and D. E. Goldberg, "*Hierarchical bayesian optimization algorithm*," *Scalable optimization via probabilistic modeling*, Springer, New York, 2006, pp. 63–90.
[23] S. Pérez-Peló, J. Sánchez-Oro, A. Gonzalez-Pardo, and A. Duarte, *A fast variable neighborhood search approach for multi-objective community detection*, Appl. Soft Comput. **112** (2021), 107838.

[24] J. Sánchez-Oro, J. J. Pantrigo, and A. Duarte, *Combining intensification and diversification strategies in vns. an application to the vertex separation problem*, Comput. Oper. Res. **52** (2014), 209–219.
[25] D. Zhang, E. K. Cetinkaya, and J. P. G. Sterbenz, *Robustness of mobile ad hoc networks under centrality-based attacks*, Proc. 2013 5th Int. Congr. Ultra Modern Telecommun. Control Syst. Worksh. (ICUMT)., IEEE, 2013, pp. 229–235.

How to cite this article: A. Casado, N. Mladenović, J. Sánchez-Oro, and A. Duarte, *Variable neighborhood search approach with intensified shake for monitor placement*, Networks.. (2022), 1–15. https://doi.org/10.1002/net.22134

APPENDIX

TABLE A1 Comparison of (LS+PI) EA [20] and BVNS for each instance, considering the median objective function value obtained along 100 executions, as well as the average computing time

	BVNS		(LS+PI) EA	
Instance	**Time (s)**	**Median O.F.**	**Time (s)**	**Median O.F.**
delaunay_n10	0.20	717.50	3.33	755
delaunay_n11	0.42	1422.00	6.79	1514
delaunay_n12	2.16	2863.00	13.46	3046
delaunay_n13	9.63	5700.50	25.11	6120
delaunay_n14	37.75	11 415.50	54.10	12 304
delaunay_n15	174.57	22 821.50	178.13	24 848
delaunay_n16	484.26	45 687.50	472.69	51 058
delaunay_n17	262.56	91 917.00	1106.82	151 210
delaunay_n18	1478.32	189 261.00	2421.55	654 566
frb30-15-1	0.22	435.00	63.30	438
frb30-15-2	0.21	435.00	72.85	437
frb30-15-3	0.25	435.00	67.55	438
frb30-15-4	0.20	435.00	72.84	437
frb30-15-5	0.18	435.00	77.69	436
frb30-17-1	0.55	578.00	230.92	582
frb30-17-2	0.43	578.00	259.31	583
frb30-17-3	0.46	578.00	218.72	582
frb30-17-4	0.47	578.00	236.29	582
frb30-17-5	0.58	578.00	273.25	582
frb40-19-1	0.76	741.00	413.36	747
frb40-19-2	0.99	741.00	414.94	747
frb40-19-3	0.90	741.00	484.33	747
frb40-19-4	1.24	741.00	473.27	745
frb40-19-5	0.93	741.00	429.84	747
internet-as	19.94	5701.00	179.18	7595
nren	26.89	422.00	2.02	452
p2p-Gnutella04	0.02	4352.50	24.69	5017
p2p-Gnutella05	5.58	3431.00	21.46	3958
p2p-Gnutella06	3.36	3407.00	22.88	3916
p2p-Gnutella08	3.44	2057.00	14.10	2344
p2p-Gnutella09	1.02	2574.00	17.19	2984
p2p-Gnutella24	1.52	7210.00	67.03	9382
p2p-Gnutella25	11.76	6017.00	36.37	7794
p2p-Gnutella30	7.15	9272.50	123.85	12 419
p2p-Gnutella31	24.52	15 694.00	320.01	23 886
tech-routers-rf	0.14	796.00	7.05	849
tech-WHOIS	3.82	2285.00	57.10	2468

TABLE A2 Individual results found by algorithm BVNS for each considered instance, considering best, worst, and average objective function value, as well as the average computing time.

Instance	Min. O.F.	Max. O.F.	Avg. O.F.	Avg. time (s)
delaunay_n10	715	720	717.60	0.20
delaunay_n11	1419	1425	1422.30	0.42
delaunay_n12	2853	2868	2861.60	2.16
delaunay_n13	5698	5708	5701.70	9.64
delaunay_n14	11 408	11 426	11 416.40	37.75
delaunay_n15	22 809	22 833	22 821.30	174.58
delaunay_n16	45 662	45 714	45 688.90	484.26
delaunay_n17	91 915	92 023	91 938.80	262.57
delaunay_n18	189 261	189 261	189 261.00	1478.33
frb30-15-1	435	435	435.00	0.23
frb30-15-2	435	435	435.00	0.22
frb30-15-3	435	435	435.00	0.25
frb30-15-4	435	435	435.00	0.20
frb30-15-5	435	435	435.00	0.18
frb35-17-1	578	578	578.00	0.56
frb35-17-2	578	578	578.00	0.43
frb35-17-3	578	578	578.00	0.46
frb35-17-4	578	578	578.00	0.48
frb35-17-5	578	578	578.00	0.59
frb40-19-1	741	741	741.00	0.76
frb40-19-2	741	741	741.00	1.00
frb40-19-3	741	741	741.00	0.91
frb40-19-4	741	741	741.00	1.24
frb40-19-5	741	741	741.00	0.94
internet-as	5701	5704	5701.70	19.95
nren	422	424	422.60	26.89
p2p-Gnutella04	4351	4355	4352.80	0.02
p2p-Gnutella05	3430	3432	3430.90	5.58
p2p-Gnutella06	3406	3410	3407.20	3.36
p2p-Gnutella08	2055	2058	2056.80	3.45
p2p-Gnutella09	2574	2574	2574.00	1.02
p2p-Gnutella24	7209	7212	7210.20	1.52
p2p-Gnutella25	6017	6017	6017.00	11.76
p2p-Gnutella30	9271	9274	9272.30	7.16
p2p-Gnutella31	15 694	15 695	15 694.30	24.52
tech-routers-rf	795	796	795.80	0.14
tech-WHOIS	2285	2286	2285.30	3.83

Capítulo 7

An iterated greedy algorithm for finding the minimum dominating set in graphs

Los detalles del artículo y la revista en la que está publicado son:

- Casado, A., Bermudo, S., López-Sánchez, A. D., & Sánchez-Oro, J. (2023). An iterated greedy algorithm for finding the minimum dominating set in graphs. Mathematics and Computers in Simulation, 207, 41-58. https://doi.org/10.1016/j.matcom.2022.12.018
- Estado: **Publicado**
- Factor de impacto (JCR 2023): 4.600
- Posición relativa: 36/110
- Cuartil: Q1

Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 207 (2023) 41–58
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

An iterated greedy algorithm for finding the minimum dominating set in graphs

A. Casado[a], S. Bermudo[b], A.D. López-Sánchez[b], J. Sánchez-Oro[a,*]

[a] *Computer Science Department. Universidad Rey Juan Carlos, Tulipán s/n. 28933 Móstoles, Madrid, Spain*
[b] *Department of Economics, Quantitative Methods, and Economic History. Universidad Pablo de Olavide, Ctra. de Utrera km 1. ES-41013 Sevilla, Spain*

Received 22 July 2022; received in revised form 29 October 2022; accepted 23 December 2022
Available online 29 December 2022

Abstract

A dominating set in a graph is a set of vertices such that every vertex outside the set is adjacent to a vertex in the set. The domination number is the minimum cardinality of a dominating set in the graph. The problem of finding the minimum dominating set is a combinatorial optimization problem that has been proved to be $\mathcal{NP}$-hard. Given the difficulty of this problem, an Iterated Greedy algorithm is proposed for its solution and it is compared to the solution given by an exact algorithm and by the state-of-art algorithms. Computational results show that the proposal is able to find optimal or near-optimal solutions within a short computational time. Specifically, from the set of instances which can be optimally solved, the proposed method presents an average deviation of 0.04%. Regarding the more complex set of instances, where the exact method is not able to reach the optimal value, the proposed method achieves an average deviation of 1.23% with respect to the best-known solution.

Keywords: Domination number; Minimum dominating set; Greedy heuristics; Iterated greedy; Exact algorithm

1. Introduction

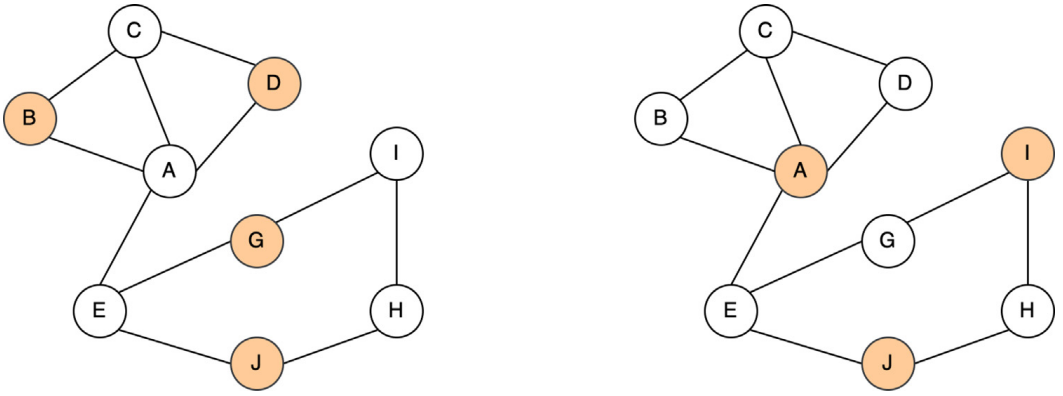
Within the last sixty years and due to the development of large networks such as road networks, social networks, electrical networks, communication networks, computer networks or security networks among others, the graph theory has been the focus of interest for many researchers and practitioners. In graph theory, the study of domination and related subset problems, such as matching, independence or covering has had a significant growth. In particular, the problem of finding dominating sets in graphs started in 1960 (see [7]), but it was in 1962 when the concept of domination number of a graph was defined in [20]. Since then, more than 2000 research papers have been published on this topic. We refer the reader to [13,14,30] for comprehensive surveys.

1.1. Problem definition

A dominating set in a graph is a set of vertices such that every vertex outside this set is adjacent to at least a vertex in the set. The application of domination in graph lies in various fields for solving real life problems. It

* Corresponding author.
E-mail address: jesus.sanchezoro@urjc.es (J. Sánchez-Oro).

https://doi.org/10.1016/j.matcom.2022.12.018



(a) A minimal dominating set D_1. (b) A minimum dominating set D_2.

Fig. 1. Example of a minimal and a minimum dominating sets.

includes social networks (see [29]), radio stations (see [9]), computer communication networks (see [31]), network surveillance (see [12]), sets of representatives, etc. Fig. 1 shows an example with 9 vertices. Notice that the set $D_1 = \{B, D, G, J\}$, depicted in Fig. 1(a), is a minimal dominating set (it does not contain a proper subset which is a dominating set), while the set $D_2 = \{A, I, J\}$, depicted in Fig. 1(b), is a minimum dominating set or a γ-set (a dominating set with the smallest cardinality) for the considered graph.

Many real-world situations fits into domination in graphs. For instance, let us consider a graph whose vertices represent a group of people and edges between two people means that they know each other. The problem of determining a good set of representatives consists on finding the smallest group of people such that everyone not on it has a known person in that group. Therefore, the solution is a dominating set with the smallest cardinality (see [13]).

A similar application could be used to monitor a computer network or to send information through a computer network. In this example, the computer network is modeled by a graph where vertices represent computers and edges are direct communication links between pairs of computers to pass information. The problem considers that from time to time it is needed to collect information from all computers. Therefore, it is needed to identify a small set of computers, usually known as backbone, connected to all other computers to collect all the information (see [13]).

Nowadays, the most popular application of domination in graphs lies in a graph representing a social network, such as Facebook, Twitter or Instagram, since they serve as an important medium for communication and information disseminating. Here, vertices represent users of the social network and edges are friendship between users on the social network. The popularity of social networks makes them a very powerful tool in the viral marketing of products and political campaigns. Many authors proposed different influence maximization problems [6,8,16] where a particular case is the problem of determining the smallest group of users able to influence the rest (thanks to the friendship connections). Therefore, the objective of finding dominating sets is intimately related with the influence maximization problem because a dominating set is a direct way to reach every user in a social network, so it is a cascade model with only one step (see [6,8,16]).

More formally, let $G = (V, E)$ be a simple graph of *order* $n = |V|$ and *size* $m = |E|$, being $|V|$ and $|E|$ the cardinality of the vertex and edge sets, respectively. A set $S \subseteq V$ is a *dominating set* in G if every vertex in $V \setminus S$ is adjacent to a vertex in S. A set $D \subseteq V$ is a *minimal dominating set* in G if it is a dominating set and it does not contain a proper subset which is a dominating set. The minimum cardinality among all dominating sets is called the *domination number*, denoted by $\gamma(G)$, and a dominating set with that cardinality is called a *minimum dominating set* or a *γ-set*. The problem of finding the minimum dominating set in a graph is known as the Minimum Dominating Set Problem (hereinafter MDSP).

There are a finite number of possible dominating sets of G, as maximum $2^n - 1$ sets. To determine the value of $\gamma(G)$ and obtain the minimum dominating set $D^\star$, it is sufficient to enumerate all subsets of V in non-decreasing order of cardinality and check, for any given subset $S \subseteq V$, whether S is a dominating set. Therefore, the value of $\gamma(G)$ is the cardinality of the first dominating set found. Although such enumeration algorithm is easy, unfortunately, it requires $O(2^n)$ steps in the worst case, that is, it has an exponential time complexity that depends on the order

of the graph. To date, as far as we know, the best exact algorithm able to find the minimum dominating set for arbitrary graphs was proposed by Fomin et al. (see [10]) and it has a complexity $O(1.5137^n)$.

1.2. Mathematical model

Next, a mathematical model based on an Integer Linear Programming (ILP) formulation is included in order to formalize the MDSP. Let $A = (a_{ij})$ be the adjacency matrix of the considered graph G. As it is well-known, the adjacency matrix is a square $n \times n$ matrix such that the element a_{ij} is equal to one when there is an edge from vertex i to vertex j, and zero otherwise. Therefore, this model uses binary variables x_i that take value 1 if vertex i is selected and 0 otherwise. The problem can be formulated as follows:

$$\min \sum_{i=1}^{n} x_i \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{n} a_{ij} x_i \geq 1 \quad \forall j = 1, \ldots, n, \tag{2}$$

$$x_i \in \{0, 1\} \quad \forall i = 1, \ldots, n. \tag{3}$$

The objective function (1) represents the total number of selected vertices while constraints (2) ensure that each non-selected vertex in G has a neighbor which is selected. Finally, constraints (3) define the nature of the decision variables.

Since the problem of finding the minimum dominating set is $\mathcal{NP}$-hard for arbitrary graphs (see, for instance, [14]), the use of metaheuristics to find such γ-sets is more than justified. Although there are only a few works focused on finding an algorithm to approximate the domination number (see [4,15,22,23,28]), the problem has been extensively studied for particular graphs, and many bounds for this parameter have been given for general graphs (see, for instance, [13,14]).

1.3. Metaheuristics for the MDSP

Metaheuristics are one of the most extended approximate algorithms for solving hard and complex combinatorial optimization problems and they can be defined as strategies that guide the search process to escape from local optima. These are robust algorithms able to provide high-quality solutions in reasonable computing times without guaranteeing optimality. Its use is justified when it is not possible to compute the optimal solution with an exact algorithm.

Although there exist several taxonomies, the most extended one classifies metaheuristics in two different categories: trajectory-based and population-based metaheuristics. The former maintains a single solution during the search, performing different movements from one solution to another one in the search space in an iterative way, with the aim of attaining a good solution along the trajectory; while the latter maintains a population of solutions in the search, using the information of the entire population to guide the search.

This work is focused on designing a metaheuristic algorithm able to approximate the minimum dominating set and, therefore, the domination number for arbitrary graphs, even for graph mining, that are large-scale real-world graphs. Specifically, an Iterated Greedy (IG) algorithm is proposed to address this problem based on the idea of starting from an initial solution that will be destructed and reconstructed in an iterative way until a stopping criterion is met (more details are included in Section 2).

There are several works in the literature addressing the minimum dominating set from a metaheuristic perspective. The most recent paper [4] proposes an order-based Randomized Local Search (RLS) algorithm, that represents a solution based on permutations of vertices which are transformed into dominating sets by means of a greedy algorithm. Then, if S is a solution and we consider the vertices of S as the first $|S|$ vertices in V, this solution is improved using a simple movement, named jump perturbation operator, that selects a position $j \in \{2, 3, \ldots, n\}$, put v_j in the first position and shift one position to the right all vertices between positions 1 and $j - 1$. The author solves the problem for unit disk graphs, scale-free networks, real-world graphs including samples of two social networks and graphs studied in the field of network science, and he performs a comparison to the state-of-the-art algorithms.

Next, the state-of-the-art algorithms are briefly mentioned. The first one is an Ant Colony Optimization Algorithm hybridized with a Local Search (ACO-LS) proposed by [22] that randomly generates a population of solutions that evolves following a probabilistic scheme known as the pheromone value of ants associated with a vertex. The LS consists on removing redundant vertices of a solution. Then, an extension of the Ant Colony Optimization algorithm hybridized with a Local Search with Pre-Processing (ACO-PP-LS) is proposed by [23], that generates a population of solutions using a greedy algorithm obtaining solutions that are not totally random, thus accelerating the convergence of ACO-LS algorithm. Furthermore, a new variant of ACO-LS, named ACO-LS-S, is presented [4], which selects the vertices involved in each transition and initialize the pheromones following a greedy criterion.

The main contributions of this paper are detailed next:

- We propose an algorithm able to solve the minimum dominating set problem for any graph without a specific structure.
- The algorithm leverages the topology of the graph identifying support and leaf vertices with the aim of reducing the solution space, thus accelerating the algorithm.
- Two constructive procedures are proposed: the first one iteratively generates a solution from scratch, while the second one follows the opposite approach, i.e., it starts from a complete graph and iteratively removes the least promising vertices.
- A post-processing method is presented to remove redundant vertices from the explored solutions, which accelerates the algorithm.
- An efficient local search is presented, which is able to perform only the most promising moves by predicting if a move will end in a feasible solution.
- We have significantly enlarged the testbed of instances in order to make more robust comparisons.

This work is organized as follows. Section 2 describes the algorithm implemented to solve the problem under consideration. Section 3 presents the computational results performed to test the quality of the proposal, including a comparison against exact algorithms. Finally, Section 4 summarizes the paper and discusses future works.

2. Algorithmic approach

This Section is devoted to provide a thorough description of the proposed algorithm designed to address the MDSP. Specifically, a metaheuristic approach based on IG framework with the aim of providing high-quality solutions in short computing times. We have opted by a metaheuristic since the MDSP is $\mathcal{NP}$-hard and an exact procedure is not able to deal with complex instances. Therefore, the metaheuristic approach emerges as the most adequate method for solving large and complex instances in the context of MDSP.

IG is a metaheuristic framework originally proposed for solving a scheduling problem [25] and it has been in continuous evolution, proving its efficiency for solving a wide variety of hard combinatorial optimization problems. For instance, IG has been considered for solving community detection problems [27], robotic flowshop scheduling problems [18], or regenerator location problems [24], among others. The success of IG relies in the simple yet effective idea of iteratively destructing and then reconstructing the incumbent solution to escape from local optima. Algorithm 1 depicts the pseudocode of IG framework.

The algorithm receives three input parameters, namely: $G = (V, E)$, the input graph; β, the percentage of vertices removed in the destruction phase; and Δ, the maximum number of iterations without improvement that IG allows to perform (stopping criterion).

IG starts by creating an initial solution using one of the constructive procedures proposed in Section 2.1 (step 1). Note that this initial solution is a minimal dominating set. Then, a local optimum is found, starting from this initial solution, by applying the local improvement method presented in Section 2.2 (step 2), becoming D_b the best solution found so far, in our case, resulting in an eventually better minimal dominating set. The method then iterates until reaching a fixed stopping criterion (steps 4–14). In the context of MDSP, the stopping criterion considered is a maximum number of iterations performed without finding any improvement, which is controlled by the input parameter Δ. In each iteration, the incumbent solution D_b is partially destroyed with one of the destruction procedures described in Section 2.3, generating a solution D_d (step 5). Notice that D_d is an unfeasible solution since the partial destruction performed results in a set that is not a minimal dominating set anymore and not even a dominating set, as it will be later discussed in Section 2.3. Then, in order to attain a new minimal dominating set, the feasibility is fixed by means of one of the reconstruction procedures presented in Section 2.4, resulting in

Algorithm 1: IG($G = (V, E)$, β, Δ)

1: $D \leftarrow InitialSolution(G)$
2: $D_b \leftarrow LocalImprovement(D)$
3: $\delta \leftarrow 0$
4: **while** $\delta < \Delta$ **do**
5: $\quad D_d \leftarrow Destruction(D_b, \beta)$
6: $\quad D_r \leftarrow Reconstruction(D_d)$
7: $\quad D_i \leftarrow LocalImprovement(D_r)$
8: $\quad$ **if** $|D_i| < |D_b|$ **then**
9: $\quad\quad D_b \leftarrow D_i$
10: $\quad\quad \delta \leftarrow 0$
11: $\quad$ **else**
12: $\quad\quad \delta = \delta + 1$
13: $\quad$ **end if**
14: **end while**
15: **return** D_b

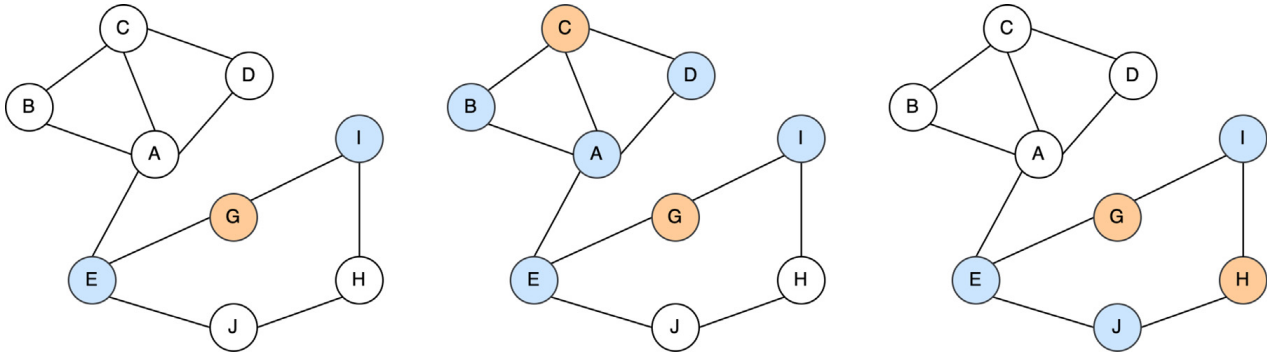
a new feasible solution D_r (step 6). The reconstructed solution D_r is not necessarily a local optimum, so the local improvement method is applied to further improve it, resulting the solution set D_i, which is a minimal dominating set again (step 7). At the end of each iteration, the objective value of the set D_i is compared with the objective value of the best set found so far D_b (step 8). Notice that, as it was aforementioned in Section 1, the objective function value is evaluated as the cardinality of every set. If D_i outperforms D_b, then D_b is updated (step 9), and the number of iterations without improvement is reset (step 10). Otherwise, the number of iterations without improvement is incremented for the next iteration (step 12). The method ends when reaching the maximum number of iterations without improvement, i.e., $\delta = \Delta$, returning as solution the best set found during the search (step 15).

2.1. Initial solution

IG requires from an initial solution to start the search from. Although the literature suggests that even a random solution is enough for IG to provide high-quality solutions, several recent research have shown that starting from a promising region of the search space usually lead to most of metaheuristic algorithms to either reduce the computing time required or to find better solutions [2,21]. Therefore, two different greedy procedures are proposed to generate an initial feasible solution for IG which, in our case, is a minimal dominating set. The first procedure is based on the greedy insertion of vertices at each iteration, meanwhile the second one is based on greedy deletion of vertices at each iteration.

It is important to remark that, leveraging the information derived from the MDSP, there are some specific vertices that can be always taken in optimal solutions for the MDSP, the so-called support vertices. Before defining a support vertex, it is necessary to introduce some preliminary concepts. Let us define $N(v)$ as the set of adjacent vertices to v, and $N[v] = N(v) \cup \{v\}$. For any $D \subseteq V$, we denote $N(D) = \bigcup_{v \in D} N(v)$ and $N[D] = N(D) \cup D$, and we say that a vertex v is dominated by D if $v \in N[D]$. A *leaf vertex* is a vertex v such that $|N(v)| = 1$, and a *support vertex* is a vertex adjacent to a leaf. We denote by L the set of *leaf vertices* of the graph and by SV the set of support vertices.

Therefore, the first procedure follows a traditional greedy approach which starts from an empty solution and it iteratively selects the most promising vertex to be included in the next iteration until reaching a feasible solution. However, instead of starting from an empty solution, it starts from a solution in which all support vertices are already included $D = SV$, and it iteratively adds new vertices until the incumbent solution becomes a dominating set. Analogously, the leaf vertices are never considered to be part of the solution, since they are always dominated by a support vertex. This idea allows the procedure to start from a partial solution, reducing its computational effort. The experiments will show the effect of considering support vertices in the performance of the algorithm. We will refer this methodology as the *Greedy Insertion Procedure* (GIP).



(a) $D = \{\texttt{G}\}$. (b) Evaluation of vertex C. (c) Evaluation of vertex H.

Fig. 2. Evolution of GIP using g_{GIP} to evaluate vertices C and H. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

As it is customary in greedy algorithms, the selection of the next vertex strictly depends on a greedy function which is usually designed specifically for the problem under consideration. In the context of MDSP, the greedy function $g_{GIP}(v)$ selected tries to evaluate the contribution of a vertex v to the solution under construction D. In particular, it is evaluated as the number of adjacent vertices of v which are not dominated by D (since those vertices will be dominated by v if it is finally selected). In mathematical terms,

$$g_{GIP}(v) = |N[v] \setminus N[D]|$$

Fig. 2(a) shows the performance of GIP when starting from $D = \{\texttt{G}\}$, where the selected vertices are colored in orange and the dominated vertices are colored in blue. On the one hand, Fig. 2(b) shows the evaluation of the greedy function when considering vertex C. In particular, selecting vertex C would result in covering vertices B, A, D, and C, which were not previously dominated, resulting in $g_{GIP}(\texttt{C}) = 4$. On the other hand, Fig. 2(c) illustrates the evaluation of greedy function over vertex H. In this case, the new dominated vertices are J and H, since I was already dominated by G, resulting in $g_{GIP}(\texttt{H}) = 2$. In this example, C is selected since $g_{GIP}(\texttt{C}) > g_{GIP}(\texttt{H})$.

Having defined the greedy function, the GIP iteratively selects the vertex with the maximum greedy function value until reaching a feasible solution, i.e., a solution in which all vertices are dominated. Therefore, the process ends when $g_{GIP}(v) = 0, \forall v \in V \setminus D$.

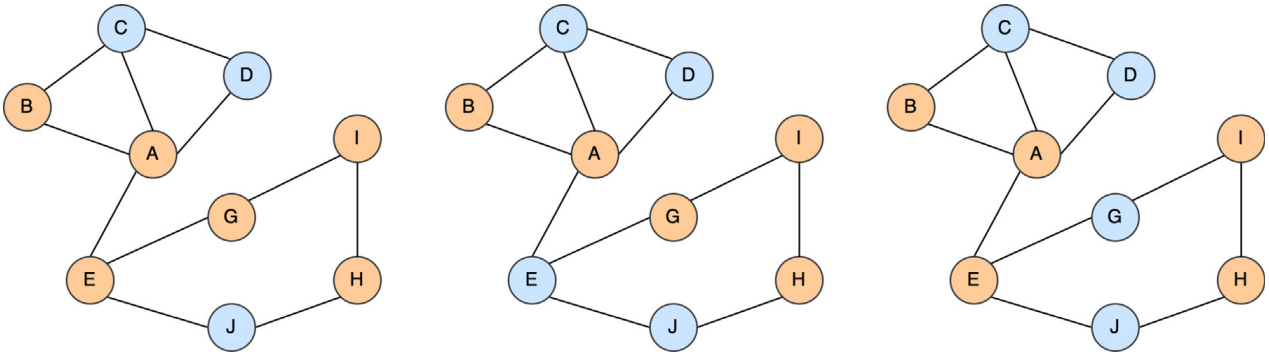
The second procedure proposed follows an opposite approach. Instead of starting from scratch (but for the support vertices), and iteratively selecting a new vertex to be included in the solution, this procedure considers that the solution initially contains all vertices of the graph. Then, the method iteratively removes a vertex from the solution until reaching a state in which the removal of any vertex results in an unfeasible solution. Similarly to the GIP, the support vertices are not considered to be removed, and the leaf vertices are not initially included in the solution. Therefore, the method actually starts with $D = G \setminus L$, where L is the set of leaf vertices. Then, the *Greedy Deletion Procedure* (GDP) iteratively removes the least promising vertex to be kept in the solution by means of a different greedy function value.

In this case, it is interesting to remove the vertex v such that the vertices in its closed neighborhood $N[v]$ remain dominated by as many vertices as possible. The rationale behind this is that this criterion will eventually allow more vertices to be removed in future iterations. More formally,

$$g_{GDP}(v) = \min_{u \in N[v]} |(D \setminus \{v\}) \cap N[u]|$$

Following this greedy function, the GDP iteratively selects the vertex v with the largest value of $g_{GDP}(v)$, until no vertex can be removed without guaranteeing the feasibility of the solution.

Fig. 3(a) shows the behavior of GDP when starting from solution $D = \{\texttt{A}, \texttt{B}, \texttt{E}, \texttt{G}, \texttt{H}, \texttt{I}\}$, following the same coloring as in Fig. 2. In Fig. 3(a) the greedy function g_{GDP} is evaluated over vertex E. The adjacent vertices to E are A, G, and J. If E is removed, the vertex A is still dominated by vertices A and B; the vertex G is dominated by G and I; the vertex E is dominated by G and A; and the vertex J is dominated by H. Therefore, the value of $g_{GDP}(\texttt{E}) = 1$, since it is the minimum number of vertices that cover a vertex of $N[\texttt{E}]$. Fig. 3(c) shows a similar



(a) $D = \{\texttt{A},\texttt{B},\texttt{E},\texttt{G},\texttt{H},\texttt{I}\}$. (b) Evaluation of vertex E. (c) Evaluation of vertex G.

Fig. 3. Evolution of GDP using g_{GDP} to evaluate E and G.

evaluation over vertex G, whose adjacent vertices are E and I. In the case of E, it remains dominated by A and E, vertex G is dominated by E and I, while vertex I is dominated by H and I. Then, the greedy function value is $g_{GDP}(\texttt{G}) = 2$. Therefore, the vertex G will be selected to be removed by this procedure before selecting vertex E.

Up to this point, a solution generated either with GIP or with GDP is always a dominating set. However, in the case of GIP, it is not guaranteed that it is a minimal dominating set. In order to assure that any solution derived from GIP or GDP is a minimal dominating set, we propose a procedure that checks if some of the vertices belonging to the dominating set are redundant. To be more accurate, the procedure checks if a vertex $v \in D$ and all their adjacent vertices are dominated by $D \setminus \{v\}$ and, if so, v will be removed from the dominated set. With the aim of checking and removing those redundant vertices, the *checking procedure* (CheckP) traverses all vertices in the dominating set D and removes all redundant vertices. A vertex $v \in D$ is redundant if and only if $|(D \setminus \{v\}) \cap N[u]| \geq 1, \forall u \in N[v]$. In this case, v can be definitively eliminated from D as long as the above condition is met.

2.2. Local improvement

In IG framework, both the initial solution and the one resulting in each iteration after the reconstruction phase are not necessarily local optima. Therefore, a local improvement method is proposed to further improve those solutions which may eventually lead to better results.

The improvement phase can be performed by considering complex methods such as Tabu Search [2], Variable Neighborhood Search [19] or any other metaheuristic. However, these strategies usually result in rather computationally demanding algorithms. In order to maintain an efficient algorithm, a simple yet effective local search procedure is proposed.

The definition of a local search is given by its three main components: the move operator considered, the neighborhood explored, and the strategy followed to explore the neighborhood. In the context of MDSP, the move operator is defined as an exchange move. This movement consists of removing a vertex from the solution under evaluation, replacing it with a new vertex which is not already included in the solution. More formally,

$$Exchange(D, u, v) \leftarrow (D \setminus \{u\}) \cup \{v\}$$

This move operator is only considered if the resulting solution is feasible. If the removal of vertex u and the insertion of vertex v leaves one or more vertices without being dominated, then the move is undone to maintain the original feasible solution.

Notice that, since the evaluation of the objective function is the cardinality of the solution, the exchange move will never lead to an improvement by itself. However, after performing an exchange move, some of the previously selected vertices may be redundant, therefore the checking procedure is again applied to remove them and improve the value of the objective function.

Hence, an improvement move in the context of MDSP is defined as an exchange move whose corresponding checking procedure is able to remove, at least, one vertex.

Having defined the move operator, the neighborhood of a solution can be directly derived from it. In particular, the local improvement phase considers the neighborhood $N_e(D)$, which is conformed with all feasible solutions that can be reached by performing an exchange move. In mathematical terms,

$$N_e(D) \leftarrow \{D' = Exchange(D, u, v), \forall u \in D \wedge \forall v \in V \setminus D\}$$

Once the neighborhood has been defined, it is necessary to establish the strategy used to traverse the neighborhood. There are two traditional strategies that can be found in the literature: best improvement and first improvement. The former traverses the neighborhood performing, in each step, the move that leads to the best solution in the neighborhood. The latter, on the contrary, performs the first move that leads to an improving solution in the neighborhood. Both of them stop when no improvement is found.

In this research first improvement is considered since it has been experimentally shown that it is able to reach similar results than best improvement but being less computationally demanding [3,11]. In the context of first improvement, the order in which the neighborhood is traversed may affect the results. In order to diversify the search, we consider a random order for traversing the neighborhood.

Analyzing the described Local Search (LS), it can be seen that several moves considered in the search lead to an unfeasible solution. Therefore, it would be desirable to perform only those moves that result in a feasible solution. To that end, an efficient version of the local search procedure is proposed. This Efficient Local Search (ELS) is able to identify which moves lead to a feasible solution before actually performing the move.

This optimization of the local search procedure consists of, given the vertex to be removed from the solution, identify which are the candidate vertices that can replace the one that will be removed. Let us consider that $u \in D$ and $v \notin D$. To remove vertex u from the solution and replace it by vertex v, the necessary condition is that every vertex dominated only by u, i.e., every vertex $w \in N[u]$ such that $|D \cap N[w]| = 1$, is dominated by v, i.e., $w \in N[v]$.

Given this definition, the efficient local search only considers those moves that lead to feasible solutions, thus reducing the number of operations performed without deteriorating the solution quality. The experimental section will show the effect of this optimization in the final algorithm.

2.3. Destruction phase

The destruction phase of IG algorithm is one of the responsible stages for diversifying the search by destroying a certain part of the solution. Destroying (partially) a solution in the context of MDSP basically consists of removing some of the selected vertices. The number of vertices to be removed depends on the input parameter β. With the aim of having a scalable algorithm, the value of β is a percentage of the size of the solution (or $\beta \in [0, 1]$), and it will be discussed in Section 3.

The destruction phase can be performed following either a random or greedy criterion. The former focuses on diversification and leads the algorithm to different regions of the search space while the latter, focused on intensification, tries to find a better solution in the region under exploration. In this research, both alternatives are presented to evaluate the effect of each one of them in the final algorithm.

The first variant, the *random destruction*, removes β of the vertices from the incumbent solution (or $\beta \cdot |D|$ vertices from the incumbent solution). Those vertices are selected at random, resulting in an unfeasible solution, which will become feasible after applying the reconstruction phase presented in Section 2.4. This random selection allows the algorithm to consider solutions that, otherwise, would be never explored.

The second variant, named *greedy destruction*, also removes β of the vertices from the solution (or $\beta \cdot |D|$ vertices from the solution), but those vertices are selected following a greedy criterion. The rationale behind this is that selecting the vertices whose contribution to the solution is minimal will eventually reduce the size of the dominating set. The greedy criterion followed in this phase is similar the one used in the GDP presented in Section 2.1. In particular, the method iteratively removes the vertex v whose adjacent vertices (which are those dominated by v) are dominated by the largest number of vertices belonging to $D \setminus \{v\}$.

2.4. Reconstruction phase

The reconstruction phase is designed to recover a feasible solution after having destructed the incumbent one in the destruction phase described in Section 2.3. In order to do so, two different approaches can be followed.

On the one hand, if the reconstruction phase is devoted to diversify, it can be performed by randomly adding vertices to the incumbent solution until all vertices are dominated, i.e., a feasible solution is reached. On the other hand, a greedy criterion can be followed to minimize the number of required vertices to be included in the solution, thus reducing the objective function value.

Therefore, it is necessary to establish a greedy criterion to select the next vertex to be included in the solution being reconstructed. Analogously to the destruction phase, the greedy criterion selected is the same as the one considered in the GIP presented in Section 2.1. Specifically, the vertex selected is the one with the largest number of adjacent vertices which are not yet dominated. This criterion maximizes, in each step, the number of new vertices dominated, thus reducing the number of required dominating vertices.

It is expected that the *random reconstruction* results in more diverse solutions but with a larger number of dominating vertices, while the *greedy reconstruction* generates less diverse solutions but with a reduced number of dominating vertices. The best reconstruction method is tested in Section 3.

2.5. Structural complexity

This Section is devoted to describe the structural complexity of each part of the proposed algorithm and in turn, the complexity of the complete algorithmic proposal. Although the *Big O* notation is used, it is worth mentioning that a good design and implementation of the algorithm usually leads to reduced computing times, as it can be seen in Section 3.

Without loss of generality, let us define p as the cardinality of a given solution. Since the support and leaf vertices are not considered in any part of the procedure, we denote by $\hat{n}$ the number of vertices in the graph which are not support or leaf vertices, and by $\hat{p}$ the number of selected vertices in the solution which are not support vertices. Notice that this value is considerably smaller than n (see Table 2 for an example on the solution size reduction using support vertices).

First of all, the constructive procedures will be analyzed. In this case, both constructive procedures present the same structural complexity. Then, the GIP iteratively adds a new vertex until the solution becomes feasible, with a complexity of $O(\hat{p})$. In each iteration, the next best vertex is selected, with a complexity of $O((\hat{n}-\hat{p})\cdot\hat{n})$, since all non-selected vertices $\hat{n}-\hat{p}$ are traversed and then their adjacent vertices, which could eventually be $\hat{n}$. Then, the complete constructive procedure has a complexity of $O(\hat{p}\cdot(\hat{n}-\hat{p})\cdot\hat{n})$. The same analysis is performed for the GDP.

The next method to be analyzed is the `CheckP`, which traverses the set of selected vertices in the solution, $O(\hat{p})$ and, for each one of them, the adjacent vertices are checked $O(\hat{n})$, resulting in a complexity of $O(\hat{p}\cdot\hat{n})$.

The local search method is the most complex procedure in any metaheuristic algorithm since it requires to traverse a complete neighborhood in several occasions. In this section, only the efficient local search complexity is analyzed since it is the main contribution of this research in the context of local search methods. Each iteration of the efficient local search traverses the set of selected vertices, $O(\hat{p})$, and, for each vertex its neighbors are analyzed, $O(\hat{n})$. Additionally, if the move can be performed (which occurs in a reduced number of iterations), the checking procedure is applied, $O(\hat{p}\cdot\hat{n})$. Then, the complexity of an iteration of the local search procedure, if an improvement is found, is $O(\hat{p}\cdot\hat{p}\cdot\hat{n})$, while an iteration where no improvement is found results in a complexity of $O(\hat{p}\cdot\hat{n})$.

The destruction phase of IG requires to remove a certain number of selected vertices, which is $\beta\cdot\hat{p}$, resulting in a complexity of $O(\beta\cdot\hat{p})$. Notice that $\beta\cdot\hat{p}\ll\hat{p}$. Conversely, the reconstruction phase iteratively adds a new vertex to the incumbent solution until it becomes feasible, requiring, in the worst case, $\hat{n}-\hat{p}$ iterations. Notice that this case indicates that all vertices have been included, which is an extremely rare situation, as it can be seen in the results. Therefore, the complexity of the reconstruction phase is $O(\hat{n}-\hat{p})$.

Summarizing, a single iteration of our IG algorithm performs a destruction phase, $O(\beta\cdot\hat{p})$; a reconstruction phase, $O(\hat{n}-\hat{p})$; a checking, $O(\hat{p}\cdot\hat{n})$; and a local search method, $O(\hat{p}\cdot\hat{p}\cdot\hat{n})$. Then, the complexity of an IG iteration is evaluated as the maximum complexity of the inner methods, which is the local search complexity, $O(\hat{p}\cdot\hat{p}\cdot\hat{n})$. Notice that IG is executed a maximum number of iterations without improvement, which cannot be analyzed from a complexity point of view.

Table 1
Comparison of proposed constructive procedures.

Algorithm	Avg.	Time	Dev. (%)	#Best
GIP	**751.67**	**0.13**	**0.01**	**8**
GDP	1070.00	5.76	0.22	2
RND	2623.33	0.37	1.90	0

3. Computational results

This Section presents and discusses the results of the computational testing conducted with the algorithm proposed in this paper, IG algorithm. A total of 76 instances are solved on an AMD EPYC 7282 (2.8 GHz) with 16 GB RAM, and the heuristic algorithms were implemented using Java 11, while the exact approach has been implemented using Gurobi 9. The set of instances is divided into two different subsets: 28 instances are directly derived from the literature to perform a fair comparison, while new and more challenging 96 instances are generated in order to evaluate the limits of both the exact and metaheuristic approaches. The best method found in the literature is the order-based Randomized Local Search (RLS) algorithm [4], which considers the set of 28 instances aforementioned, divided in: unit disk graphs, random scale-free networks generated by Barabási–Albert model [1], and a set of real-world graphs, including samples from two social network services, graphs studied in network science, as well as several DIMACS graphs. All instances and source code are publicly available at https://grafo.etsii.urjc.es/MDSP.

The experiments are divided into two different phases: preliminary and final experiments. The former is devoted to tune the input parameters of the algorithm and the best strategy to conduct the search, while the latter is designed to perform a direct comparison with the state-of-the-art algorithm, with the aim of evaluating the contribution of our proposal. In order to avoid overfitting, the preliminary experimentation is performed over a subset of 9 representative instances derived from the original set of 28 instances.

All tables report the following metrics: Avg., the average objective function value obtained by each algorithm over the set of considered instances; Time, the average computing time required by the algorithm (in seconds); Dev., the average percentage deviation with respect to the best solution found in the experiments; and #Best, the number of times that each algorithm reaches the best solution of the experiment. Additionally, the best result of each experiment is highlighted in bold font.

3.1. Preliminary experimentation

The first preliminary experiment is devoted to select the best procedure to get the initial solution, in our problem, a minimal dominating set as already explained in Section 2.1. To that end, two methods are considered: the GIP and the GDP. Additionally, the comparison includes a third procedure, denoted as RND, that generates a feasible solution by selecting a vertex at random in each iteration. This random method is included just for illustrative purposes, to show the relevance of designing an specific algorithm for the MDSP. Table 1 shows the results obtained by each constructive procedure.

As it can be derived from Table 1, both the GIP and the GDP obtain consistently better results than the RND. It is worth mentioning that the results obtained by the random procedure also consider that the support vertices are in the solution and the leaf vertices are not, thus starting from a partially constructed solution. Comparing the GIP and the GDP, the differences in computing time are remarkable (0.13 versus 5.76 s on average). This is partially explained because, in the MDSP, the number of vertices required to have a feasible solution is small with respect to the order of the graph, so, generally it will require from more iterations. Additionally, the constructive procedure is able to reach 8 out of 9 best solutions with an average deviation of 0.01%, indicating that in the solution in which it is not able to reach the best solution, it still remains really close to it. This behavior, together with the reduced computing time required, lead us to select the constructive procedure as the best method to generate an initial solution.

The aim of the following experiment is to evaluate the effect of the efficient local search and the naïve local search. Specifically, this comparison is done with respect to the computational effort required to find a local optimum when starting from the same initial solution (the one generated by the GIP). As expected, both methods will obtain

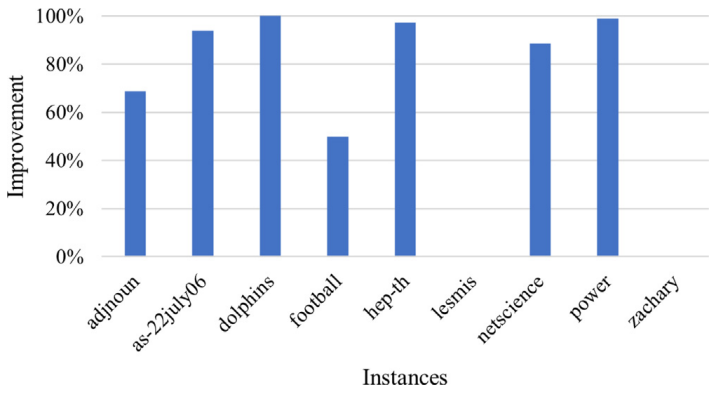


Fig. 4. Percentage of improvement in time obtained when using the efficient local search method.

Table 2
Percentage of support vertices in each instance and percentage of dominated vertices if support vertices are included in the solution.

Instance	$\frac{\lvert SV\rvert}{\lvert V\rvert}\%$	$\frac{N[SV]}{\lvert V\rvert}\%$
`adjnoun`	50.00	80.36
`as-22july06`	91.56	98.95
`dolphins`	46.67	69.35
`football`	0.00	0.00
`hep-th`	74.57	72.11
`lesmis`	70.00	80.52
`netscience`	65.41	51.23
`power`	62.11	68.97
`zachary`	25.00	50.00
Average	53.92	63.50

the same results in terms of quality, however, it would be interesting to compare the computing time required to find a local optimum. To graphically visualize this performance, Fig. 4 shows, for each instance, the percentage of improvement in computing time obtained by ELS when comparing it with the naïve LS method.

Results show the relevance of considering the efficient local search, being, on average, 66% faster than the naïve local search. Moreover, except from instances `lesmis` and `zachary`, in which no improvement in terms of computing time is found, the improvement in computing time is 85% on average. It is worth mentioning that those two instances are the smallest ones, being the most easily solved by the local search method. Therefore, the contribution of the efficient local search is confirmed.

Once that the procedure to generate the initial solution and the efficient local search improvement have been configured, the next experiment tries to analyze the effect of including the support vertices in the solution instead of start from scratch and avoiding the leaf vertices being part of such solution. In order to do so, an analysis of the preliminary set of instances is shown in Table 2 to evaluate the percentage of vertices that are support vertices (column 2) and, additionally, the percentage of vertices that are dominated if support vertices are included in the solution (column 3).

Notice that, on average, almost 54% of the vertices belongs to the set of support vertices and, selecting those vertices result in 63.5% of the vertices dominated. This result shows the importance of considering support vertices. Analyzing every instance individually, in some cases, such as `as-22july06`, these percentages are high and the inclusion of support vertices in the solution almost completely solve the MDSP. On the contrary, there are some instances, such as `football`, in which there are not support vertices. Therefore, it is intelligent to maintain this idea of considering the support vertices to be part of the solution.

In IG framework, it is necessary to tune three parameters: the destruction and reconstruction methodology that will be considered, the percentage of destruction, and the stopping criterion. In this research, we follow a sequential experimental design, since it considerably reduces the number of experiments when comparing it with a full-factorial

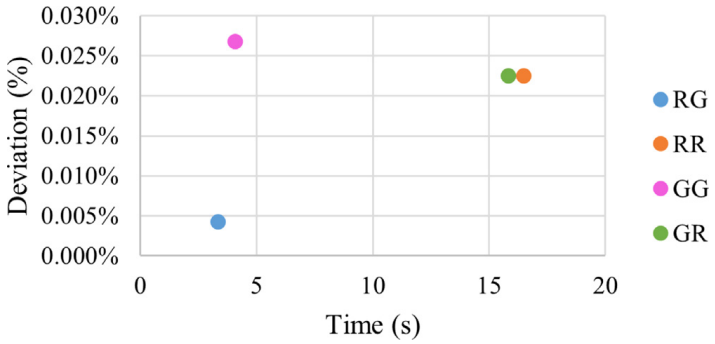


Fig. 5. Comparison of time and average deviation when considering the different combinations of destruction and reconstruction.

Table 3
Effect of different percentage of destruction in the Iterated Greedy algorithm.

β	Avg.	Time	Dev. (%)	#Best
0.1	739.78	**3.34**	0.93	8
0.2	**739.67**	4.85	**0.00**	**9**
0.3	739.78	7.60	0.93	8
0.4	**739.67**	12.10	**0.00**	**9**
0.5	739.78	15.26	0.93	8

design, obtaining similar results [26]. First of all, it is necessary to select the method used in the destruction and reconstruction phase: random or greedy destruction and random or greedy reconstruction. All combinations are tested fixing $\beta = 0.1$ and $\Delta = 250$. The rationale behind this is that we have followed an iterative experimental design, which usually leads to equivalent results as a full-factorial design [26]. Fig. 5 compares the average deviation with respect to the best solution found in the experiments, in percentage, with the computing time spend, in seconds.

In this comparison, it is desirable to be as close as possible to the origin of coordinates, since it would indicate that the method requires from small computing times and has a small deviation with respect to the best solution found in the experiments. Following this analysis, the best method is the *random destruction* and the *greedy reconstruction* (RG), since it is able to produce the best results in the shortest computing times. It is worth mentioning that considering the *random reconstruction* no matter if the destruction is random or greedy (GR and RR) will require from considerably larger computing times, producing very similar results between them. The reason behind this statement is that, when vertices are randomly added, the local search will spend more computational time improving the solutions after the destruction and reconstruction. Then, in the final algorithm, the destruction is performed following the random approach while the reconstruction is performed in a greedy way, balancing diversification and intensification.

The next experiment tries to evaluate the best value for the β parameter, which indicates the percentage of vertices destroyed from a minimal dominating set in each iteration. In particular, the values $\beta = 0.1, 0.2, 0.3, 0.4, 0.5$ are tested, while values larger than 0.5 are not considered since it would destroy more than half of the solution, being equivalent to have a multi-start approach. Table 3 shows the obtained results in this experiment.

As it can be observed, results are quite similar among them in terms of objective function value, although $\beta = 0.2$ and $\beta = 0.4$ seem to be the most promising values regarding the quality. Obviously, the computing time considerably increases with increasing β values. Therefore, comparing these two values, it seems reasonable to set the β value to 0.2, since it is the fastest one.

Finally, it is necessary to establish the value of Δ, which is the number of iterations without improvement allowed in IG metaheuristic and it is used as the stopping criterion. Table 4 shows the results obtained when considering $\Delta = \{50, 100, 150, 200, 250\}$.

In view of the results, it can be stated that the larger the value of Δ, the better the results and, obviously, the larger the computing time required. However, the algorithm converges when reaching $\Delta = 200$ and, therefore, this is the value fixed in the final configuration.

Table 4
Comparison of different values of Δ in the Iterated Greedy algorithm.

Δ	Avg.	Time	Dev. (%)	#Best
50	739.78	**1.96**	0.93	8
100	739.78	2.78	0.93	8
150	739.78	3.85	0.93	8
200	**739.67**	4.27	**0.00**	**9**
250	**739.67**	4.84	**0.00**	**9**

Table 5
Contribution of each part of the algorithm to the final design.

Algorithm	Avg.	Time	Dev. (%)	#Best
GIP	751.67	**0.13**	4.90	3
GIP + ELS	740.67	0.19	3.61	5
IG	**739.67**	4.28	**0.00**	**9**

The final preliminary experiment is not designed for configuring any parameter of the algorithm but for evaluating the contribution of each part of the proposed algorithm to the final version. Therefore, first results of the GIP to generate solutions are compared with the GIP coupled with the efficient local search (ELS) and, finally, with the complete IG algorithm. To summarize, IG algorithm generates an initial solution using the GIP, then a local search method is applied. Next, 20% of vertices are randomly removed in the destruction phase, and they are greedily added in the reconstruction phase until 200 iterations without improvements are reached, returning then the minimal dominating set found. Table 5 compares these three approaches.

As it can be derived from Table 5, IG algorithm is the most relevant part of the final design, and it does not considerably increase the computing time, requiring less than 5 s on average. It is important to remark that the local search is able to reduce the average percentage deviation and find a larger number of best solutions than the GIP marginally increasing the computing time. Finally, it is worth mentioning that the GIP isolated is able to produce high-quality solutions, being, on average, a 4.90% of the best solution found. This experiment proves the relevance of the complete IG framework.

3.2. Final experimentation

Having configured the complete algorithm, this Section is devoted to perform a competitive testing against the state-of-the-art methods, with the aim of evaluating the contribution of our proposal. To that end, two experiments are performed. The first one considers the same set of 28 instances used in [4]. The algorithms compared in this experiment are: the proposed Iterated Greedy, IG; an Order-based Randomized Local Search which is the state of the art, RLS; the implementation of the mathematical model using the mathematical programming solver Gurobi, ILP; a basic Ant Colony Optimization algorithm coupled with a local search, ACO-LS, by [22]; a variant of ACO-LS with a preprocessing phase, ACO-PP-LS, by [23]; and, finally, a new variant of ACO-LS which select vertices involved in each transition and initialize the pheromones according to the greedy results, ACO-LS-S.

It is worth mentioning that, with the aim of having a fair comparison, we have used exactly the same parameters for each algorithm as established in the original work. We refer the reader to [4] for a detailed explanation of each parameter and its value. Table 6 shows the obtained results. In all the results, the total computing time is considered, including the generation of the initial solution.

If we focus on RLS, ACO-LS, ACO-PP-LS and ACO-LS-S, the four algorithms proposed in the state-of-the-art, we can secure that ACO-LS and ACO-PP-LS are those that obtained worst results in terms of quality. Between ACO-LS-S and RLS, RLS reaches a smaller deviation and more number of best solutions, confirming that RLS is better than the other algorithms included in the comparison. Comparing IG and RLS, IG clearly outperforms RLS not only considering the objective function value but also the computational time, being almost 4 times faster on average. Results obtained suggest that the considered instances are not really challenging given that the mathematical programming solver (ILP) is able to solve all of them in less than a second, on average. This is mainly due to the inherent structure of the instances, which probably present a flat landscape in which several solutions reach the

Table 6
Comparison of the Iterated Greedy (IG), the Order-based Randomized Local search method (RLS), a mathematical programming solver (ILP), and three variants of Ant Colony Optimization (ACO-LS, ACO-PP-LS, and ACO-LS-S), when considering the same set of instances presented in [4].

Algorithm	Avg.	Time	Dev. (%)	#Best
IG	1304.68	164.38	0.04	20
ILP	**1303.11**	**0.25**	**0.00**	**28**
RLS	1309.35	600.00	0.16	20
ACO-LS	1550.66	600.00	11.82	13
ACO-PP-LS	1545.96	600.00	11.51	14
ACO-LS-S	1341.33	600.00	1.33	15

Table 7
Comparison of the Iterated Greedy (IG), the Order-based Randomized Local search method (RLS), a mathematical programming solver (ILP), and three variants of Ant Colony Optimization (ACO-LS, ACO-PP-LS, and ACO-LS-S), considering the newly generated set of 96 instances.

Algorithm	Avg.	Time	Dev. (%)	#Best
IG	**10.75**	**23.88**	**1.23**	**88**
ILP	11.94	1799.77	10.18	43
RLS	12.53	600.00	15.07	14
ACO-LS	14.12	600.00	33.19	0
ACO-PP-LS	14.11	600.00	33.17	0
ACO-LS-S	14.12	600.00	33.18	0

optimal value. This means that the minimal dominating set is not unique, being able to get more than one set with the same objective function value. In such situations, ILP is able to effectively stop the search having identified the optimal value, while the heuristic approaches are not able to identify that situations. Notwithstanding, the proposed IG is able to reach a slightly smaller deviation by requiring considerably smaller computing times.

With the aim of evaluating the three approaches in a set of more challenging instances, we generate a new testbed of 96 instances using the well-known Knuth's Algorithm S (see [17]) to generate a random set of instances which are more diverse and lack of structure. Table 7 shows the results obtained by the six approaches over this new set of instances.

As expected, this set of 96 more challenging instances makes ILP reach its limits, since it is not able to reach the optimal value in the time limit of 1800 in most of the instances. However, it is remarkable that ILP is able to reach the best known solution in 43 instances in that time limit, although it is not able to guarantee optimality. On the contrary, IG shows its performance by reaching 88 out of 96 best solutions with an average percentage deviation of 1.23%. The reduced value in this metric indicates that, even in those instances in which IG does not reach the best value, it still remains really close to it.

If we now focus on the best previous approach, RLS, it reaches 14 out of 96 best solutions with a deviation of 15.07%. Additionally, the computing time required by RLS is one order of magnitude larger than the current proposal, emerging IG as a competitive algorithm to address the MDSP in terms of both quality and computing time. Finally, the performance of the solver is also remarkable, being able to reach a smaller deviation and a larger number of best solutions than RLS.

The remaining algorithms of the literature, ACO-LS, ACO-PP-LS, and ACO-LS-S, show a similar performance in terms of quality and computing time, being the differences among them neglectable. These results highlight the difficulty of the newly generated instances, where only the best previous approach and the proposed algorithm are competitive for solving the MDSP in reasonable computing times.

Table 8
Ranking resulting from applying non-parametric Friedman statistical test. The obtained p-value is smaller than 0.01.

Algorithm	Mean rank
IG	1.40
ILP	2.24
RLS	2.88
ACO-LS	4.84
ACO-PP-LS	4.83

Table 9
Results obtained when applying the non-parametric Wilcoxon Signed Ranks Test considering all possible pairs derived from the best three approaches, obtaining a p-value smaller than 0.01 in all of them.

ILP–IG	ILP < IG	8
	ILP > IG	53
	ILP = IG	35
RLS–IG	RLS < IG	6
	RLS > IG	76
	RLS = IG	14
RLS–ILP	RLS < ILP	10
	RLS > ILP	63
	RLS = ILP	23

3.3. Analysis

The previous section shows the experimental results obtained when executing the proposed algorithm and the best methods found in the literature over the original set of instances and, also, over a more challenging set generated to evaluate the limits of the algorithms. This section is designed to further analyze the results obtained by the compared algorithms.

First of all, the well-known non-parametric Friedman test is applied, with the aim of ranking the compared algorithms. The resulting p-value smaller than 0.01 indicates that, as expected, there are statistically significant differences among them, resulting in the ranking presented in Table 8.

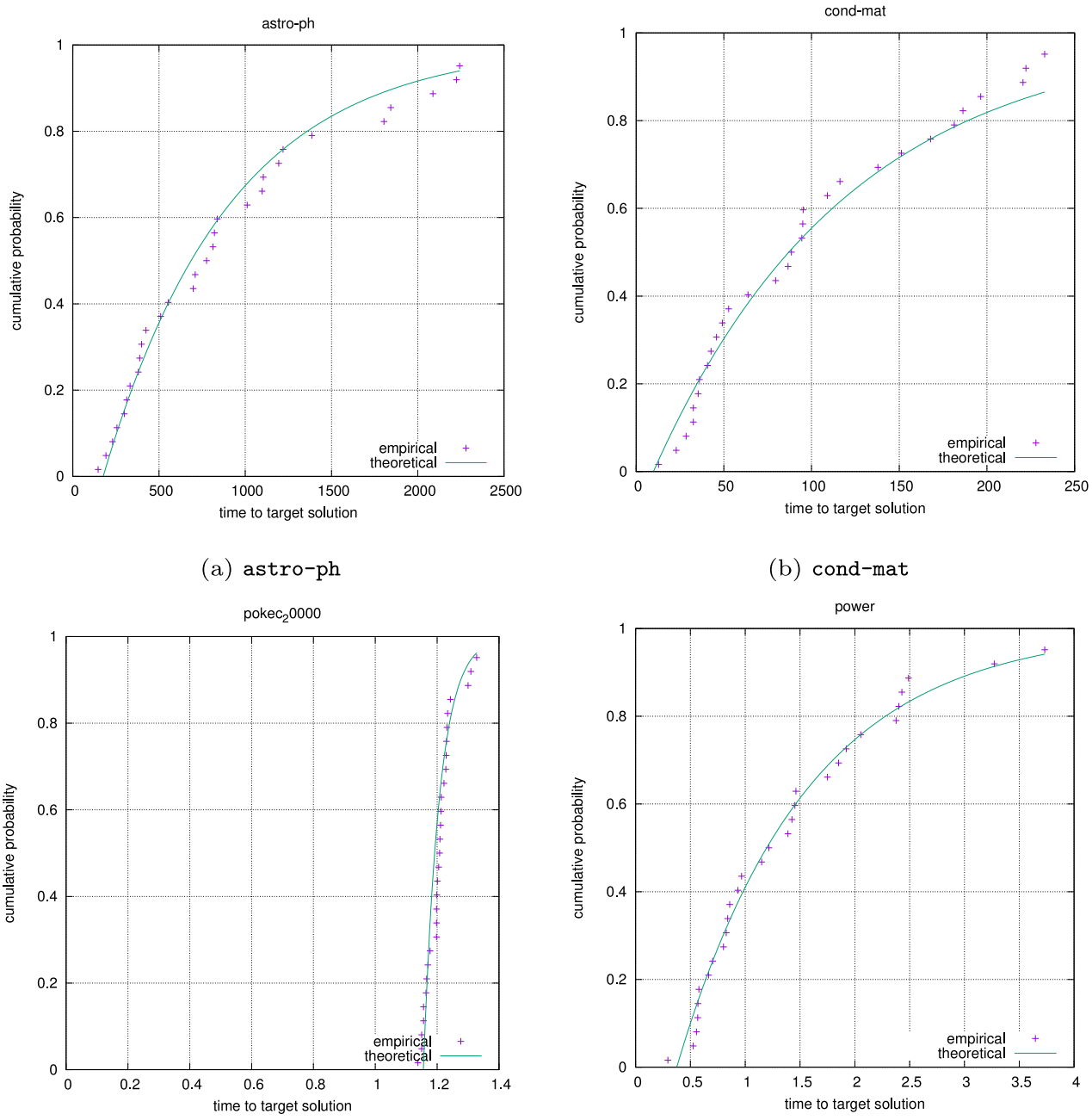
As it can be seen, the first algorithm in the ranking is clearly IG, followed by ILP and RLS, which are close to each other. Finally, the three ACO variants show similar performance, ranking in the last three positions.

Once it is confirmed that the three best algorithms are IG, RLS, and ILP, we now perform the pairwise non-parametric Wilcoxon statistical test to check if the null hypothesis that two of the samples belong to the same population can be rejected. In this case, since three algorithms are compared, we evaluate the three possible pairs derived, namely: IG–RLS, IG–ILP, and RLS–ILP. First of all, the proposed IG is compared with the best heuristic method RLS. Table 9 shows the results obtained in this test:

The resulting p-value smaller than 0.01 indicates that the null hypothesis is rejected, which indicates that IG is statistically better than ILP. Additionally, it can be seen that ILP is able to reach a better solution than IG only in 8 out of 96 instances, while IG outperforms the best solution reached by ILP (in those cases where the optimal value is not reached) in 53 out of 96.

Then, when considering IG versus the best heuristic approach, RLS, the null hypothesis is again rejected with the resulting p-value smaller than 0.01. Therefore, IG emerges as the best approach for solving the MDSP. In this case, RLS outperforms IG in just 6 cases while IG is better than RLS in 76 out of 96 instances.

Finally, it is interesting to evaluate the best previous exact and heuristic approach. In this case, the resulting p-value smaller than 0.01 indicates rejects the null hypothesis, indicating that ILP is statistically better than RLS. Notice that this test does not consider the computing time, so ILP is only suitable for those cases in which the time is not a constraint.



(a) `astro-ph` (b) `cond-mat`

(c) `pokec_20000` (d) `power`

Fig. 6. Time to target plots for four representative instances extracted from the used dataset.

After the discussion of results, it is interesting to analyze the probability of finding the best solution given a certain time horizon, to verify the robustness of the proposal. In order to do so, we use time-to-target plots. The hypothesis is that computing times fit a shifted exponential distribution. Given a certain instance, the algorithm is executed 30 times measuring the computing time required to reach the best solution found by the algorithm in each run. Then, the empirical and theoretical distributions are compared following the graphical methodology for data analysis described in [5]. For each instance, the computing times are ordered in an ascending order and then, each sorted running time is assigned a probability. Fig. 6 shows the cumulative probability distribution for each considered instance.

Fig. 6(a) shows the results when considering the instance `astro-ph`. In this case, it can be seen that, although it requires 2500 s approximately in the worst case to reach the best solution, it can reach it in 1500 s with a probability larger than 80%. In the case of `cond-mat`, depicted in Fig. 6(b), the maximum time required by IG is 250 s, but in 100 s it finds the best solution with a probability of 60%. The `pokec_2000` instance, shown in Fig. 6(c), is a particular case in which most of the runs require similar times, approximately 1.3 s, so there are not specific differences in terms of probability. Finally, the best solution for `power`, see Fig. 6(d), requires 4 s in the worst case, but in 2 s the probability of reaching the best solution is almost 80%. These results show the robustness of the method when considering 30 independent runs.

4. Conclusions

This paper solves an interesting problem, known as the Minimum Dominating Set Problem (MDSP), which models real situations appearing in many different real situations such as, the identification group of people being representatives for all the group, monitoring for a computer network, or influence maximization problem in social networks, among others. The Iterated Greedy is conformed by intuitive procedures to get an initial solution, an efficient local search method to improve it, and simple but intelligent strategies for destructing and reconstructing a solution during the search.

The proposed strategies have been proven to be efficient and effective over the set of 28 instances used in the state of the art. Additionally, new instances have been generated to test the limitations of the exact approach. The presented IG algorithm is able to reach the best values in almost all instances without requiring vast computational efforts, emerging as a competitive algorithm for solving the MDSP.

In the context of the design of metaheuristic algorithms, it is important to highlight the strengths and weaknesses of our proposal. The destruction phase allows the algorithm to escape from local optima, guiding the search to new and close promising regions of the search space, which will eventually lead to improvements. However, the method stagnates where the region explored is a basin of attraction and the global optimum is located in a region which is located far away from the region under exploration.

The knowledge of the MDSP allows us to include intelligent information in the design of the algorithm. First of all, the solution space is drastically reduced since support vertices must be mandatory included in the solution and leaf vertices are never considered in the solution. This is a significant reduction when the number of support and leaf vertices is large, which results in a positive effect in the efficiency of the algorithm. However, the number of leaf and support vertices are not necessarily large in certain types of graphs, therefore this idea will not have any effect in the quality of the generated solutions.

Furthermore, the efficient local search method is able to drastically reduce the number of necessary moves to reach a local optimum with respect to the naïve one. This optimization allows the algorithm to decrease the required computing time, thus being able to solve more complex instances. Although this reduction usually leads to high-quality solutions in reduced computing times, as it has been experimentally tested, it is worth mentioning that the local search may miss to perform some movements that are not originally promising but eventually results in the exploration of new promising regions of the search space.

As future work, it would be interesting to solve different variants of this class of combinatorial optimization problems that bring the theoretical model closer to real-life situations such as considering weights in the selected vertices/edges, or being resilient to failures, among others.

Acknowledgments

A. Casado and J. Sánchez-Oro are supported by the “Ministerio de Ciencia e Innovación, Spain”, Grant Ref. PID2021-125709OA-C22, and by “Comunidad de Madrid” and “Fondos Estructurales” of European Union with Grant Refs. S2018/TCS-4566, Y2018/EMT-5062.

S. Bermudo and A.D. López-Sánchez acknowledge support from the Junta de Andalucía, FEDER-UPO Research & Development Call, Spain, reference number UPO-1263769.

References

[1] Albert-László Barabási, Réka Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[2] Alejandra Casado, Sergio Pérez-Peló, Jesús Sánchez-Oro, Abraham Duarte, A GRASP algorithm with tabu search improvement for solving the maximum intersection of k-subsets problem, J. Heuristics 28 (1) (2022) 121–146.
[3] Pedro Casas-Martínez, Alejandra Casado-Ceballos, Jesús Sánchez-Oro, Eduardo G. Pardo, Multi-objective grasp for maximizing diversity, Electronics 10 (11) (2021) 1232.
[4] David Chalupa, An order-based algorithm for minimum dominating set with application in graph mining, Inf. Sci. 426 (2018) 101–116.
[5] John M. Chambers, William S. Cleveland, Beat Kleiner, Paul A. Tukey, Graphical Methods for Data Analysis, Chapman and Hall/CRC, 2018.
[6] Aron Culotta, Maximizing Cascades in Social Networks: An Overview, 2003.
[7] C.F. De Jaenisch, Applications de L'analuse Mathematique an Jen des Echecs, Petrograd, 1862.
[8] Pedro Domingos, Matt Richardson, Mining the Network Value of Customers, KDD '01, Association for Computing Machinery, New York, NY, USA, 2001, pp. 57–66.
[9] D. Erwin, Dominating broadcasts in graphs, Int. J. Comput. Aided Eng. Technol. 42 (2004) 89–105.
[10] Fedor V. Fomin, Fabrizio Grandoni, Dieter Kratsch, A measure & conquer approach for the analysis of exact algorithms, J. Assoc. Comput. Mach. 56 (5) (2009) 1–32.
[11] Pierre Hansen, Nenad Mladenović, First vs. best improvement: An empirical study, Discrete Appl. Math. 154 (5) (2006) 802–817.
[12] Teresa W. Haynes, Sandra M. Hedetniemi, Stephen T. Hedetniemi, Michael A. Henning, Domination in graphs applied to electric power networks, SIAM J. Discrete Math. 15 (4) (2002) 519–529.
[13] Teresa W. Haynes, S.T. Hedetniemi, Peter J. Slater, Fundamentals of Domination in Graphs, Marcel Dekker, New York, 1998.
[14] Teresa W. Haynes, S.T. Hedetniemi, Peter J. Slater, Domination in Graphs: Advanced Topics, Marcel Dekker, New York, 1998.
[15] Abdel-Rahman Hedar, Rashad Ismail, Hybrid genetic algorithm for minimum dominating set problem, in: ICCSA, 2010.
[16] D. Kempe, J.M. Kleinberg, É. Tardos, Influential nodes in a diffusion model for social networks, in: L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (Eds.), ICALP, in: Automata, Languages and Programming, vol. 3580, Springer, Berlin, Heidelberg, 2005, pp. 1127–1138.
[17] Donald E. Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, third ed., Addison-Wesley, Reading, Mass., 1997.
[18] Wenhan Li, Xiaolong Chen, Junqing Li, Hongyan Sang, Yuyan Han, Shubo Du, An improved iterated greedy algorithm for distributed robotic flowshop scheduling with order constraints, Comput. Ind. Eng. 164 (2022) 107907.
[19] Ana Dolores López-Sánchez, Jesús Sánchez-Oro, Alfredo García Hernández-Díaz, GRASP and VNS for solving the p-next center problem, Comput. Oper. Res. 104 (2019) 295–303.
[20] O. Ore, Theory of Graphs, AMS, Providence, 1962.
[21] Sergio Pérez-Peló, Jesús Sánchez-Oro, Ana Dolores López-Sánchez, Abraham Duarte, A multi-objective parallel iterated greedy for solving the p-center and p-dispersion problem, Electronics 8 (12) (2019).
[22] Anupama Potluri, Alok Singh, Two Hybrid Meta-heuristic Approaches for Minimum Dominating Set Problem, 2011, pp. 97–104.
[23] Anupama Potluri, Alok Singh, Hybrid metaheuristic algorithms for minimum weight dominating set, Appl. Soft Comput. 13 (2013) 76–88.
[24] Juan David Quintana, Raul Martin-Santamaria, Jesus Sanchez-Oro, Abraham Duarte, Solving the regenerator location problem with an iterated greedy approach, Appl. Soft Comput. 111 (2021) 107659.
[25] Ruben Ruiz, Thomas Stutzle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European J. Oper. Res. 177 (3) (2007).
[26] Jesús Sánchez-Oro, Manuel Laguna, Rafael Martí, Abraham Duarte, Scatter search for the bandpass problem, J. Global Optim. 66 (4) (2016) 769–790.
[27] Jesús Sánchez-Oro Calvo, Abraham Duarte, Iterated greedy algorithm for performing community detection in social networks, Future Gen. Comput. Syst. 88 (2018).
[28] Laura A. Sanchis, Experimental analysis of heuristic algorithms for the dominating set problem, Algorithmica 33 (2002) 3–18.
[29] Peng Sun, Xiaoke Ma, Dominating communities for hierarchical control of complex networks, Inf. Sci. 414 (2017) 247–259.
[30] Haynes Teresa W., Hedetniemi Stephen T., Henning Michael A., Structures of Domination in Graphs, Springer Nature, 2021.
[31] Peng-Jun Wan, K.M. Alzoubi, O. Frieder, Distributed construction of connected dominating set in wireless ad hoc networks, in: Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, 2002, pp. 1597–1604.

Capítulo 8

Heuristics for the weighted total domination problem

Los detalles del artículo y la revista en la que está publicado son:

- Casado, A., Sánchez-Oro, J., & Martínez-Gavara, A. (2025). Heuristics for the weighted total domination problem. TOP, 1-42. https://doi.org/10.1007/s11750-025-00695-1
- Estado: **Publicado**
- Factor de impacto (JCR 2024): 1.400
- Posición relativa: 82/106
- Cuartil: Q4

TOP
https://doi.org/10.1007/s11750-025-00695-1

ORIGINAL PAPER

Heuristics for the weighted total domination problem

Alejandra Casado[1] · **Jesús Sánchez-Oro**[1] · **Anna Martínez-Gavara**[2]

Received: 7 July 2024 / Accepted: 13 January 2025

Abstract

The weighted total domination problem (WTDP) belongs to the family of dominating set problems. Given an edge- and vertex- weighted graph, the WTDP consists in selecting a total dominating set D, such that the sum of vertices and edges weights of the subgraph induced by D plus, for each vertex not in D, the minimum weight of its edge to a vertex in D is minimized. A total dominating set D is a subset of the graph's vertices, such that every vertex, including those in D, is at least adjacent to one vertex in D. This problem arises in many real-life applications closely related to covering and independent set problems; however, it remains computationally challenging due to its $\mathcal{NP}$-hardness. This work presents a variable neighborhood search (VNS) procedure to tackle the WTDP, and investigates the advantages and disadvantages of a multi-start strategy within VNS methodology. In addition, we develop a biased greedy randomized adaptive search procedure (Biased GRASP) that keeps adding elements once a feasible solution is found to produce high-quality initial solutions. We perform extensive numerical analysis to look into the influences of the algorithmic components and to disclose the contribution of the elements and strategies of our method. Finally, the empirical analysis shows that our proposal outperforms the state-of-art results, and the statistical analysis confirms the superiority of our proposal to find the best total dominating sets.

Keywords Weighted total domination problem · Graph domination · Biased grasp · Variable neighborhood search · Metaheuristics

✉ Anna Martínez-Gavara
gavara@uv.es
https://www.uv.es/gavara/

Alejandra Casado
alejandra.casado@urjc.es
https://alejandracasado.github.io/

Jesús Sánchez-Oro
jesus.sanchezoro@urjc.es
https://jesussanchezoro.github.io/

[1] Department Computer Science and Statistics, Universidad Rey Juan Carlos, C/Tulipan S/N, 28933 Móstoles, Madrid, Spain

[2] Departament d'Estadística i Investigació Operativa, Universitat de València, C/Doctor Moliner, 50, 46100 Burjassot, Valencia, Spain

Published online: 10 February 2025

Mathematics Subject Classification 68W99 · 90B99

1 Introduction

There are many optimization problems in scientific literature that consist in selecting a subgraph of a given input graph with a topological relation to the unselected vertices. Covering, independent, and dominating set problems are representative examples and constitute a relevant branch within graph theory (Beasley 1987; Beasley and Chu 1996; Haynes et al. 1998; Caprara et al. 2000; Tutte and Tutte 2001). In particular, domination in graphs is an important research area in graph optimization. It is difficult to say when the study of domination in graphs began, but we can say that the publications of Ore (1962), Berge (1962), and Cockayne and Hedetniemi (1977) are among the first references. The literature on domination graph theory is vast from both theoretical and practical perspectives, so we refer the reader to comprehensive surveys and books, such as Haynes et al. (1998), Henning (2009), Haynes et al. (2021), and Haynes et al. (2022b) to mention a few.

There are various types of dominating set problems based on the properties being considered. For instance, concerning the type of connectivity, we can find the independent dominating set, the connected dominating set, or the total dominating set (Guha and Khuller 1998; Goddard and Henning 2013; Cockayne et al. 1980). Neighboring variants emerge if each node needs to have a neighbor or k-neighbors in the dominating set (Hwang and Chang 1991; Corcoran and Gagarin 2021). Moreover, if each node has a limit on the number of neighbors that it has the capacity to dominate, the problem is then called capacitated dominating set (Li et al. 2018). Weighted variants arise when the aim is to minimize weight instead of cardinality (Balakrishnan and Ranganathan 2012). These are just some examples of the multiple variants that can be found in the literature (Haynes et al. 1998; Levin 2020; Haynes et al. 2020). The most relevant real-life applications such as facility location or social networks are shown in Table 1.

In this research paper, we study the edge- and vertex-weighted version of the total domination problem proposed by Ma et al. (2019), in which the objective is to find a total dominating set with a minimum total weight. This problem is called the Weighted Total Domination Problem (WTDP). This is an extension of the Total Domination Problem (TDP) introduced by Cockayne et al. (1980). The TDP is an $\mathcal{NP}$-hard (Laskar et al. 1984) optimization problem that has been widely studied in graph theory (Haynes et al. 1998, 2002a; Henning 2009). It is easy to see that the WTDP is $\mathcal{NP}$-hard, since the TDP is a particular case of WTDP with the edge and vertex weights equal to 0 and 1, respectively. Before defining the WTDP in mathematical terms, we introduce terminology and basic concepts of graph theory.

1.1 Graph theory terminology and basic definitions

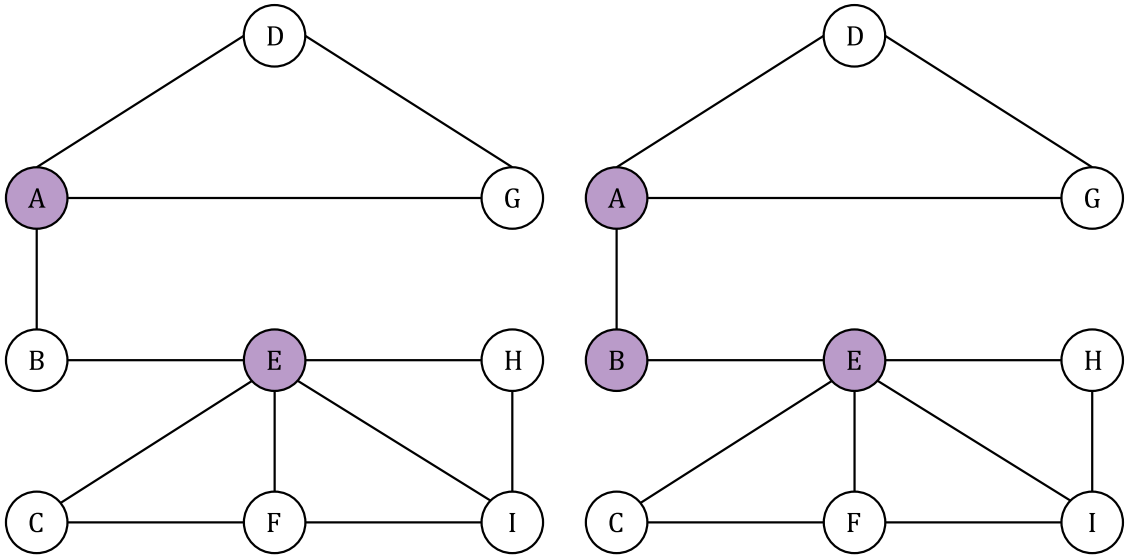
Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Following the same notation as in Haynes et al. (1998), we define the *neighborhood* of a vertex $v \in V$, $N(v)$, as the set of vertices adjacent to v, i.e.,

Table 1 Applications of dominating set and related problems

Context	References	Description
Chess problems	Berge (1962)	To find the minimum number of queens needed to cover or dominate every square on a chess board is one of the fundamental problems in graph theory
School bus routing	Sarubbi et al. (2016)	Location of the minimum number of bus stops for transporting children to and from school
Communication networks	Bai et al. (2020)	Construction of virtual backbone in wireless ad hoc networks
Radio stations	Haynes et al. (1998)	Location of the minimum number of radio stations, so that messages can be broadcast to all cities in the region
Electric power network	Haynes et al. (2002a)	Topology design for a power electricity networked system
Monitoring systems	Henning and Jafari Rad (2012)	Placement of monitoring devices such as fire alarms or surveillance cameras
Facility location	Corcoran and Gagarin (2021)	Location of one or more facilities in street networks
Social Networks	Campan et al. (2015); Wang et al. (2011)	Representation of the relationships between people to ensure a positive influence on the entire social network, or in order to communicate quickly within the network (e.g., in an emergency situation)

$N(v) = \{u \in V : e = (u, v) \in E\}$. We denote as $N[v] := N(v) \cup \{v\}$ the *closed neighborhood* of a vertex v. Note that each edge $e \in E$ connects two different vertices u and v from V, equally denoted by $e = (u, v)$ and by $e = (v, u)$. Given a subset of vertices S, we denote $N(S)$ as the set of neighboring vertices of the set S, i.e., $N(S) = \{v \in N(u),\ \forall u \in S\}$, and $N[S] = N(S) \cup S$. Furthermore, let $E(S)$ denote all the edges in E that have both endpoints in S, and let $G[S]$ be the subgraph induced in G by S. Throughout this paper, we will use vertex and node interchangeably.

Given an undirected graph G, a *dominating set* (DS) is a subset D of V, where each vertex in $V \backslash D$ is adjacent to some vertex from D. Furthermore, a *total dominating set* (TDS) of G with no isolated vertex is a subset $S \subset V$ of the vertices, such that every vertex is adjacent to a vertex in S, that is, $N(S) = V$. The existence of a total dominating set in a graph G requires that all vertices have at least one adjacent vertex, i.e., there are no isolated vertices. If there is no subset of S that can serve as a total dominating set of G, except for S itself, then S is considered to be a minimal total dominating set of G. A total dominating set S differs from a dominating set D in that its members must have adjacent vertices within S. On the other hand, in an ordinary dominating set D, its members may either be in the set or have adjacent vertices within



(a) Dominating set {A, E}. (b) Total dominating set {A, B, E}.

Fig. 1 Example of a dominating set and a total dominating set on a graph G

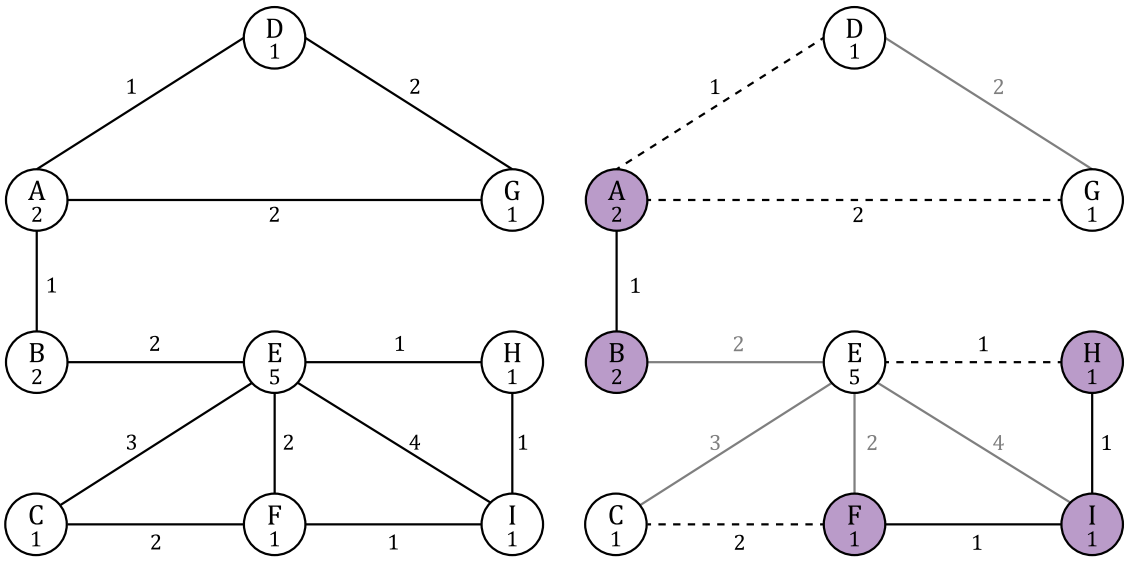
the set. The goal of the *total dominating set problem* is to find the total dominating set with minimum cardinality. Figure 1 shows an undirected graph $G = (V, E)$ with 9 vertices and 12 edges. The dominating set represented in Fig. 1a consists of the vertices with solid background A and E, whereas adding the vertex B a total dominating set is obtained; see Fig. 1b.

1.2 The weighted total domination problem

Consider an edge- and vertex-weighted graph $G = (V, \omega_V, E, \omega_E)$, where V and E are the set of vertices and edges, respectively. Let $\omega_V : V \to \mathbb{R}^+$ be the vertex weight function that maps all vertices onto the set of positive real numbers. Similarly, the edge weight function $\omega_E : E \to \mathbb{R}^+$ assigns a certain weight to each edge e of G. Specifically, $\omega_V(v)$ and $\omega_E(e)$ are the weights of vertex $v \in V$ and edge $e \in E$, respectively. For simplicity without loss of generality, we will use ω instead of ω_V and ω_E whenever there is no ambiguity between them. Then, the *weighted total domination problem* (WTDP) consists in finding a total dominating set S in G with minimum total weight, where the total weight is computed as the following cost function:

$$f(S) := \sum_{u \in S} \omega(u) + \sum_{e \in E(S)} \omega(e) + \sum_{v \in V \setminus S} min\{\omega(v, u) : u \in N(v) \cap S\}. \quad (1)$$

In Fig. 2a, we represent a double-weighted graph G by adding weights in the vertices and edges to the graph of Fig. 1. The label of vertices and their weights are included inside each node, and the edges weights are indicated besides the edges. The subset $S = \{\texttt{A}, \texttt{B}, \texttt{F}, \texttt{H}, \texttt{I}\}$ is the optimal solution of the WTDP of G. In Fig. 2b, vertices in the solution are filled with a solid background. The objective function value is obtained with the sum of the vertices' weights in S $(2 + 2 + 1 + 1 + 1)$, the sum of the edges of



(a) Original weighted graph. (b) Feasible solution.

Fig. 2 Example of a weighted total dominating set on a graph G

the subgraph induced in G by S marked with bold lines $(1 + 1 + 1)$, and the sum of the minimum edge weight between vertices not in S and vertices in S $(1 + 2 + 1 + 2)$, which are marked in bold dashed line. The total cost/weight is 16. Notice that those edges which are not involved in the objective function evaluation have been made less visible in the figure.

Most of the applications of the total domination problem or the weighted dominating set problem can be found in the context of the WTDP (Wang et al. 2011; Henning and Jafari Rad 2012; Zverovich 2021). For instance, the problem of placing a limited number of devices in a communication network, such that every site in the system (including the monitoring transmitters themselves) is adjacent to a monitor, can be modeled by a weighted total domination in graphs. The objective is to ensure that every site in the network can be directly linked to a site that has a transmitter. Furthermore, the placement of a device in each location and the rate of flow (transport service) incur in specific costs. Then, the minimum weight total dominating set problem aims to identify the smallest-weight dominating set in a double-weighted graph, without any restrictions on the size of the dominating set, i.e., number of devices. Note that communication networks usually involve high-operational costs and, therefore, the optimization problem considered here results significant cost reduction for the company providing this communication service.

Regarding the example depicted in Fig. 2, the graph models a network with six devices connected with 12 different links. To have a secure network, we need to locate some transmitters in the network in such a way that every device is directly connected to a transmitter. The weight of each node represents the cost of deploying a transmitter in that node and the weight of each link indicates the cost of transmitting information between the two endpoints of the link. Then, the solution depicted in Fig. 2b represents a network where the traffic flow is guaranteed minimizing the total cost of both deploying the transmitters and communicating them with the devices.

1.3 Literature review

As far as we know, there are only two research papers that deal with the WTDP. In 2019, Ma et al. (2019) proposed the WTDP, which is a combination of the TDP and the minimum weighted dominating set problem, and verify its computational complexity. Furthermore, the authors proposed three integer linear programming (ILP) formulations to solve it using an optimization solver like CPLEX. The benchmark set consists of 9 random graphs generated by the Erdös–Rényi model with different sizes, $|V| = 20, 50, 100$ (three instances per size). The three ILP models are able to find the optimal solution for instances with 20 and 50 vertices within a time limit of 1800 seconds; however, in the same time limit, none of the three models is able to solve the instances with 100 vertices.

In 2021, Álvarez-Miranda and Sinnl (2021) proposed two new Mixed-Integer Programming (MIP) models for the problem including valid inequalities. Moreover, the authors developed a greedy randomized adaptive search procedure (GRASP) and a genetic algorithm (GA) to solve up to 125 nodes. The authors generated 180 instances using the Erdös–Rényi model as in Ma et al. (2019) and considering the following number of vertices $|V| = 20, 50, 75, 100, 125$. An extensive computational experiment revealed that both MIP models are able to find almost all the optimal solutions of instances with $|V| = 100$ vertices within a time limit of 1800s, and some of the instances with $|V| = 125$. Finally, their GRASP and GA are able to find some of the optimal solutions within a short runtime.

These two previous research papers greatly contributed to the development of linear programming models that are able to compute optimal solutions in a reasonable time frame for small size instances. However, they require a significant amount of time in medium-size instances (up to 125) as it is expected due to the $\mathcal{NP}$-hardness of the WTDP. Moreover, the GRASP and GA methods proposed in the previous paper have difficulties in consistently producing the best solutions, and have not been tested in large-size instances. Therefore, there is a need for a strategy to overcome these limitations.

1.4 Contribution and outline

Literature review shows that WTDP is a computationally challenging problem that has been mainly studied from an exact perspective. The practical significance of this problem and its applicability to large networks make the use of heuristics specially suited for it. Our main contributions are summarized as follows:

(i) We propose a metaheuristic procedure based on VNS methodology to obtain high-quality solutions for large-size instances in a reasonable CPU time frame.
(ii) We study different strategies for the construction of high-quality initial solutions. We propose two heuristic algorithms with different greedy functions, and a biased-grasp construction (BGC) algorithm. In the BGC algorithm, the probability of selecting an element is biased toward those that not only have high quality but also contribute to the diversity of the solution. In addition, two versions are considered for each constructive method, the basic (that stops when a

feasible solution is found) and the extended design, which keeps adding elements to the solution.

(iii) We study efficient search strategies to reduce complexity and save computational time in the improvement phase of our proposal. We also study the comparison between a basic VNS design with a multi-start one.

(iv) We perform with statistical tests numerical experiments that find the best strategies and validate our proposals.

The following sections of this paper are organized as follows: Sect. 2 provides a detailed description of our metaheuristics for the WTDP. Section 3 presents computational experiments that first, Sect. 3.1 reveals the most effective strategies, and then, Sect. 3.2 shows the final results of our experimental study. The paper concludes in Sect. 4.

2 Algorithmic approach

As mentioned in Sect. 1.2, the WTDP is an $\mathcal{NP}$-hard problem, which makes exact algorithms not suitable when dealing with large and complex instances. In this research, a metaheuristic based on Variable Neighborhood Search (VNS) is proposed Brimberg et al. (2023). VNS requires starting the search from an initial solution, which can be generated either at random or with a more elaborate procedure. The latter is usually considered in the literature (Casado et al. 2023b) with the aim of providing VNS with a promising starting point, thus reducing the computational effort required to reach high-quality solutions. To that end, we propose three constructive procedures, where two of them are totally greedy methods, while the third method is based on the Greedy Randomized Adaptive Search Procedure (GRASP) framework. In particular, it considers the Biased GRASP version, which assigns a certain probability to each candidate element. The constructive methods will be thoroughly described in Sect. 2.1.

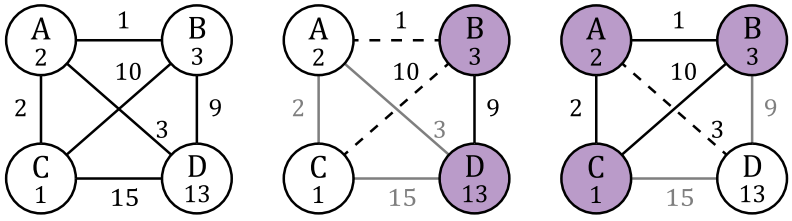
The constructed solutions can be locally improved, since they are not necessarily a local optimum with respect to any neighborhood. A local search method is described in Sect. 2.2 with the aim of reaching a local optimum with respect to the initially generated solution.

Finally, Sect. 2.3 presents the VNS algorithm which is designed for escaping from local optima by perturbing the solutions with a shake procedure. In this research, two different shake approaches are presented to evaluate the impact of diversification and intensification in the VNS framework.

2.1 Solution generation

This section is devoted to present three different constructive approaches to generating high-quality initial solutions for the WTDP. The first two approaches are purely greedy completely focused on quality, while the third approach adds diversity considering the inclusion of randomization to generate better solutions.

The three approaches follow the classical greedy scheme in which the procedure starts from an empty solution and iteratively adds elements until it becomes feasible.



(a) Original graph. (b) $S_1 = \{\texttt{B}, \texttt{D}\}$ (c) $S_2 = \{\texttt{A}, \texttt{B}, \texttt{C}\}$

Fig. 3 Example of a graph with four nodes and two possible solutions for the WTDP

However, regarding the WTDP constraints, it is important to remark that even when a solution is feasible, it is possible to add new elements and continue improving the incumbent solution. Analyzing Eq. (1) of the objective function, it can be divided into three different components: the sum of the weights of the selected nodes, the sum of the edge weights between every pair of selected nodes, and the sum of edge weights between a non-selected node and its adjacent selected node with the smallest edge weight. Including new nodes will increase the value of the two first components, but it may decrease the value of the third one. Therefore, if the decrease of the third component is, in absolute value, larger than the sum of the increase in the two first ones, the objective function will be improved. An example of this behavior can be seen in Fig. 3.

Figure 3a shows the input graph with 4 nodes. The capital letter and the number inside each node indicate its label and weight, respectively, and the number close to each edge denotes the weight of the edge. Solution $S_1 = \{\texttt{B}, \texttt{D}\}$, depicted in Fig. 3b, is conformed with two nodes, and the objective function value is evaluated as $f(S_1) = 16 + 9 + 11 = 36$. If we now analyze solution $S_2 = \{\texttt{A}, \texttt{B}, \texttt{C}\}$, presented in Fig. 3c, the result is $f(S_2) = 6 + 13 + 3 = 22$. Therefore, solution S_2 is better than S_1, even with an additional node in it.

Following this reasoning, a second variant of each constructive method is considered. In this variant, the method continues adding new elements to the solution until all the elements have been included in it. This strategy returns the best feasible solution found so far during the construction process. The main steps of the three constructive methods are described below.

Objective function greedy constructive

The first proposal is a greedy algorithm that selects a node at each iteration based on its objective function value. We refer to this approach as the Objective Function Greedy Constructive Method (OGC). The general scheme is presented in Algorithm 1.

The method starts by creating the solution, S, to which nodes will be iteratively added (step 1). Additionally, solution S_b is created (step 2). This solution is intended to maintain the best solution found, since S becomes feasible until the stopping criterion of the greedy procedure is met. Then, the set of unselected nodes U is initialized with all the nodes in the graph (step 3). The method iteratively selects a node to be included in the solution (steps 4–11) until reaching a stopping criterion.

Algorithm 1 `Greedy constructive`$(G = (V, E))$

1: $S \leftarrow \emptyset$
2: $S_b \leftarrow \emptyset$
3: $U \leftarrow V$
4: **while** $U \neq \emptyset$ **and** $f(S_b) \geq \sum_{e \in E(S)} \omega(e) + \sum_{s \in S} \omega(s)$ **do**
5: $\quad c \leftarrow \arg\min_{u \in U} g_{\texttt{OGC}}(u)$
6: $\quad S \leftarrow S \cup \{c\}$
7: $\quad U \leftarrow U \backslash \{c\}$
8: $\quad$ **if** $f(S) < f(S_b)$ **and** $N(S) = V$ **then**
9: $\quad\quad S_b \leftarrow S$
10: $\quad$ **end if**
11: **end while**
12: **return** S_b

In the context of WTDP, there are two main reasons to stop adding more nodes to a partial solution under construction (step 4). The first one is the simplest: there are no more candidate to add, i.e., all the nodes have already been included in the incumbent solution. The second one tries to reduce the computational effort by stopping the search when it is impossible to add a node without resulting in a solution of worse quality than the best solution found so far. In particular, if the sum of the weights of the selected nodes plus the sum of the internal edge weights is larger than or equal to the objective function value of the best solution found so far, then selecting any unselected node to be included in the incumbent solution will result in a solution with worse quality than S_b.

In each iteration, the node with the best value of a certain greedy criterion is selected (step 5). In this constructive method, the greedy criterion is directly the objective function value of the incumbent solution if the candidate node were included. More formally, $g_{\texttt{OGC}}(u) = f(S \cup \{u\})$. Once a node is selected, it is included in the solution under construction (step 6) and removed from the unselected nodes (step 7). At this point, it is necessary to check if solution S is feasible and better than the best solution found so far S_b (step 8). If so, the best solution found is updated (step 9).

Once the algorithm reaches the stopping criterion, it returns the best solution found during the search S_b (step 12).

Ratio-based greedy constructive

The second constructive procedure, named Ratio-based Greedy Constructive (`RGC`), follows the same scheme as `OGC` but vary the greedy function used in the selection of the next candidate. The main drawback of `OGC` is that it only evaluates the objective function value if the considered candidate is included in the solution. However, it does not necessarily reflect the actual contribution of the node to the objective function value. This is mainly because the external edges involved in the solution evaluation undergo through large variations when adding new nodes, so the selection of the next node in the next iteration will eventually result in a completely different selection of external edges, thus affecting to the objective function value. For instance, regarding solution S_1 depicted in Fig. 3b, the external edge (`B`, `C`) with weight $\omega(\texttt{B}, \texttt{C}) = 10$ is considered. However, if the node `A` were included in the solution, then node `C` would

be connected to the solution by the external edge (A, C), with weight $\omega(\texttt{A}, \texttt{C}) = 2$, thus replacing the external edge (B, C).

With the aim of overcoming this disadvantage, we propose to calculate a ratio between the node and edge weights for each candidate to analyze the relevance of including it in the solution. Notice that, as in OGC, the method will continue adding nodes until all the candidates are included in the solution. Therefore, there are two different situations to be considered during construction: unfeasible and feasible solution. The former refers to a solution which requires from additional nodes to be feasible, while the latter refers to a solution which is already feasible, but there are still more candidates that can be added. In short, given S the incumbent solution in one iteration of the construction algorithm, S is an unfeasible solution if $N(S) \neq V$, and it is feasible, otherwise. Then, the greedy function $g_{\text{RGC}}(u)$ is defined differently depending on the situation.

Given a node $u \in U = V \backslash S$ for the first situation, i.e., S is an unfeasible solution, the ratio evaluates the sum of the weights of the nodes that will be dominated if u were included in the solution, divided by the weight of u plus the sum of the weights of the new internal edges if u is included in the solution. More formally

$$g^u_{\text{RGC}}(u) = \frac{\sum_{v \in N(u) \cap N^c(S) \cap U} \omega(v)}{\omega(u) + \sum_{v \in N(u) \cap S} \omega(u, v)}, \tag{2}$$

where $N^c(S)$ represents the complement of the set of nodes adjacent to the solution S, i.e., $N^c(S) = V \backslash N(S)$. Therefore, $N(u) \cap N^c(S) \cap U$ is the set of unassigned nodes, adjacent to node u, and not dominated by the solution S. Notice that, in this ratio, the weights of the external edges are ignored, since their evaluation is computationally demanding and, additionally, the selection of future nodes will drastically modify the final external edges, as it was aforementioned.

The second situation occurs when the incumbent solution is already feasible. In that situation, if we use the same ratio, the numerator of the division is always equal to zero, since the inclusion of a node in a feasible solution will never dominate new nodes. Therefore, it is necessary to propose a modification of the ratio for feasible solutions. In this case, the modification maintains the weight of internal edges in the denominator. Then, the numerator is modified by including the sum over non-selected nodes adjacent to the candidate node, the weight of minimum edge weight to a selected node. In mathematical terms

$$g^f_{\text{RGC}}(u) = \frac{\sum_{v \in N(u) \cap S^c} \omega(v, S)}{\omega(u) + \sum_{v \in N(u)} \omega(u, v)}, \tag{3}$$

where $\omega(v, S) := \min\{\omega(v, s) : s \in N(v) \cap S\}$.

In both cases, the larger the value, the better the candidate, so step 5 in Algorithm 1 is replaced by $c \leftarrow \arg\max_{u \in U} g_{\text{RGC}}(u)$. Then, the greedy function $g_{\text{RGC}}(u)$ takes the value of $g^u_{\text{RGC}}(u)$ when the incumbent solution is not feasible, and $g^f_{\text{RGC}}(u)$, otherwise.

Biased GRASP constructive

The two previous greedy algorithms lacks of diversity in the construction, since they select the next candidate following a totally greedy criterion. In this third constructive procedure, diversity becomes more relevant by considering randomization in the construction phase.

One of the most popular metaheuristics that combines randomization and greediness is the Greedy Randomized Adaptive Search (GRASP) methodology (Feo and Resende 1989). In the standard implementation of the GRASP construction phase, the next element to be introduced in the solution is chosen at random from the elements in the restricted candidate list. However, instead of the random function other probability distributions can be used to incorporate bias or preference information in the construction of the initial solutions (Resende and Ribeiro 2016). The bias information is used during the solution construction phase, with the aim to guide the construction process toward solutions that are likely to be of higher quality. As it is pointed out in Resende and Ribeiro (2016), these bias functions can be evaluated for all element in the candidate list (CL), or alternatively, their evaluation can be limited to the elements that belong to a restricted candidate list (RCL). There is a vast literature in heuristic methods that uses bias functions in the construction phase, so we refer interested readers to the following relevant research papers on this topic (Bresina 1996; Juan et al. 2013; Grasas et al. 2017; Buxey 1979; Faulin and Juan 2008). Specifically, we follow the biased randomization GRASP constructive framework formulated in Ferone et al. (2019), using an empirical non-uniform probability distribution instead of other well-known skewed (non-symmetric) distributions (e.g., geometric distribution).

Both the standard GRASP method and the bias variants are memoryless techniques. Generally, using bias functions, a weight is assigned to each candidate element based on a quality criterion (e.g., ranking, the objective function value) that does not incorporate information from previous iterations. There may be benefits to preserving such information to develop construction methods that make guided selection within the solution space. We propose adding a frequency memory function that modifies the evaluations of the greedy function to take into account previously recorded information (Resende and Ribeiro 2016; Fleurent and Glover 1999; Napoletano et al. 2019). To do so, each candidate vertex u has associated a ranking function value $g_{\text{BCG}}(u)$ computed as a combination of quality and diversity as follows:

$$g_{\text{BCG}}(u) = \alpha \cdot D + (1 - \alpha) \cdot Q, \tag{4}$$

where D and Q are metrics of diversity and quality, respectively, while $\alpha \in [0, 1]$ is an input parameter of the algorithm that balances both metrics. On the one hand, the quality metric used is calculated as the greedy function g_{RGC} considered in RGC. On the other hand, the diversity metric proposed is computed as the number of solutions in which the node has not been included. This diversity metric tries to prioritize the inclusion of nodes which have not appeared in a large number of solutions, which will eventually lead the method to explore a wider region of the search space. Then, the

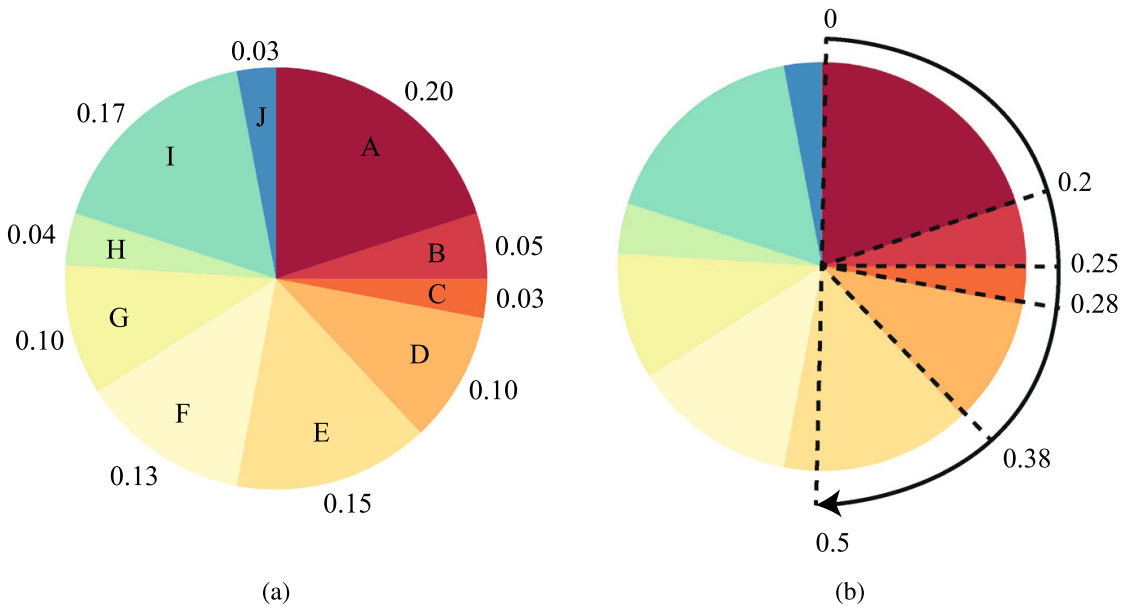


Fig. 4 Example of roulette selection considering 10 nodes

selection probability of a vertex u is set as

$$P(u) = \frac{g_{\texttt{BCG}}(u)}{\sum_{v \in CL} g_{\texttt{BCG}}(v)}. \tag{5}$$

The general framework of our biased GRASP constructive (BGC) method is similar to the RGC method but modifying the selection criterion. In particular, instead of selecting the most promising node according to the greedy function value, in BGC, the next candidate is selected from the *CL* according to the probability distribution. Our proposal differs from the classical framework in that it does not use a restrictive candidate list (RCL) to construct a feasible solution. Instead, our method selects the next vertex to be included in the current solution using a discrete non-uniform distribution based on the probabilities specified in Eq. (5) for all elements in the candidate list. Furthermore, as in the previous greedy constructive methods, BGC continues adding nodes until reaching the stopping criterion described in OGC.

From a computational perspective, the next candidate is selected using the roulette selection method, which effectively reduces computational time. In particular, a probability of being selected is assigned to each node, which is directly evaluated as the greedy function $g_{\texttt{BGC}}$ normalized in the range [0, 1]. Then, a random number in the range [0, 1] is generated. The method then traverses the candidate list accumulating the probability of each node. The selected node will be the first node whose accumulated probability exceeds the random number generated. Notice that, although this selection includes a random selection, the most promising nodes have larger probabilities and, so, they are more likely to be selected than the least promising nodes. Let us illustrate this method with a graphical example depicted in Fig. 4, considering a social network with 10 nodes.

Figure 4a shows an example of the probability assigned to each node. As it can be seen in the figure, the nodes with larger probability are more likely to be selected than the ones with small probability. Then, a random number in the range [0, 1] is obtained,

being 0.51 in this particular example. Figure 4b then selects the first node whose accumulated probability is larger than or equal to 0.51. In particular, the selected node is E, since the accumulated probability is evaluated as $0.20+0.25+0.28+0.38+0.15 = 0.53$.

The inclusion of randomness in this procedure increases diversity, so it is interesting to generate more than a single solution to assure that a wide portion of the search space is explored. As it is customary in GRASP (Campos et al. 2014; Ferone et al. 2019), we perform 100 independent constructions.

2.2 Improvement method

The solution generated by any of the previously described constructive methods is not necessarily a local optimum. Therefore, it is interesting to apply a local improvement method to each generated solution.

The local search method proposed in this research is based on the exchange move operator, which consists of replacing one or more nodes from the solution with another nodes which are not included in it yet. In the particular case of the WTDP, the single exchange operator is considered, which removes a single node from the solution S, replacing it with another one not belonging to the solution ($U = V \backslash S$). More formally,

$$Exchange(S, s, u) \leftarrow (S \backslash \{s\}) \cup \{u\}$$

with $s \in S$ and $u \in U$.

Having defined the move operator, it is now required to indicate the neighborhood that the local improvement method will explore. In particular, the neighborhood is conformed with all the solutions that can be reached by performing a single exchange move. In mathematical terms,

$$N_e(S) = \{S' \leftarrow Exchange(S, s, u) \quad \forall s \in S \wedge \forall u \in U\}$$

.

Finally, it is necessary to define the strategy followed to traverse the proposed neighborhood. There are two main strategies to explore a neighborhood: best and first improvement. In both of them, the stopping criterion is usually the same: the method stops when no improvement is found.

In a best improvement strategy, the neighborhood is completely explored in each iteration, continuing the search with the best solution found in the neighborhood. This strategy is usually rather computationally demanding, since it requires to evaluate the complete neighborhood in each iteration of the local search to select the best neighbor solution.

On the contrary, first improvement strategy performs the first movement that leads to a better solution, even if that solution is not the best one of the neighborhood. This behavior allows the search to reduce the computational effort by avoiding the evaluation of the complete neighborhood.

Each strategy has different advantages and drawbacks. In the case of best improvement, although it always selects the best solution in the neighborhood, the movement

may quickly fall in a basin of attraction, which will eventually result in the stagnation in a local optimum. First improvement, in turn, does not select the best solution in the neighborhood but the first achieving and improve, thus eventually ignoring solutions that can be better than the selected one. However, if this selection is performed with certain randomness, the diversity of the search will increase, reducing the opportunities to fall in a basin of attraction. Additionally, each iteration of first improvement is usually faster than best improvement, since it does not require to evaluate the complete neighborhood.

The local search proposed for the WTDP follows the first improvement approach to reduce the computational effort, since its effectiveness compared with best improvement has been shown in several recent works (Hansen and Mladenović 2006; Casado et al. 2023a). In the context of first improvement, the order in which the neighborhood is explored may affect to the obtained solutions. The proposed local search follows a random exploration to favor diversity. Algorithm 2 presents the pseudocode of the local search method.

Algorithm 2 `Local search`(S)

```
1: improve ← TRUE
2: while improve do
3:    OF ← f(S)
4:    for s ∈ S do
5:       for u ∈ U do
6:          S ← Exchange(S, s, u)
7:          if f(S) < OF and N(S) = V then
8:             improve ← TRUE
9:             go to step 2
10:         end if
11:         S ← Exchange(S, u, s)
12:      end for
13:   end for
14:   improve ← FALSE
15: end while
```

The method receives as input parameter the solution S to be locally improve. Notice that this solution will be modified during the search and, therefore, the local search method does not return any value. The local search iterates, while an improvement is found (steps 2–15). In each iteration, the objective function value of the incumbent solution is stored (step 3). Then, the exploration of the neighborhood starts by randomly traversing the neighborhood (steps 4–13). For each pair of selected and unselected nodes s and u, respectively, the exchange move is performed (step 6). If the resulting solution is feasible and its objective function value is better than the previously stored value (step 7), then an improvement has been found (step 8), so the search is restarted (step 9). Otherwise, it is necessary to undo the exchange move (step 11). If all the neighborhood has been explored, then no improvement is found (step 14), indicating that the search must finish.

It is important to remark that this local search, hereinafter denoted as Exhaustive Local Search (`ELS`), requires to perform the movement, evaluate its quality and, in

case of a non-improvement move, undo the exchange movement, resulting in a very computationally demanding method. To reduce its complexity, we propose two local search variants which include different optimizations.

The first variant, named Predictive Local Search (PLS), tries to reduce the complexity by computing the objective function value obtained with the exchange move but without actually performing the exchange move. If the prediction indicates that the resulting solution would be feasible and better than the incumbent one, then the exchange move is effectively performed. The aim of this optimization is to reduce the number of exchange move performed, thus reducing the total computing time required by the local search method.

The second optimization, named Reduced Local Search (RLS), is designed to explore only those solutions that are feasible when performing an exchange move. In particular, when selecting a node s for the exchange move, not all the nodes $u \in U$ can be considered, since only the inclusion of some of them would result in a feasible solution. In particular, all the nodes adjacent to s which are dominated only by s must be connected to u to guarantee feasibility. Specifically, this optimization result in skipping the execution of steps 6–11 in those nodes that do not satisfy this constraint, thus avoiding the most computationally demanding part of the local search for a considerably large percentage of nodes. The exact reduction in computing time will be later discussed in Sect. 3. Notice that this optimization also includes the one presented in PLS; specifically, the objective function is evaluated without performing the exchange move.

2.3 Variable neighborhood search

Variable Neighborhood Search (VNS) is a metaheuristic framework which leverages systematic changes of neighborhood to escape from local optima. VNS was originally defined in Mladenović and Hansen (1997) and it is in constant evolution, leading to a wide variety of strategies such as Basic VNS, Reduced VNS, Variable Neighborhood Descent, or Variable Formulation Search, among others. We refer the reader to Hansen et al. (2019) for a recent survey on VNS methodology and applications.

The main differences among VNS variants rely on how the considered neighborhoods are explored: from a totally random exploration in the case of Reduced VNS to a completely deterministic search in Variable Neighborhood Descent. In this work, a Jump VNS (JVNS) is proposed, which combines the deterministic search of the local search method with the random exploration of the shake procedure, trying to balance intensification and diversification. Jump VNS (JVNS) was originally presented in Fleszar and Hindi (2004) as a variant of Basic VNS (BVNS). The main difference between JVNS and BVNS lies on the neighborhood change. Classical BVNS sequentially explores the neighborhoods from $k = 1$ to $k_{\max}$, increasing the neighbor in each step by one unit. However, in the context of WTDP, such a small neighborhood change will result in a rather similar solution, thus leading to the same local optimum. With the aim of increase efficiency by avoiding the exploration of the same solutions, JVNS modifies the size of the neighborhood change with an additional input parameter k_{step}.

Algorithm 3 shows the pseudocode of the proposed Jump VNS procedure (JVNS) for the WTDP.

Algorithm 3 JVNS(S, $k_{\max}$, k_{step}, Δ)

1: $S_b \leftarrow S$
2: **for** $i \in 1 \ldots \Delta$ **do**
3: $k \leftarrow k_{step}$
4: **while** $k \leq k_{\max}$ **do**
5: $S' \leftarrow Shake(S_b, k)$
6: $S'' \leftarrow LocalSearch(S')$
7: **if** $f(S'') < f(S_b)$ **then**
8: $S_b \leftarrow S''$
9: $k \leftarrow k_{step}$
10: **else**
11: $k \leftarrow k + k_{step}$
12: **end if**
13: **end while**
14: **end for**
15: **return** S_b

The method receives four input parameters: the initial solution, S; the maximum neighborhood to be explored, $k_{\max}$; the step performed when changing the neighborhood, k_{step}; and the number of VNS iterations executed, Δ. The values for all these parameters are determined in Sect. 3. The procedure starts initializing the best solution found with the initial solution S (step 1). Then, the algorithm performs a fixed number of complete JVNS iterations (step 2–14). Each iteration of JVNS starts initializing the neighborhood k to be explored. Then, the method starts the neighborhoods exploration until reaching the maximum considered neighborhood $k_{\max}$ (steps 4–13). The exploration of a given neighborhood starts with the *Shake* procedure, which is responsible of the diversification in the VNS methodology (step 5). In this research, two different shake procedures are proposed, which are detailedly described in Sect. 2.3. The method generates a solution S' in the current neighborhood k and, since S' is not necessarily a local optimum with respect to any neighborhood, the local search method described in Sect. 2.2 is applied to further improved S', resulting in S'' (step 6). Then, the neighborhood change method is applied, which decides the next neighborhood to be explored (steps 7–12). In particular, if the local optimum S'' outperforms the best solution found so far S_b (step 7), then it is updated (step 8), restarting the search from the first neighborhood (step 9). Otherwise, the method continues with the next considered neighborhood (step 11). Once all the iterations are performed, the method returns the best solution found during the search S_b (step 15).

Shake

The *Shake* procedure proposed in this work follows the classical approach in which random modifications are performed to the incumbent solution to generate a random neighbor solution. This modifications are usually performed by a move operator, which in this case is defined as removing k elements from the solution and then add new

elements until reaching the same stopping criterion as the one defined in Sect. 2.1, resulting in the best solution explored among the feasible ones. The elements removed are selected completely at random, and so the newly added nodes.

Multi-start JVNS

In VNS (and therefore, JVNS), the shake procedure introduces random changes to the current solution, allowing the algorithm to escape local optima and explore different regions of the solution space. The multi-start version of VNS extends this concept by incorporating multiple initial solutions. Instead of starting with a single solution, several diverse solutions are generated and treated as separate initial solutions, and the VNS procedure is run independently. The motivation behind multi-start VNS is to overcome the issue of being trapped in local optima by exploring various parts of the search space.

It is worth noting that a VNS algorithm with an improved perturbation phase can potentially achieve similar exploration capabilities as multi-start VNS. By designing a more effective perturbation strategy, the VNS algorithm can enhance its ability to diversify the search and discover high-quality solutions. For example, the recent study of Casado et al. (2023b) has proposed an intensified shake strategy with successful results.

Nonetheless, the choice between the standard VNS method and the multi-start approach remains a controversial subject. The effectiveness of each algorithm depends on various factors, including the specific problem, the instances being considered, and even the quality of the initial solutions generated. As a result, the selection between VNS and multi-start VNS continues to be debated upon within the optimization community. In the upcoming Sect. 3, we aim to shed light on this debate through experimental analysis.

3 Computational experiments

This section has two main objectives: select the best configuration of the proposed algorithm and then compare it with the best algorithm found in the literature for the WTDP. All the experiments have been performed in an AMD Ryzen 9 5950x (3.4 GHz) with 128GB RAM using Java 17 as programming language. The testbed of instances is divided into two different subsets. On the one hand, there are 180 instances directly derived from the state of the art for the WTDP, with the number of nodes ranging from 20 to 125. This set of instances is divided into two subsets in the literature, named as MA (45 instances) and NEW (135 instances). On the other hand, we propose a new set of 135 more challenging instances, named CSM, with size from 200 to 500 nodes to analyze the limits of the compared algorithms. Both the code of the proposed algorithms and the instances used are publicly available at https://grafo.etsii.urjc.es/wtdp.

The experiments are divided into two different stages: preliminary and final experimentation, Sects. 3.1 and 3.2, respectively. The former is intended to select the best configuration for the proposed algorithms, while the latter is designed to evaluate the

Table 2 Comparison of different values for α parameter in BGC

α	Avg.	Time (s)	Dev. (%)	#Best
0.0	612.84	0.13	0.36	22
0.1	665.28	0.13	7.70	2
0.2	681.24	0.13	10.65	2
0.3	686.68	0.13	12.23	0
0.4	707.16	0.15	15.76	0
0.5	710.08	0.15	16.41	0
0.6	730.76	0.15	19.16	0
0.7	735.04	0.16	19.58	0
0.8	730.60	0.16	20.06	0
0.9	725.24	0.16	19.08	0
RND	677.44	0.15	11.25	0

quality of the proposal when comparing it with the state-of-the-art algorithm for the WTDP. The following metrics are considered in all the experiments: Avg., the average objective function value; Time (s), the average computing time required by the algorithm measured in seconds; Dev. (%), the average deviation with respect to the best solution found in the experiment; and # Best, the number of times that the algorithm reaches the best solution found in the experiment.

3.1 Preliminary experimentation

This section presents the preliminary experiments performed to select the best configuration for the algorithms proposed in this research. With the aim of avoiding overfitting, these experiments have been performed over a subset of 25 representative instances extracted from the original set of 180 instances.

First of all, it is necessary to select the best value for the α parameter for the Biased GRASP constructive, named as BGC. The values tested are selected from $\alpha = 0.0$ to $\alpha = 0.9$, to evaluate the impact of diversification and intensification in the construction phase. Additionally, we have included a variant with $\alpha = RND$, which randomly selects the value for each construction. This variant has been successfully tested in traditional GRASP constructive procedures in some recent research Casado et al. (2022). Table 2 shows the results obtained in this experiment.

As it can be seen in the experiment, when considering the constructive procedure isolatedly, the quality of the solutions obtained is deteriorated with the increase in the α value, while maintaining similar computing times. In particular, the best solutions are found when considering $\alpha = 0.0$, with a deviation of 0.36% and 22 best solutions. It is worth mentioning that increasing the value to $\alpha = 0.1$ drastically worsens the quality of the solutions, obtaining an average deviation of 7.70% and only 2 best solutions. The results of this experiment suggest that the best option for biased GRASP constructive is considering that the probability of selecting a node depends on a totally greedy criterion. Therefore, for the remaining experiments, $\alpha = 0.0$ is considered.

Table 3 Comparison of the constructive procedure when considering the stopping criterion or the feasibility constraint

Constructive	Avg.	Time (s)	Dev. (%)	#Best
BGC (feasibility constraint)	640.80	0.06	5.09	8
BGC (stopping criterion)	612.84	0.13	1.43	19

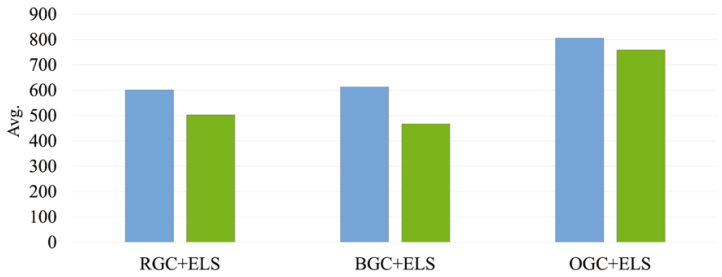


Fig. 5 Comparison of average objective function values (Avg.) obtained by constructive methods executed isolatedly (blue bars, on the left) and coupled with a local search method (green bars, on the right)

The next experiment is devoted to evaluate the impact of considering the stopping criterion described in Sect. 2.1 when comparing it with stopping the construction when reaching a feasible solution. To do so, the constructive selected in the previous experiment is evaluated using the stopping criterion and the feasibility criterion in Table 3.

As expected, the computing time is slightly larger when considering the stopping criterion, but it remains negligible. However, when analyzing the quality of the solutions, it can be seen that the constructive method that considers the stopping criterion is able to reach 19 out of 25 solutions, with a deviation of 1.43%. On the contrary, the one that stops when reaching a feasible solution achieves a deviation of 5.09% and only 8 best solutions, being considerably worse. Therefore, we select the constructive method based on the stopping criterion for the remaining experiments.

The next experiment is designed to analyze the performance of the constructive methods and then coupling them with a local search. Although the quality of one of the constructive methods may be better than the other ones when executing them isolatedly, the best constructive procedure may vary when considering the local search method. The rationale behind this is that those solutions which are initially worse but more diverse can eventually lead to better local optimum when applying a local optimizer. In this experiment, the Exhaustive Local Search (ELS) is considered as local optimizer for each constructive procedure. Notice that RGC and OGC are totally greedy procedures, so they only generate a single solution, while the randomization of BGC allows us to generate 100 independent solutions. Figure 5 shows the results obtained in this experiment. Blue bars (on the left of each algorithm) represent the results obtained by each constructive method and green bars (on the right) represent the results when considering the local search method.

Regarding the results provided by the constructive procedures, it can be seen that both RGC and BGC show similar performance, being RGC slightly better than BGC. Additionally, OGC seems to be the worst constructive method proposed, which highlights the relevance of providing an appropriate greedy function instead of directly evaluating the objective function value. If we now analyze the impact of the local

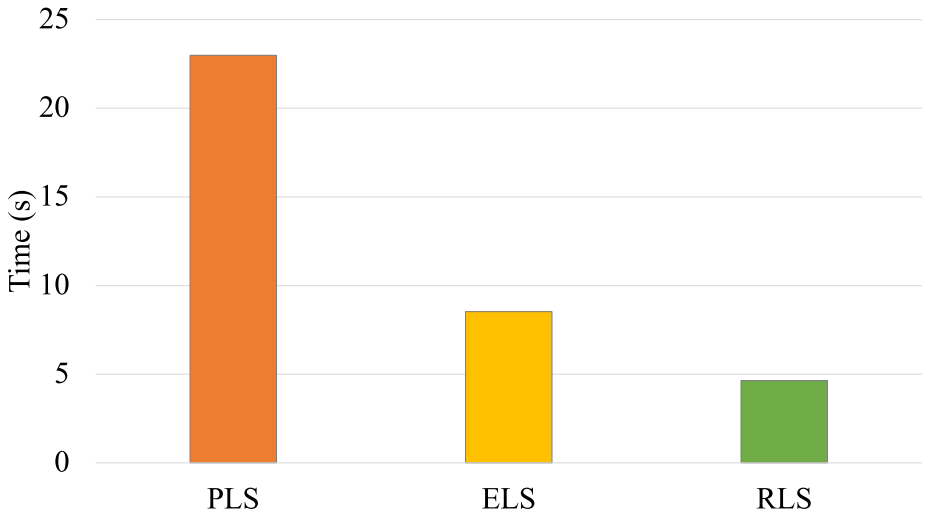


Fig. 6 Comparison of the performance of the three proposed local search methods using BGC as constructive procedure

search in each constructive procedure, it seems that it is not able to perform a relevant improvement in the OGC. However, with respect to RGC and BGC, the local search is able to further improve the obtained solutions. Finally, it is worth mentioning that BGC+ELS provides better quality solutions than RGC+ELS, emerging BGC as the best constructive procedure. The computing times are not included in the comparison, since the differences among algorithms are negligible.

In the last experiment, ELS was considered as the improvement procedure, but it is necessary to evaluate each one of the proposed local search methods: PLS, ELS, and RLS. This analysis is done by comparing only computing times, since the three proposed local search methods explore the same neighborhood in the same order, i.e., PLS and RLS are designed to increase the efficiency but not the efficacy of ELS. Figure 6 shows the average computing time required by the proposed local search when constructing and improving 100 solutions.

The most remarkable aspect of the figure is that PLS requires from more than twice the computing time required by ELS. This is mainly because the prediction performed by PLS to select the most promising moves is more time consuming than performing the move itself and then undo it if it is not an improvement. Therefore, PLS is discarded for the remaining experiments. If we now analyze RLS, it can be seem that it requires half of the computing time of ELS, resulting in an more efficient search. These results indicate that discarding those moves that lead to an unfeasible solution achieves a considerably improvement with respect to the required computing time. We then select RLS as the best local search method for the remaining experimentation.

The next experiment is devoted to select the maximum neighborhood to be explored in JVNS, i.e., the value of $k_{\max}$ parameter. The value of the jump step, k_{step}, is fixed to 0.1, which is 10% of the nodes already in the solution, and the values tested for $k_{\max}$ are $k_{\max} = \{0.2, 0.3, 0.4, 0.5\}$. Larger values are not tested, since it will result in a large modification of the incumbent solution, which may be equivalent to generate a completely new solution. Table 4 shows the results obtained in this experiment.

Table 4 Performance of JVNS when considering different values for the maximum neighborhood k_{max}

k_{max}	Avg.	Time (s)	Dev. (%)	#Best
0.2	464.56	19.59	0.22	22
0.3	464.04	30.66	0.16	23
0.4	463.88	32.13	0.02	24
0.5	463.88	46.48	0.02	24

Table 5 Performance of the multi-start variant of JVNS when considering different values of k_{max}

k_{max}	Avg.	Time (s)	Dev. (%)	#Best
0.2	464.08	34.19	0.31	19
0.3	463.04	47.91	0.11	21
0.4	463.04	57.39	0.11	21
0.5	463.20	73.21	0.15	20

Table 6 Comparison of JVNS with the multi-start approach

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
Multi-start JVNS	463.04	47.91	0.12	21
Standard JVNS	463.88	37.32	0.18	23

Before analyzing the results, it is worth mentioning that the initial solution for JVNS is the best solution found among 100 solutions generated with Biased GRASP (BGC + RLS), and then 100 iterations of JVNS are performed, i.e., $\Delta = 0.1$. As expected, the computing time increases with the value of k_{max}. The quality of the solutions found stagnates when reaching $k_{max} = 0.4$, providing the same results with $k_{max} = 0.5$ but requiring more computing time. For that reason, we select $k_{max} = 0.4$ as the maximum neighborhood to be explored.

Having evaluated the results of JVNS, the next experiment evaluates a multi-start JVNS approach, where a single complete iteration of JVNS is performed for each solution generated with Biased GRASP. Table 5 shows the results obtained in this experiment.

In this experiment, the value of k_{max} is not as relevant as in the previous one, providing similar results for the different configurations in terms of quality. However, the computing time drastically increases with the k_{max} value and, therefore, we select $k_{max} = 0.3$, since it provides a good compromise between quality and computing time.

Having configured the best variant for JVNS and for the multi-start approach, the next experiment compares both algorithms. As in the previous experiments, the standard JVNS performs 100 iterations using as initial solution the best solution found in 100 iterations of Biased GRASP. On the contrary, the multi-start approach performs a single JVNS iteration for each one of the 100 solutions generated by Biased GRASP. Table 6 shows the results obtained in this experiment.

The results show that the multi-start approach is not significantly contributing to the quality of the final solution, providing similar results than the standard JVNS

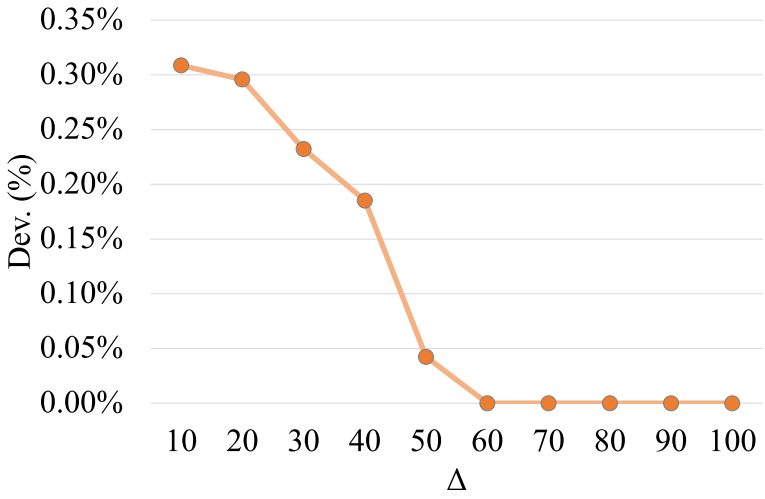


Fig. 7 Profile of the average deviation evolution in steps of 10 iterations for JVNS

Table 7 Comparison of constructive procedure, constructive procedure couple with local search and complete JVNS

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
BGC	612.84	0.13	28.74	0
Biased GRASP	467.00	4.64	0.62	12
JVNS	463.88	24.43	0.00	25

approach. However, it requires considerably more computing time, so we have decided to maintain the standard JVNS for the remaining experiments.

It is important to analyze the evolution of JVNS with the increase in the number of iterations, since it may stagnate in a certain value, performing iterations in which the algorithm will not be able to outperform the quality of the incumbent solution. To select the best number of JVNS iterations, i.e., the value of parameter Δ, Fig. 7 shows a profile of the evolution of the average deviation obtained in steps of 10 iterations.

As it can be seen in the figure, the algorithm finds new improvements until reaching 60 iterations, where it is not able to find further improvements. This indicates that the last 40 iterations are only consuming computing time and, therefore, the maximum number of iterations can be reduced from 100 to 60. Then, for the remaining experiments, the value of Δ is set to 60.

Once the final version of JVNS has been configured, it is important to evaluate the contribution of each part to the complete algorithm. Table 7 shows the comparison among the constructive procedure BGC, the constructive procedure coupled with the local search Biased GRASP and, finally, the complete JVNS algorithm.

Regarding the computing time, the addition of new components to the algorithm also affects to the computational time, being JVNS the slowest algorithm and BGC the fastest one. A similar but inverse analysis can be done when considering the quality. The initial solutions provided by BGC remain at a considerably average deviation with respect to the best ones found in the experiment, but the inclusion of the local search drastically reduces the average deviation. However, JVNS is required to reach the best solutions of the experiment. Notice that, although Biased GRASP has a small deviation (0.62%), the number of best solutions found is smaller than half of the

Table 8 Comparison of the initial solutions of JVNS and GA1, generated with Biased GRASP and GRASP, respectively

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
Biased GRASP	523.20	2.89	0.07	123
GRASP	539.96	2.95	2.69	55

best solutions found by JVNS. These results highlight that JVNS is a key part of the proposal.

3.2 Final experimentation

The final set of experiments is devoted to performing a thorough comparison of the proposed JVNS with the best algorithms found in the state of the art. In particular, the best method found in the literature for the WTDP is a Genetic Algorithm (GA1) where the initial population is generated with a traditional GRASP algorithm and each individual is improved with a local search method Álvarez-Miranda and Sinnl (2021). In that research, authors also propose an exact approach, but they conclude that GA1 is the best algorithm for the WTDP. Notice that, in this section, the complete set of instances is used in each experiment. The hardware used in that research is equivalent or even better than the one considered in this work, resulting in a fair comparison. We refer the reader to the appendix to see the individual results of each algorithm over every single instance (Tables 12, 13, 14, 15).

The first experiment is designed to compare the quality of the initial solution in both JVNS and GA1 approaches. As it was aforementioned, the previous work considers a GRASP algorithm, while the initial solution for JVNS is generated by the proposed Biased GRASP method. The results of the previous algorithm have been directly transcribed from the original manuscript. Table 8 shows the results obtained in this experiment.

Analyzing the computing time, both algorithms required similar computational time. In terms of quality, Biased GRASP shows its superiority by reaching 123 out of 135 instances, with an average deviation of 0.07%. This deviation indicates that, in those instances in which biased GRASP is not able to reach the best value, it still remains close to it. It is worth mentioning that the GRASP proposed in the state of the art also shows a good performance in terms of deviation, but it is not able to reach more than 55 best solutions. We have performed a non-parametric pairwise Wilcoxon statistical test to evaluate if there are statistically significant differences between both results, and the obtained p value smaller than 0.01 confirms this hypothesis.

The best work in the literature proposes four different exact methods for solving the WTDP, evaluating their performance over the set of MA instances, which are less challenging than the NEW set for an exact approach. We have tested JVNS over this set of instances to evaluate how far from the optimal value JVNS is when it is known. Table 9 shows the results obtained in this experiment.

As it can be derived from the results, the best exact procedure is f2+, which is able to reach all the optimal values in reasonable computing times when compared with the other exact approaches. If we analyze the results of JVNS, we can clearly see that the

Table 9 Comparison of JVNS with the exact methods over the set of MA instances

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
JVNS	95.35	3.48	0.03	44
f1	95.51	449.08	0.11	42
f1+	95.33	157.28	0.00	45
f2	95.33	151.22	0.00	45
f2+	95.33	51.04	0.00	45

Table 10 Comparison of JVNS and GA1 over the NEW testbed proposed in the original work

Number of nodes	Algorithm	Avg.	Time (s)	Dev. (%)	#Best
75	JVNS	421.78	4.43	0.02	43
	GA1	423.87	9.91	0.34	39
100	JVNS	525.53	11.14	0.05	42
	GA1	526.51	23.13	0.23	36
125	JVNS	611.31	22.83	0.12	39
	GA1	611.80	47.76	0.20	37

computing time is an order of magnitude smaller than the best exact method, and two orders of magnitude smaller than the other exact approaches. Indeed, in only 3 s on average, JVNS is able to reach 44 out of 45 instances with a deviation of 0.03%. This value indicates that in the only instance in which it is not able to reach the optimal value, JVNS finds a high-quality local optimum. In this case, the Wilcoxon statistical test results in a p value of 0.317, indicating that there are not statistically significant differences between both results, highlighting the performance of JVNS.

The next experiment is devoted to compare JVNS with the best previous heuristic approach, which is the Genetic Algorithm (GA1) proposed in Álvarez-Miranda and Sinnl (2021). In this case, the set of more challenging NEW instances is considered, which was originally proposed by the previous authors with the aim of highlighting the necessity of a non-exact approach for the WTDP. Table 10 depicts the comparison between these two algorithms.

The results are divided by the number of nodes of the considered instances to better analyze the performance of the algorithms. Regarding the computing times, JVNS is consistently requiring half of the computing time required by GA1. In terms of quality, JVNS reaches almost all the best solutions in each subset, showing the smallest deviation with respect to the best solution found, while GA1 also shows a good performance. The Wilcoxon statistical test results in a p value smaller than 0.01, confirming the superiority of JVNS.

Since both MA and NEW sets seem to be easily solved by both JVNS and GA1, mainly due to the small size of the considered instances for a heuristic approach, we propose a new set of larger and more challenging instances with nodes ranging from 200 to 500, named CSM. With the aim of having a fair comparison, we contacted the authors for their original source code, but, unfortunately, it was not available.

Table 11 Comparison of JVNS and GA1 in the set of more challenging instances CSM

Number of nodes	Algorithm	Avg.	Time (s)	Dev. (%)	#Best
200	JVNS	871.40	100.73	0.00	45
	GA1	937.08	962.77	6.72	1
350	JVNS	1332.04	616.34	0.00	45
	GA1	1794.06	1824.58	29.79	0
500	JVNS	1776.40	1164.68	0.00	45
	GA1	2539.57	1900.48	37.56	0

For that reason, we carefully reimplemented the GA1 algorithm following the detailed description of the original work. Table 11 shows the results obtained in this comparison.

As it can be seen in the results, JVNS performs better when the instances become more complex, showing its scalability when comparing it with GA1. It is worth mentioning the increase in the computing for both approaches, which highlights the difficulty of solving this new set of instances. In spite of this increase, JVNS is still requiring, approximately, half of the computing time than GA1. In terms of quality, it seems that GA1 reaches its limits when solving instances with 200 nodes, since for larger instances, the average deviation drastically increases and it is not able to reach any best solution. On the contrary, JVNS emerges as a competitive algorithm for the WTDP, finding all the best solutions for this new set of challenging instances. Finally, the p value smaller than 0.01 obtained in the Wilcoxon statistical test indicates that there are statistically significant differences between JVNS and GA1.

4 Conclusions and future research

In this research paper, we study the $\mathcal{NP}$-hard Weighted Total Dominating Set Problem (WTDP), which extends the Total Dominating Set Problem (TDP) by incorporating weights to the vertices and edges of the graph. This practical problem has applications in areas, such as facility location, social networks, and communications networks where the number of required open facilities, users, or devices is often large. To test our proposed methodology in realistic scenarios, we generated large-size instances.

The merit of our study goes beyond the mere application of a methodology to a problem, as we delve into deeper analysis and investigation of the methodologies themselves. As such, the main contributions of this research paper are mainly methodological. We proposed a metaheuristic procedure based on the Jump Variable Neighborhood Search (JVNS) methodology to obtain high-quality solutions for large-size instances within a reasonable CPU time frame. We also studied efficient search strategies to compare basic JVNS design with a multi-start one.

Additionally, we analyzed different strategies for the construction of high-quality initial solutions, proposing two heuristic algorithms with different greedy functions and a biased GRASP construction algorithm. In the latter, two stopping criteria, the basic design and the extended, were empirically compared to investigate the impact of

both criteria on the constructed solution. Furthermore, three different variants of the local search procedures based on exchange moves have been studied to obtain a desired trade-off between solution quality and computation time. Finally, we performed statistical tests and numerical experiments to validate our proposals, which showed that our proposed methods and strategies outperformed existing state-of-the-art approaches.

To sum up, our research makes significant contributions to the field of metaheuristic optimization, particularly in the context of the WTDP. Our findings provide a valuable basis for developing further improvements in this area, with potential applications in communication and social networks, among others.

As a future work, we propose the investigation of matheuristic approaches to tackle the WTDP. Matheuristics combine mathematical programming techniques with heuristic methods, such as metaheuristics, to improve the efficiency and effectiveness of the optimization process. We believe that the combination of mathematical models and heuristic search can lead to better solutions for large instances of the WTDP in a reasonable computational time. This would allow us to find high-quality solutions that are guaranteed to be near-optimal.

Moreover, in this research paper, we have focused on finding the dominating set with the smallest weight. As mentioned in the Introduction, this problem arises in communication networks, where reducing the number of transmitters can also lead to decreased radiation emission. For future research, we plan to transform the problem into a multi-criteria minimization problem where the weight assigned to each vertex in the graph represents a cost or nuisance measurement parameter, like the noise level at the transmitter site. Then, the aim is to design an efficient metaheuristic to identify the smallest-size dominating set in a weighted graph that incurs the lowest cost.

Appendix

See Tables 12, 13, 14, and 15.

Table 12 Individual results obtained by the initial solutions of JVNS and GA1 (generated with biased GRASP and GRASP, respectively) for each instance in the NEW testbed

Instance	Biased GRASP		GRASP	
	Avg.	Time(s)	Avg.	Time(s)
NEW-75—0.2-10-50-1	690	1.88	769	1.00
NEW-75—0.2-10-50-2	784	2.02	871	1.00
NEW-75—0.2-10-50-3	662	1.93	765	1.00
NEW-75—0.2-10-50-4	746	1.79	762	1.00
NEW-75—0.2-10-50-5	766	1.90	857	1.00
NEW-75—0.2-25-25-1	502	1.38	556	1.00
NEW-75—0.2-25-25-2	546	1.55	607	1.00
NEW-75—0.2-25-25-3	518	1.44	603	1.00
NEW-75—0.2-25-25-4	503	1.47	521	1.00
NEW-75—0.2-25-25-5	513	1.45	526	1.00
NEW-75—0.2-50-10-1	340	1.30	340	1.00
NEW-75—0.2-50-10-2	382	1.22	414	1.00
NEW-75—0.2-50-10-3	335	1.18	341	1.00
NEW-75—0.2-50-10-4	333	1.22	338	1.00
NEW-75—0.2-50-10-5	348	1.17	353	1.00
NEW-75—0.5-10-50-1	581	1.52	590	1.00
NEW-75—0.5-10-50-2	602	1.43	641	1.00
NEW-75—0.5-10-50-3	554	1.34	545	1.00
NEW-75—0.5-10-50-4	540	1.30	580	1.00
NEW-75—0.5-10-50-5	530	1.38	551	1.00
NEW-75—0.5-25-25-1	387	1.04	402	1.00
NEW-75—0.5-25-25-2	384	1.23	413	1.00
NEW-75—0.5-25-25-3	362	1.09	380	1.00
NEW-75—0.5-25-25-4	366	1.08	371	1.00
NEW-75—0.5-25-25-5	331	1.19	331	1.00
NEW-75—0.5-50-10-1	240	0.69	244	1.00
NEW-75—0.5-50-10-2	238	0.63	245	1.00
NEW-75—0.5-50-10-3	215	0.73	215	1.00
NEW-75—0.5-50-10-4	235	0.68	235	1.00
NEW-75—0.5-50-10-5	206	0.57	206	1.00
NEW-75—0.8-10-50-1	571	0.90	613	2.00
NEW-75—0.8-10-50-2	520	0.82	520	2.00
NEW-75—0.8-10-50-3	543	0.86	543	2.00
NEW-75—0.8-10-50-4	571	0.90	571	2.00
NEW-75—0.8-10-50-5	509	0.86	509	2.00
NEW-75—0.8-25-25-1	357	0.77	360	2.00
NEW-75—0.8-25-25-2	338	0.73	356	2.00
NEW-75—0.8-25-25-3	323	0.75	323	2.00

Table 12 continued

Instance	Biased GRASP		GRASP	
	Avg.	Time(s)	Avg.	Time(s)
NEW-75−0.8-25-25-4	345	0.82	345	2.00
NEW-75−0.8-25-25-5	311	0.77	311	2.00
NEW-75−0.8-50-10-1	182	0.43	182	2.00
NEW-75−0.8-50-10-2	188	0.38	188	2.00
NEW-75−0.8-50-10-3	191	0.38	191	2.00
NEW-75−0.8-50-10-4	196	0.46	196	2.00
NEW-75−0.8-50-10-5	192	0.42	192	2.00
NEW-100−0.2-10-50-1	897	5.10	930	1.00
NEW-100−0.2-10-50-2	961	4.81	983	1.00
NEW-100−0.2-10-50-3	895	5.23	905	1.00
NEW-100−0.2-10-50-4	846	4.85	879	1.00
NEW-100−0.2-10-50-5	840	5.35	907	1.00
NEW-100−0.2-25-25-1	596	3.04	591	1.00
NEW-100−0.2-25-25-2	657	3.47	687	1.00
NEW-100−0.2-25-25-3	615	3.37	648	1.00
NEW-100−0.2-25-25-4	558	3.63	602	1.00
NEW-100−0.2-25-25-5	619	3.15	646	1.00
NEW-100−0.2-50-10-1	418	2.30	422	1.00
NEW-100−0.2-50-10-2	447	2.43	472	1.00
NEW-100−0.2-50-10-3	420	3.34	427	1.00
NEW-100−0.2-50-10-4	403	2.70	418	1.00
NEW-100−0.2-50-10-5	378	2.42	379	1.00
NEW-100−0.5-10-50-1	758	3.23	749	2.00
NEW-100−0.5-10-50-2	700	3.30	705	3.00
NEW-100−0.5-10-50-3	724	3.02	730	3.00
NEW-100−0.5-10-50-4	754	3.14	775	2.00
NEW-100−0.5-10-50-5	726	3.28	743	2.00
NEW-100−0.5-25-25-1	461	2.83	461	3.00
NEW-100−0.5-25-25-2	437	2.92	448	2.00
NEW-100−0.5-25-25-3	434	2.35	443	3.00
NEW-100−0.5-25-25-4	489	2.33	489	2.00
NEW-100−0.5-25-25-5	462	2.71	470	3.00
NEW-100−0.5-50-10-1	260	1.45	260	2.00
NEW-100−0.5-50-10-2	271	1.26	271	2.00
NEW-100−0.5-50-10-3	283	1.33	283	3.00
NEW-100−0.5-50-10-4	291	1.73	296	2.00
NEW-100−0.5-50-10-5	269	1.33	269	2.00
NEW-100−0.8-10-50-1	734	1.88	730	4.00

Table 12 continued

Instance	Biased GRASP		GRASP	
	Avg.	Time(s)	Avg.	Time(s)
NEW-100−0.8-10-50-2	688	1.75	688	4.00
NEW-100−0.8-10-50-3	718	1.94	718	4.00
NEW-100−0.8-10-50-4	712	2.09	709	4.00
NEW-100−0.8-10-50-5	703	1.85	710	4.00
NEW-100−0.8-25-25-1	442	1.64	452	5.00
NEW-100−0.8-25-25-2	430	1.69	430	4.00
NEW-100−0.8-25-25-3	426	1.79	426	4.00
NEW-100−0.8-25-25-4	428	1.63	428	4.00
NEW-100−0.8-25-25-5	432	1.64	432	4.00
NEW-100−0.8-50-10-1	259	0.83	259	4.00
NEW-100−0.8-50-10-2	246	0.95	246	4.00
NEW-100−0.8-50-10-3	238	0.90	238	4.00
NEW-100−0.8-50-10-4	253	0.96	258	4.00
NEW-100−0.8-50-10-5	248	0.85	250	5.00
NEW-125−0.2-10-50-1	1031	10.39	1112	2.00
NEW-125−0.2-10-50-2	1070	10.20	1069	2.00
NEW-125−0.2-10-50-3	958	10.72	1124	2.00
NEW-125−0.2-10-50-4	1072	9.74	1121	2.00
NEW-125−0.2-10-50-5	979	10.93	1112	2.00
NEW-125−0.2-25-25-1	735	7.24	803	2.00
NEW-125−0.2-25-25-2	751	8.19	768	2.00
NEW-125−0.2-25-25-3	733	7.22	752	2.00
NEW-125−0.2-25-25-4	707	7.25	726	2.00
NEW-125−0.2-25-25-5	692	7.57	747	2.00
NEW-125−0.2-50-10-1	455	4.80	457	2.00
NEW-125−0.2-50-10-2	481	5.38	493	2.00
NEW-125−0.2-50-10-3	490	5.12	501	2.00
NEW-125−0.2-50-10-4	468	5.13	504	2.00
NEW-125−0.2-50-10-5	457	4.84	468	2.00
NEW-125−0.5-10-50-1	817	6.58	817	4.00
NEW-125−0.5-10-50-2	821	6.11	827	5.00
NEW-125−0.5-10-50-3	878	5.64	880	4.00
NEW-125−0.5-10-50-4	867	6.31	914	4.00
NEW-125−0.5-10-50-5	871	6.27	906	5.00
NEW-125−0.5-25-25-1	573	5.06	566	5.00
NEW-125−0.5-25-25-2	533	5.36	561	5.00

Table 12 continued

Instance	Biased GRASP		GRASP	
	Avg.	Time(s)	Avg.	Time(s)
NEW-125−0.5-25-25-3	538	4.81	567	5.00
NEW-125−0.5-25-25-4	557	4.65	565	4.00
NEW-125−0.5-25-25-5	548	4.96	548	5.00
NEW-125−0.5-50-10-1	336	2.68	336	4.00
NEW-125−0.5-50-10-2	330	2.59	330	4.00
NEW-125−0.5-50-10-3	315	2.74	315	5.00
NEW-125−0.5-50-10-4	316	2.41	316	5.00
NEW-125−0.5-50-10-5	313	2.59	311	4.00
NEW-125−0.8-10-50-1	793	3.85	793	9.00
NEW-125−0.8-10-50-2	860	3.51	854	8.00
NEW-125−0.8-10-50-3	787	3.81	829	9.00
NEW-125−0.8-10-50-4	777	3.96	829	9.00
NEW-125−0.8-10-50-5	836	3.56	827	8.00
NEW-125−0.8-25-25-1	508	3.44	521	9.00
NEW-125−0.8-25-25-2	498	3.27	499	9.00
NEW-125−0.8-25-25-3	523	3.04	523	9.00
NEW-125−0.8-25-25-4	495	3.45	506	8.00
NEW-125−0.8-25-25-5	512	2.81	519	8.00
NEW-125−0.8-50-10-1	309	1.67	307	8.00
NEW-125−0.8-50-10-2	296	2.05	296	8.00
NEW-125−0.8-50-10-3	297	1.80	294	8.00
NEW-125−0.8-50-10-4	270	2.01	270	9.00
NEW-125−0.8-50-10-5	278	1.87	278	9.00

Table 13 Individual results obtained by the exacts methods f1, f1+, f2, f2+ and the proposed algorithm JVNS for each instance in the MA testbed

Instance	JVNS		f1		f1+		f2		f2+	
	OF	Time(s)	OF	Time(s)	OF	Time(s)	OF	Time(s)	OF	Time(s)
MA-20−0.2-5-5-1	63	0.06	63	1.00	63	1.00	63	1.00	63	1.00
MA-20−0.2-5-5-2	58	0.04	58	1.00	58	1.00	58	1.00	58	1.00
MA-20−0.2-5-5-3	58	0.05	58	3.00	58	1.00	58	1.00	58	1.00
MA-20−0.2-5-5-4	51	0.06	51	1.00	51	1.00	51	1.00	51	1.00
MA-20−0.2-5-5-5	55	0.04	55	1.00	55	1.00	55	1.00	55	1.00
MA-20−0.5-5-5-1	44	0.04	44	1.00	44	1.00	44	1.00	44	1.00
MA-20−0.5-5-5-2	47	0.04	47	1.00	47	1.00	47	1.00	47	1.00
MA-20−0.5-5-5-3	46	0.04	46	1.00	46	1.00	46	1.00	46	1.00
MA-20−0.5-5-5-4	40	0.04	40	1.00	40	1.00	40	1.00	40	1.00
MA-20−0.5-5-5-5	41	0.04	41	1.00	41	1.00	41	1.00	41	1.00
MA-20−0.8-5-5-1	37	0.04	37	1.00	37	1.00	37	1.00	37	1.00
MA-20−0.8-5-5-2	35	0.04	35	3.00	35	1.00	35	1.00	35	1.00
MA-20−0.8-5-5-3	40	0.04	40	1.00	40	1.00	40	1.00	40	1.00
MA-20−0.8-5-5-4	34	0.03	34	1.00	34	1.00	34	1.00	34	1.00
MA-20−0.8-5-5-5	34	0.04	34	1.00	34	1.00	34	1.00	34	1.00
MA-50−0.2-5-5-1	111	1.31	111	2.00	111	1.00	111	4.00	111	1.00
MA-50−0.2-5-5-2	106	1.32	106	3.00	106	1.00	106	3.00	106	1.00
MA-50−0.2-5-5-3	111	1.35	111	8.00	111	1.00	111	4.00	111	1.00
MA-50−0.2-5-5-4	101	1.34	101	4.00	101	1.00	101	4.00	101	1.00
MA-50−0.2-5-5-5	108	1.37	108	13.00	108	1.00	108	6.00	108	1.00
MA-50−0.5-5-5-1	82	0.84	82	4.00	82	3.00	82	3.00	82	2.00

Table 13 continued

Instance	JVNS		f1		f1+		f2		f2+	
	OF	Time(s)	OF	Time(s)	OF	Time(s)	OF	Time(s)	OF	Time(s)
MA-50−0.5-5-5-2	85	0.73	85	5.00	85	2.00	85	3.00	85	2.00
MA-50−0.5-5-5-3	85	0.75	84	22.00	84	3.00	84	4.00	84	2.00
MA-50−0.5-5-5-4	82	1.06	82	16.00	82	3.00	82	4.00	82	2.00
MA-50−0.5-5-5-5	82	0.72	82	15.00	82	4.00	82	4.00	82	2.00
MA-50−0.8-5-5-1	77	0.38	77	6.00	77	7.00	77	5.00	77	4.00
MA-50−0.8-5-5-2	72	0.44	72	3.00	72	3.00	72	3.00	72	2.00
MA-50−0.8-5-5-3	74	0.39	74	3.00	74	3.00	74	4.00	74	2.00
MA-50−0.8-5-5-4	76	0.39	76	6.00	76	5.00	76	4.00	76	3.00
MA-50−0.8-5-5-5	79	0.36	79	13.00	79	15.00	79	7.00	79	7.00
MA-100−0.2-5-5-1	175	18.70	175	898.00	175	92.00	175	566.00	175	50.00
MA-100−0.2-5-5-2	174	19.04	174	251.00	174	14.00	174	276.00	174	12.00
MA-100−0.2-5-5-3	177	19.36	178	1800.00	177	239.00	177	1434.00	177	121.00
MA-100−0.2-5-5-4	169	17.81	169	1800.00	169	81.00	169	562.00	169	37.00
MA-100−0.2-5-5-5	167	17.32	171	1800.00	167	97.00	167	1473.00	167	47.00
MA-100−0.5-5-5-1	147	7.21	147	1800.00	147	304.00	147	292.00	147	108.00
MA-100−0.5-5-5-2	144	6.56	144	707.00	144	158.00	144	152.00	144	51.00
MA-100−0.5-5-5-3	147	6.64	147	995.00	147	401.00	147	186.00	147	128.00
MA-100−0.5-5-5-4	146	7.33	149	1800.00	146	725.00	146	289.00	146	214.00
MA-100−0.5-5-5-5	139	7.42	139	1800.00	139	466.00	139	242.00	139	155.00
MA-100−0.8-5-5-1	136	3.35	136	1655.00	136	346.00	136	172.00	136	97.00
MA-100−0.8-5-5-2	140	2.94	140	759.00	140	894.00	140	283.00	140	249.00
MA-100−0.8-5-5-3	141	3.31	141	1212.00	141	1032.00	141	236.00	141	325.00
MA-100−0.8-5-5-4	141	3.22	141	1800.00	141	1652.00	141	334.00	141	495.00
MA-100−0.8-5-5-5	134	3.27	134	990.00	134	509.00	134	231.00	134	160.00

Table 14 Individual results obtained by JVNS and GA1 for each instance in the NEW testbed

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
NEW-75−0.2-10-50-1	686	9.08	686	5.00
NEW-75−0.2-10-50-2	770	10.36	794	6.00
NEW-75−0.2-10-50-3	661	9.81	661	6.00
NEW-75−0.2-10-50-4	705	9.35	740	7.00
NEW-75−0.2-10-50-5	758	9.64	779	6.00
NEW-75−0.2-25-25-1	498	6.73	504	6.00
NEW-75−0.2-25-25-2	546	5.48	546	6.00
NEW-75−0.2-25-25-3	518	6.02	518	5.00
NEW-75−0.2-25-25-4	500	8.85	498	6.00
NEW-75−0.2-25-25-5	513	9.12	513	6.00
NEW-75−0.2-50-10-1	339	6.74	339	6.00
NEW-75−0.2-50-10-2	382	7.21	382	5.00
NEW-75−0.2-50-10-3	335	6.23	341	5.00
NEW-75−0.2-50-10-4	333	7.37	333	6.00
NEW-75−0.2-50-10-5	348	6.40	347	6.00
NEW-75−0.5-10-50-1	581	4.25	581	13.00
NEW-75−0.5-10-50-2	602	4.17	602	11.00
NEW-75−0.5-10-50-3	545	4.36	545	10.00
NEW-75−0.5-10-50-4	540	4.53	540	10.00
NEW-75−0.5-10-50-5	519	4.63	519	10.00
NEW-75−0.5-25-25-1	387	4.04	387	10.00
NEW-75−0.5-25-25-2	384	3.81	384	10.00
NEW-75−0.5-25-25-3	362	4.02	362	10.00
NEW-75−0.5-25-25-4	366	3.60	371	9.00
NEW-75−0.5-25-25-5	331	3.84	331	10.00
NEW-75−0.5-50-10-1	240	2.00	240	9.00
NEW-75−0.5-50-10-2	238	2.40	238	9.00
NEW-75−0.5-50-10-3	215	2.68	215	9.00
NEW-75−0.5-50-10-4	235	3.36	235	9.00
NEW-75−0.5-50-10-5	206	3.18	206	8.00
NEW-75−0.8-10-50-1	571	2.19	571	16.00
NEW-75−0.8-10-50-2	520	2.12	520	15.00
NEW-75−0.8-10-50-3	543	2.39	543	15.00
NEW-75−0.8-10-50-4	571	2.28	571	15.00
NEW-75−0.8-10-50-5	509	2.12	509	17.00
NEW-75−0.8-25-25-1	357	1.87	357	15.00
NEW-75−0.8-25-25-2	338	1.80	338	15.00
NEW-75−0.8-25-25-3	323	2.07	323	13.00

Table 14 continued

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
NEW-75−0.8-25-25-4	345	2.16	345	13.00
NEW-75−0.8-25-25-5	311	2.00	311	15.00
NEW-75−0.8-50-10-1	182	1.10	182	14.00
NEW-75−0.8-50-10-2	188	0.94	188	11.00
NEW-75−0.8-50-10-3	191	0.96	191	11.00
NEW-75−0.8-50-10-4	196	1.04	196	12.00
NEW-75−0.8-50-10-5	192	0.98	192	15.00
NEW-100−0.2-10-50-1	873	20.45	873	12.00
NEW-100−0.2-10-50-2	944	25.17	944	13.00
NEW-100−0.2-10-50-3	878	22.57	878	11.00
NEW-100−0.2-10-50-4	837	22.34	837	11.00
NEW-100−0.2-10-50-5	840	24.68	870	12.00
NEW-100−0.2-25-25-1	591	24.36	591	12.00
NEW-100−0.2-25-25-2	653	22.61	655	11.00
NEW-100−0.2-25-25-3	615	22.84	616	12.00
NEW-100−0.2-25-25-4	552	22.47	552	11.00
NEW-100−0.2-25-25-5	613	22.97	607	12.00
NEW-100−0.2-50-10-1	418	16.64	420	12.00
NEW-100−0.2-50-10-2	447	16.74	456	11.00
NEW-100−0.2-50-10-3	420	17.85	419	11.00
NEW-100−0.2-50-10-4	403	14.97	410	12.00
NEW-100−0.2-50-10-5	375	18.17	379	13.00
NEW-100−0.5-10-50-1	758	8.41	749	26.00
NEW-100−0.5-10-50-2	700	10.29	700	25.00
NEW-100−0.5-10-50-3	718	8.19	718	24.00
NEW-100−0.5-10-50-4	726	8.82	726	26.00
NEW-100−0.5-10-50-5	702	8.65	702	25.00
NEW-100−0.5-25-25-1	461	9.02	461	25.00
NEW-100−0.5-25-25-2	437	9.21	437	19.00
NEW-100−0.5-25-25-3	434	9.52	434	22.00
NEW-100−0.5-25-25-4	482	9.19	482	25.00
NEW-100−0.5-25-25-5	456	9.12	457	23.00
NEW-100−0.5-50-10-1	260	7.57	260	22.00
NEW-100−0.5-50-10-2	271	4.81	271	21.00
NEW-100−0.5-50-10-3	283	7.88	283	21.00
NEW-100−0.5-50-10-4	291	6.00	291	22.00

Table 14 continued

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
NEW-100−0.5-50-10-5	269	7.48	269	21.00
NEW-100−0.8-10-50-1	730	6.03	730	39.00
NEW-100−0.8-10-50-2	683	5.76	683	37.00
NEW-100−0.8-10-50-3	718	4.62	718	37.00
NEW-100−0.8-10-50-4	709	5.84	709	41.00
NEW-100−0.8-10-50-5	700	5.18	704	39.00
NEW-100−0.8-25-25-1	442	4.16	442	40.00
NEW-100−0.8-25-25-2	430	4.04	430	32.00
NEW-100−0.8-25-25-3	426	4.11	426	36.00
NEW-100−0.8-25-25-4	428	4.05	428	35.00
NEW-100−0.8-25-25-5	432	4.16	432	42.00
NEW-100−0.8-50-10-1	259	2.72	259	32.00
NEW-100−0.8-50-10-2	246	2.87	246	9.00
NEW-100−0.8-50-10-3	238	2.72	238	34.00
NEW-100−0.8-50-10-4	253	3.09	253	34.00
NEW-100−0.8-50-10-5	248	2.76	248	31.00
NEW-125−0.2-10-50-1	1031	43.17	1026	24.00
NEW-125−0.2-10-50-2	1046	58.66	1038	22.00
NEW-125−0.2-10-50-3	935	59.19	947	23.00
NEW-125−0.2-10-50-4	1068	41.43	1051	21.00
NEW-125−0.2-10-50-5	974	62.42	975	25.00
NEW-125−0.2-25-25-1	720	47.17	720	26.00
NEW-125−0.2-25-25-2	751	46.71	748	24.00
NEW-125−0.2-25-25-3	715	38.54	717	21.00
NEW-125−0.2-25-25-4	701	51.23	705	22.00
NEW-125−0.2-25-25-5	685	41.35	697	23.00
NEW-125−0.2-50-10-1	455	28.84	455	21.00
NEW-125−0.2-50-10-2	477	34.22	477	23.00
NEW-125−0.2-50-10-3	490	48.71	490	21.00
NEW-125−0.2-50-10-4	467	30.64	467	23.00
NEW-125−0.2-50-10-5	457	42.30	459	24.00
NEW-125−0.5-10-50-1	817	18.41	817	41.00
NEW-125−0.5-10-50-2	815	18.83	815	45.00
NEW-125−0.5-10-50-3	836	18.40	872	45.00
NEW-125−0.5-10-50-4	867	15.69	867	55.00
NEW-125−0.5-10-50-5	867	17.30	867	55.00
NEW-125−0.5-25-25-1	566	17.01	566	48.00

Table 14 continued

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
NEW-125−0.5-25-25-2	533	15.43	533	48.00
NEW-125−0.5-25-25-3	538	14.25	538	49.00
NEW-125−0.5-25-25-4	552	15.44	552	53.00
NEW-125−0.5-25-25-5	548	14.35	548	48.00
NEW-125−0.5-50-10-1	334	12.43	334	40.00
NEW-125−0.5-50-10-2	330	12.02	330	38.00
NEW-125−0.5-50-10-3	315	11.15	315	49.00
NEW-125−0.5-50-10-4	316	13.56	316	51.00
NEW-125−0.5-50-10-5	311	13.86	311	40.00
NEW-125−0.8-10-50-1	793	10.99	793	78.00
NEW-125−0.8-10-50-2	845	10.82	845	72.00
NEW-125−0.8-10-50-3	787	11.36	787	74.00
NEW-125−0.8-10-50-4	777	11.80	777	83.00
NEW-125−0.8-10-50-5	827	10.61	813	77.00
NEW-125−0.8-25-25-1	508	7.91	510	69.00
NEW-125−0.8-25-25-2	498	7.75	498	65.00
NEW-125−0.8-25-25-3	513	7.95	513	77.00
NEW-125−0.8-25-25-4	495	8.13	493	75.00
NEW-125−0.8-25-25-5	504	9.07	504	76.00
NEW-125−0.8-50-10-1	307	5.30	307	64.00
NEW-125−0.8-50-10-2	296	6.18	296	57.00
NEW-125−0.8-50-10-3	294	5.72	294	71.00
NEW-125−0.8-50-10-4	270	5.56	270	86.00
NEW-125−0.8-50-10-5	278	5.58	278	77.00

Table 15 Individual results obtained by JVNS and GA1 for each instance in the CSM testbed

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
CSM_200_0.2_10_50_0	600	144.80	654	770.15
CSM_200_0.2_10_50_1	602	161.99	692	744.12
CSM_200_0.2_10_50_2	600	182.17	675	692.04
CSM_200_0.2_10_50_3	616	131.27	684	766.95
CSM_200_0.2_10_50_4	569	139.64	658	778.11
CSM_200_0.2_25_25_0	981	217.21	1161	970.28
CSM_200_0.2_25_25_1	976	240.23	1110	1005.88
CSM_200_0.2_25_25_2	960	209.94	1107	1070.22
CSM_200_0.2_25_25_3	961	212.59	1044	1088.88
CSM_200_0.2_25_25_4	986	223.07	1154	947.46
CSM_200_0.2_50_10_0	1448	222.21	1692	1369.55
CSM_200_0.2_50_10_1	1469	261.30	1766	1219.78
CSM_200_0.2_50_10_2	1474	251.46	1681	1204.24
CSM_200_0.2_50_10_3	1388	256.15	1610	1292.35
CSM_200_0.2_50_10_4	1395	250.85	1650	1098.83
CSM_200_0.5_10_50_0	452	43.83	468	901.42
CSM_200_0.5_10_50_1	459	42.29	475	925.97
CSM_200_0.5_10_50_2	468	42.78	469	877.50
CSM_200_0.5_10_50_3	485	47.17	495	876.27
CSM_200_0.5_10_50_4	473	39.85	503	855.06
CSM_200_0.5_25_25_0	777	57.11	809	977.23
CSM_200_0.5_25_25_1	789	56.17	813	941.31
CSM_200_0.5_25_25_2	762	57.52	782	1021.13
CSM_200_0.5_25_25_3	795	60.33	808	1008.91
CSM_200_0.5_25_25_4	810	62.45	814	1045.16
CSM_200_0.5_50_10_0	1260	73.20	1278	1046.46
CSM_200_0.5_50_10_1	1219	70.46	1294	1108.63
CSM_200_0.5_50_10_2	1212	95.07	1231	1058.00
CSM_200_0.5_50_10_3	1192	71.99	1267	1005.29
CSM_200_0.5_50_10_4	1199	71.46	1274	1007.62
CSM_200_0.8_10_50_0	415	28.51	419	892.54
CSM_200_0.8_10_50_1	438	26.81	451	867.71
CSM_200_0.8_10_50_2	421	26.61	429	891.12
CSM_200_0.8_10_50_3	437	28.02	450	906.00
CSM_200_0.8_10_50_4	413	29.21	425	883.29
CSM_200_0.8_25_25_0	734	36.57	748	929.82
CSM_200_0.8_25_25_1	714	40.10	744	913.50

Table 15 continued

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
CSM_200_0.8_25_25_2	717	36.53	729	902.35
CSM_200_0.8_25_25_3	738	37.17	773	916.45
CSM_200_0.8_25_25_4	761	40.67	786	920.69
CSM_200_0.8_50_10_0	1208	40.26	1229	927.94
CSM_200_0.8_50_10_1	1230	39.92	1246	929.90
CSM_200_0.8_50_10_2	1216	40.61	1226	923.72
CSM_200_0.8_50_10_3	1197	43.95	1197	922.46
CSM_200_0.8_50_10_4	1197	41.54	1199	922.77
CSM_350_0.2_10_50_0	927	1014.89	1073	1803.19
CSM_350_0.2_10_50_1	862	858.24	1058	1804.19
CSM_350_0.2_10_50_2	895	1090.68	1070	1818.24
CSM_350_0.2_10_50_3	896	960.16	1051	1804.46
CSM_350_0.2_10_50_4	887	1023.91	1068	1817.50
CSM_350_0.2_25_25_0	1440	1269.64	1934	1805.28
CSM_350_0.2_25_25_1	1459	1262.89	2010	1806.73
CSM_350_0.2_25_25_2	1412	1288.04	1910	1808.76
CSM_350_0.2_25_25_3	1394	1322.82	1956	1804.63
CSM_350_0.2_25_25_4	1438	1424.21	1980	1813.41
CSM_350_0.2_50_10_0	2279	1741.24	3433	1819.43
CSM_350_0.2_50_10_1	2156	1554.72	3494	1812.11
CSM_350_0.2_50_10_2	2215	1483.53	3320	1820.55
CSM_350_0.2_50_10_3	2214	1530.55	3556	1803.45
CSM_350_0.2_50_10_4	2157	1551.85	3471	1805.70
CSM_350_0.5_10_50_0	700	244.40	860	1839.04
CSM_350_0.5_10_50_1	713	228.98	858	1827.00
CSM_350_0.5_10_50_2	676	250.13	869	1848.07
CSM_350_0.5_10_50_3	696	230.46	850	1831.88
CSM_350_0.5_10_50_4	725	201.95	878	1818.23
CSM_350_0.5_25_25_0	1171	457.92	1807	1842.03
CSM_350_0.5_25_25_1	1227	452.56	1837	1827.81
CSM_350_0.5_25_25_2	1179	433.61	1611	1841.12
CSM_350_0.5_25_25_3	1177	416.77	1749	1833.80
CSM_350_0.5_25_25_4	1193	431.12	1697	1838.18
CSM_350_0.5_50_10_0	1900	477.80	3212	1837.24
CSM_350_0.5_50_10_1	1933	549.72	3066	1840.80
CSM_350_0.5_50_10_2	1959	490.02	3328	1837.95
CSM_350_0.5_50_10_3	1963	537.02	2900	1831.29

Table 15 continued

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
CSM_350_0.5_50_10_4	1959	464.76	2970	1807.96
CSM_350_0.8_10_50_0	650	131.58	682	1843.37
CSM_350_0.8_10_50_1	632	135.53	679	1846.13
CSM_350_0.8_10_50_2	641	141.96	684	1812.22
CSM_350_0.8_10_50_3	631	139.10	688	1826.25
CSM_350_0.8_10_50_4	644	135.21	669	1841.98
CSM_350_0.8_25_25_0	1133	157.19	1214	1833.01
CSM_350_0.8_25_25_1	1111	174.04	1208	1847.53
CSM_350_0.8_25_25_2	1091	186.70	1247	1850.23
CSM_350_0.8_25_25_3	1125	169.01	1230	1807.24
CSM_350_0.8_25_25_4	1092	166.26	1182	1867.19
CSM_350_0.8_50_10_0	1858	196.68	2110	1826.85
CSM_350_0.8_50_10_1	1883	192.70	2011	1820.90
CSM_350_0.8_50_10_2	1876	181.90	2074	1810.52
CSM_350_0.8_50_10_3	1889	177.40	2093	1808.97
CSM_350_0.8_50_10_4	1884	205.55	2086	1814.04
CSM_500_0.2_10_50_0	1135	1830.06	1392	1819.39
CSM_500_0.2_10_50_1	1141	1813.15	1376	1802.85
CSM_500_0.2_10_50_2	1170	1806.12	1452	1832.91
CSM_500_0.2_10_50_3	1136	1819.35	1483	1812.93
CSM_500_0.2_10_50_4	1139	1803.09	1449	1831.61
CSM_500_0.2_25_25_0	1932	1819.89	2843	1893.62
CSM_500_0.2_25_25_1	1854	1835.34	2785	1891.55
CSM_500_0.2_25_25_2	1886	1829.45	2786	1808.43
CSM_500_0.2_25_25_3	1879	1853.40	2516	1803.62
CSM_500_0.2_25_25_4	1946	1831.72	2761	1818.01
CSM_500_0.2_50_10_0	3108	1848.74	4769	1893.00
CSM_500_0.2_50_10_1	2967	1835.35	5013	1800.93
CSM_500_0.2_50_10_2	3011	1811.88	5227	1824.67
CSM_500_0.2_50_10_3	2989	1800.92	4905	1894.16
CSM_500_0.2_50_10_4	2931	1814.70	5037	1826.73
CSM_500_0.5_10_50_0	957	851.29	1322	1866.73
CSM_500_0.5_10_50_1	902	1020.22	1236	1854.97
CSM_500_0.5_10_50_2	927	900.02	1274	1859.99
CSM_500_0.5_10_50_3	938	889.09	1227	1871.50
CSM_500_0.5_10_50_4	949	688.51	1309	1854.16

Table 15 continued

Instance	JVNS		GA1	
	OF	Time(s)	OF	Time(s)
CSM_500_0.5_25_25_0	1586	1357.61	2360	1853.90
CSM_500_0.5_25_25_1	1565	1328.90	2720	1852.22
CSM_500_0.5_25_25_2	1550	1310.22	2465	1859.24
CSM_500_0.5_25_25_3	1590	1234.01	2769	1864.02
CSM_500_0.5_25_25_4	1569	1192.46	2600	1872.73
CSM_500_0.5_50_10_0	2587	1404.38	4557	1829.10
CSM_500_0.5_50_10_1	2659	1338.04	4580	1831.23
CSM_500_0.5_50_10_2	2634	1410.40	4357	1823.80
CSM_500_0.5_50_10_3	2553	1446.41	4235	1843.98
CSM_500_0.5_50_10_4	2631	1427.80	4609	1834.04
CSM_500_0.8_10_50_0	864	394.51	949	2002.55
CSM_500_0.8_10_50_1	876	373.37	917	2023.97
CSM_500_0.8_10_50_2	891	357.42	963	2013.56
CSM_500_0.8_10_50_3	855	363.48	914	1998.57
CSM_500_0.8_10_50_4	860	358.12	919	2039.70
CSM_500_0.8_25_25_0	1475	562.28	1665	1978.82
CSM_500_0.8_25_25_1	1487	503.48	1563	2042.10
CSM_500_0.8_25_25_2	1464	514.17	1681	2035.73
CSM_500_0.8_25_25_3	1478	510.39	1606	2033.05
CSM_500_0.8_25_25_4	1486	522.88	1604	2007.98
CSM_500_0.8_50_10_0	2493	557.17	2868	2025.01
CSM_500_0.8_50_10_1	2458	582.73	2840	2003.86
CSM_500_0.8_50_10_2	2442	543.08	2830	1994.57
CSM_500_0.8_50_10_3	2514	555.86	2842	1998.85
CSM_500_0.8_50_10_4	2474	559.22	2706	1997.36

Acknowledgements This research has been partially supported by the Ministerio de Ciencia e Innovación of Spain (Grant Ref. PID2021–125709OB-C21 and PID2021–125709OA-C22) funded by MCIN/AEI /10.13039/501100011033 / FEDER, UE. It has been also supported by the Generalitat Valenciana (CIAICO/2021/224).

Author Contributions Alejandra Casado: algorithm design, algorithm implementation, and writing. Jesús Sánchez-Oro: methodology, algorithm design, and writing—original draft preparation. Anna Martínez-Gavara: methodology, algorithm design, and writing.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability All the data, including instances, results, and source code, are publicly available at https://grafo.etsii.urjc.es/wtdp.

References

Álvarez-Miranda E, Sinnl M (2021) Exact and heuristic algorithms for the weighted total domination problem. Comput Oper Res 127:105157

Bai X, Zhao D, Bai S, Wang Q, Li W, Mu D (2020) Minimum connected dominating sets in heterogeneous 3d wireless ad hoc networks. Ad Hoc Netw 97:102023

Balakrishnan R, Ranganathan K (2012) A textbook of graph theory. Springer, Berlin

Beasley JE (1987) An algorithm for set covering problem. Eur J Oper Res 31:85–93

Beasley JE, Chu PC (1996) A genetic algorithm for the set covering problem. Eur J Oper Res 94:392–404

Berge C (1962) The theory of graphs and its applications. Methuen & co. Ltd, London

Bresina JL (1996) Heuristic-biased stochastic sampling. AAAI/IAAI 1:271–278

Brimberg J, Salhi S, Todosijević R, Urošević D (2023) Variable neighborhood search: the power of change and simplicity. Comput Oper Res 155:106221

Buxey G (1979) The vehicle scheduling problem and monte carlo simulation. J Oper Res Soc 30:563–573

Campos V, Martí R, Sánchez-Oro J, Duarte A (2014) Grasp with path relinking for the orienteering problem. J Oper Res Soc 65:1800–1813

Campan A, Truta TM, Beckerich M (2015) Fast dominating set algorithms for social networks. In: Midwest Artificial Intelligence and Cognitive Science Conference, pp. 55–62

Caprara A, Toth P, Fischetti M (2000) Algorithms for the set covering problem. Ann Oper Res 98:353–371

Casado A, Pérez-Peló S, Sánchez-Oro J, Duarte A (2022) A grasp algorithm with tabu search improvement for solving the maximum intersection of k-subsets problem. J Heuris 28:121–146

Casado A, Bermudo S, López-Sánchez A, Sánchez-Oro J (2023) An iterated greedy algorithm for finding the minimum dominating set in graphs. Math Comput Simul 207:41–58

Casado A, Mladenović N, Sánchez-Oro J, Duarte A (2023) Variable neighborhood search approach with intensified shake for monitor placement. Networks 81:319–333

Cockayne EJ, Hedetniemi ST (1977) Towards a theory of domination in graphs. Networks 7:247–261

Cockayne EJ, Dawes R, Hedetniemi ST (1980) Total domination in graphs. Networks 10:211–219

Corcoran P, Gagarin A (2021) Heuristics for k-domination models of facility location problems in street networks. Comput Oper Res 133:105368

Faulin J, Juan AA (2008) The algacea-1 method for the capacitated vehicle routing problem. Int Trans Oper Res 15:599–621

Feo TA, Resende MG (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8:67–71

Ferone D, Gruler A, Festa P, Juan AA (2019) Enhancing and extending the classical grasp framework with biased randomisation and simulation. J Oper Res Soc 70:1362–1375

Fleszar K, Hindi KS (2004) Solving the resource-constrained project scheduling problem by a variable neighbourhood search. Eur J Oper Res 155: 402–413. (Financial Risk in Open Economies)

Fleurent C, Glover F (1999) Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. INFORMS J Comput 11:198–204

Goddard W, Henning MA (2013) Independent domination in graphs: a survey and recent results. Disc Math 313:839–854

Grasas A, Juan AA, Faulin J, De Armas J, Ramalhinho H (2017) Biased randomization of heuristics using skewed probability distributions: a survey and some applications. Comput Ind Eng 110:216–228

Guha S, Khuller S (1998) Approximation algorithms for connected dominating sets. Algorithmica 20:374–387

Hansen P, Mladenović N, Brimberg J, Pérez JAM (2019) Variable neighborhood search. Springer, Berlin

Hansen P, Mladenović N (2006) First vs. best improvement: An empirical study. Discrete Appl Math 154: 802–817. (IV ALIO/EURO Workshop on Applied Combinatorial Optimization)

Haynes TW, Hedetniemi S, Slater P (1998) Fundamentals of domination in graphs. CRC Press, Boca Raton

Haynes TW, Hedetniemi ST, Henning MA (2020) Topics in domination in graphs, vol 64. Springer, Berlin
Haynes TW, Hedetniemi ST, Henning MA et al (2021) Structures of domination in graphs, vol 66. Springer, Berlin
Haynes TW, Hedetniemi SM, Hedetniemi ST, Henning MA (2002) Domination in graphs applied to electric power networks. SIAM J Disc Math 15:519–529
Haynes TW, Hedetniemi ST, Henning MA (2022) Domination in graphs: core concepts. Manuscript. Springer, New York
Henning MA (2009) A survey of selected recent results on total domination in graphs. Disc Math 309:32–63
Henning MA, Jafari Rad N (2012) Locating-total domination in graphs. Disc Appl Math 160:1986–1993
Hwang SF, Chang GJ (1991) The k-neighbor domination problem. Eur J Oper Res 52:373–377
Juan AA, Faulin J, Ferrer A, Lourenço HR, Barrios B (2013) Mirha: multi-start biased randomization of heuristics with adaptive local search for solving non-smooth routing problems. Top 21:109–132
Laskar R, Pfaff J, Hedetniemi S, Hedetniemi S (1984) On the algorithmic complexity of total domination. SIAM J Algeb Disc Methods 5:420–425
Levin MS (2020) On combinatorial optimization for dominating sets (literature survey, new models). arXiv preprint arXiv:2009.09288
Li R, Hu S, Zhao P, Zhou Y, Yin M (2018) A novel local search algorithm for the minimum capacitated dominating set. J Oper Res Soc 69:849–863
Ma Y, Cai Q, Yao S (2019) Integer linear programming models for the weighted total domination problem. Appl Math Comput 358:146–150
Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100
Napoletano A, Martínez-Gavara A, Festa P, Pastore T, Martí R (2019) Heuristics for the constrained incremental graph drawing problem. Eur J Oper Res 274:710–729
Ore O (1962) Theory of graphs. In: Colloquium Publications. American Mathematical Society
Resende MG, Ribeiro CC (2016) Optimization by GRASP—greedy randomized adaptive search procedures. Springer, Berlin
Sarubbi JF, Mesquita CM, Wanner EF, Santos VF, Silva CM (2016) A strategy for clustering students minimizing the number of bus stops for solving the school bus routing problem. In: NOMS 2016-2016 IEEE/IFIP network operations and management symposium, IEEE. pp. 1175–1180
Tutte WT, Tutte WT (2001) Graph theory, vol 21. Cambridge university press, Cambridge
Wang F, Du H, Camacho E, Xu K, Lee W, Shi Y, Shan S (2011) On positive influence dominating sets in social networks. Theor Comput Sci 412:265–269
Zverovich V (2021) Modern applications of graph theory. Oxford University Press, Oxford

Capítulo 9

A novel parallel framework for scatter search

Los detalles del artículo y la revista en la que está publicado son:

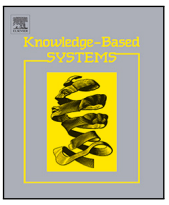
- Casado, A., Pérez-Peló, S., Sánchez-Oro, J., Duarte, A., & Laguna, M. (2025). A novel parallel framework for scatter search. Knowledge-Based Systems, 314, 113248. https://doi.org/10.1016/j.knosys.2025.113248
- Estado: **Publicado**
- Factor de impacto (JCR 2024): 7.600
- Posición relativa: 26/204
- Cuartil: Q1

Knowledge-Based Systems 314 (2025) 113248

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



A novel parallel framework for scatter search

A. Casado [a], S. Pérez-Peló [a,*], J. Sánchez-Oro [a], A. Duarte [a], M. Laguna [b]

[a] Universidad Rey Juan Carlos, C/ Tulipán S/N, Móstoles, 28933, Spain
[b] Leeds School of Business, University of Colorado at Boulder, Boulder, 80309, CO, United States

ARTICLE INFO

Keywords:
Scatter search
Parallel algorithms
Capacitated dispersion problem
MaxCut problem
Profile minimization problem

ABSTRACT

Scatter search (SS) is a well-established metaheuristic for hard combinatorial optimization problems. SS is characterized by its versatility and ease of context adaptation and implementation. Although the literature includes SS parallelization schemes for specific problems, a general parallel framework for scatter search has not been developed and tested. We introduce three SS parallel designs, each focusing on a different task, namely, reducing computational time, increasing search exploration, and balancing search intensification and diversification. The proposed designs are tested on problems where the state of the art is a traditional (sequential) SS approach. This testing platform helps us assess the contributions of the parallel computing strategies to solution speed and quality. Our publicly available code is designed to be adapted to optimization problems that are not considered here. The results show promising avenues for establishing a general framework of SS parallelization.

1. Introduction

Scatter search (SS) is a population-based metaheuristic originally proposed by Glover [1], and various articles have appeared in the literature, such as the one by Glover et al. [2], that provide structured descriptions of the methodology. SS is based on the idea that systematic designs and strategies for exploring the search space usually lead to better results than procedures that rely heavily on randomization. Traditional evolutionary algorithms are designed to operate on a single population of solutions. The creation and evolution of this population depend on rules and operations that involve a fair amount of probabilistic elements. SS philosophy, on the other hand, is to create a large population of solutions with a process that emphasizes diversification. The main SS loop then operates on a small subset of solutions that are carefully selected among the best and most diverse ones, in order to balance search diversification and intensification. SS has been successfully applied to a wide variety of optimization problems [3–5].

Most metaheuristics are designed to search the solution space through a series of sequential decisions. This occurred because multiprocessor computer hardware was not common at the time metaheuristic search was developed and introduced to the optimization community. Today, however, almost every modern device (computers, mobile phones, car devices, etc.) has more than one processor. Sequential computer programs are not designed to take advantage of multiple processors, leading to suboptimal performance when considering modern hardware. The lack of inherent parallelism results in underutilized resources and wasted potential.

Designing a search procedure that takes advantage of parallelism is not trivial. Managing synchronization and communication between multiple processes in a multiprocessor environment can be complex and error prone, potentially leading to race conditions, deadlocks, and increased debugging and maintenance efforts. Furthermore, the overhead associated with splitting tasks among processors and coordinating their execution can sometimes outweigh the benefits of parallelism, especially for small or simple computer programs. With this in mind, advances in metaheuristic technology have produced parallel designs to reduce computational effort or increase the exploration of the solution space [6–8]. The ultimate goal of a solution methodology is to provide a general framework for implementation that can be adapted to specific situations. This is the main motivation of this work in the context of scatter search.

The structure of scatter search makes this methodology suitable for parallelization, and some recent work has shown promising results [9–11]. Attempts have been made to create general SS designs for specific problem classes, for instance, multiobjective optimization [12]. Several parallel versions of scatter search have been proposed in the literature for specific optimization problems [13]. However, to the best of our

* Corresponding author.
E-mail addresses: alejandra.casado@urjc.es (A. Casado), sergio.perez.pelo@urjc.es (S. Pérez-Peló), jesus.sanchezoro@urjc.es (J. Sánchez-Oro), abraham.duarte@urjc.es (A. Duarte), laguna@colorado.edu (M. Laguna).
URLs: https://alejandracasado.github.io/ (A. Casado), https://sergiop3rez.github.io/ (S. Pérez-Peló), https://jesussanchezoro.github.io/ (J. Sánchez-Oro), https://gestion2.urjc.es/pdi/ver/abraham.duarte (A. Duarte), https://www.colorado.edu/business/leeds-directory/faculty/manuel-laguna (M. Laguna).

https://doi.org/10.1016/j.knosys.2025.113248
Received 21 December 2024; Received in revised form 19 February 2025; Accepted 21 February 2025
Available online 3 March 2025

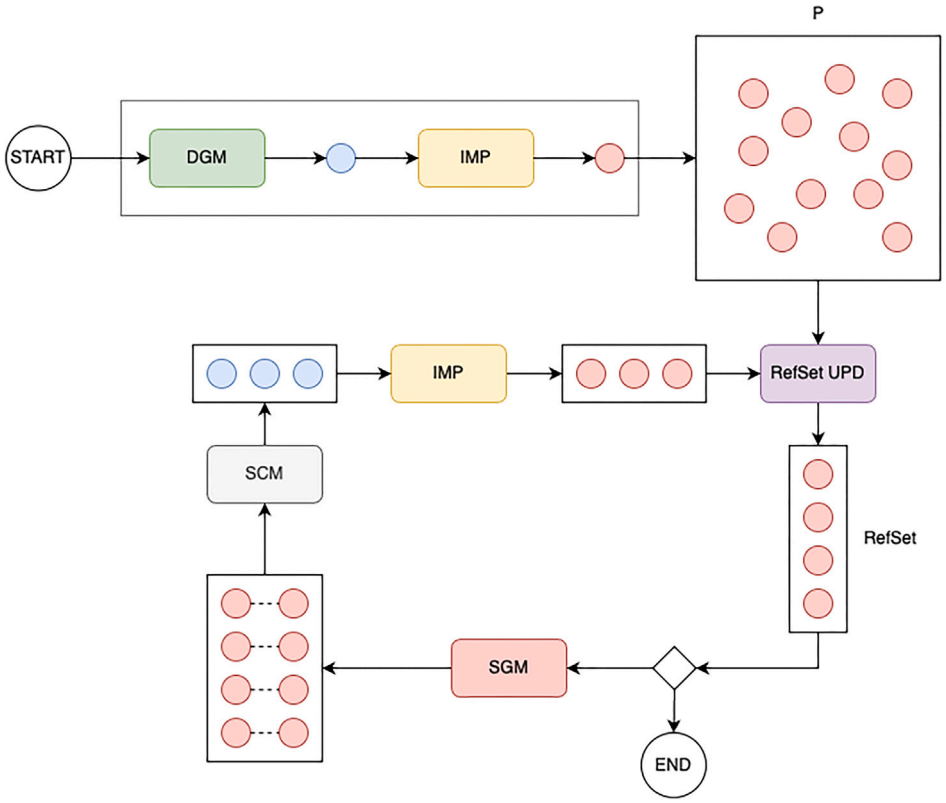


Fig. 1. Basic scheme of scatter search metaheuristic.

knowledge, no attempt has been made to suggest a set of guidelines to implement a parallel scatter search from a general perspective, without focusing on a single problem. Therefore, the main objective of our research is to establish a common framework and guidelines for parallel implementations of scatter search.

Parallelization may focus on specific goals, such as reducing computational effort or increasing the search space explored. The former distributes the computational tasks among processors with the aim of minimizing the time needed to achieve the same results as a sequential design. The latter uses the processors to explore additional regions in an attempt to find a better solution in the same amount of time allotted to a sequential procedure. We refer the reader to some recent works on parallel algorithms that propose novel approaches for solving hard combinatorial optimization problems, e.g., a local search designed to be implemented in CUDA for the set-union knapsack problem [14] and a parallel binary arithmetic optimization algorithm for feature selection [15]. We develop three parallel versions of SS that focus on one or both of the aforementioned parallel design goals.

This research focuses on proposing three different parallel designs for scatter search for combinatorial optimization problems. It is important to note that the objective is not to reach a new state of the art for any of the problems under consideration, but to provide the research community with a general structure for parallelizing scatter search, a framework that can be easily adapted to any combinatorial optimization problem.

2. Sequential scatter search

Before introducing parallel designs for SS, we provide a short description of the traditional sequential SS scheme. SS consists of five methods to construct and maintain a so-called reference set (*RefSet*) of solutions. Fig. 1 is a graphical representation of the solution procedure.

SS constructs an initial population P that must include a combination of high-quality and diverse solutions. To that end, the Diversification Generation Method (DGM) constructs diverse solutions that serve as the starting point for the Improvement Method (IMP) in the first phase of the procedure. The resulting solutions are then added to the initial population P. These two methods are repeated until a predefined number of initial solutions is reached. Once P is created, the Reference Set Update (*RefSet* UPD) method generates a subset of b solutions (the *RefSet*) that is maintained throughout the search. The first *RefSet* is constructed with the $b/2$ best solutions in P and the $b/2$ most diverse solutions with respect to those already in *RefSet*. A usual stopping criterion for SS is the absence of new solutions in the *RefSet* after an iteration of the main loop. At each iteration, the *RefSet* is the input to the Subset Generation Method (SGM), which generates the subset of solutions (e.g., pairs) to be combined. Basic implementations of SGM generate all pairs of solutions in *RefSet*. However, problem-specific characteristics may lead to the generation of subsets of higher order or specific rules for pairing reference solutions. Once the solution subsets are generated, the Solution Combination Method (SCM) creates new trial solutions as combination of reference solutions. The new solutions are subjected to the Improvement Method (IMP) and the Reference Set Update Method updated the *RefSet*. One typical updating rule is to add a trial solution to the *RefSet* if and only if it is better than the worse reference solution in the *RefSet*. In order to maintain diversity, the new trial solution replaces the most similar solution among those in *RefSet* of lower quality. The main SS loop stops when no new solutions enter the *RefSet*. At this point the entire search may end or the *RefSet* could be rebuilt by keeping the best solution found so far and adding $b-1$ solutions to the *RefSet* in the same way as in the initial phase. The search would then stop after a specified number of rebuilding steps.

Out of the five SS methods, there are two, namely *RefSet* UPD and SGM, that have standard implementations that can be directly applied to any combinatorial optimization problem [16]. On the other hand, DGM, IMP, and SCM are typically problem dependent and must be specifically designed for the context under consideration.

3. Parallel SS designs

Since our goal is to propose a general framework for parallelizing SS, we present the designs at a level of abstraction that make them applicable to a wide range of combinatorial optimization problem. Section 3.1 briefly describes the parallel techniques used here and, then, Sections Section 3.2, 3.3, and 3.4 present the parallel designs that are specific to scatter search.

3.1. Parallelization techniques

First, it is important to differentiate between parallelism and concurrency. The former refers to the simultaneous execution of multiple tasks, while the latter refers to the concept of making progress on multiple tasks at the same time, without necessarily requiring them to execute simultaneously. The design of a parallel algorithm aims to maximize parallelism, but it is not always possible due to hardware constraints.

When designing a parallel algorithm [17], a *thread* is the fundamental mechanism for achieving parallelism. A thread is a lightweight, independent unit of execution within a process that enables multiple tasks to run concurrently on a multi-core processor or in a multi-processor environment. Using multiple threads allows an algorithm to continue executing even when some of its phases are time-consuming or blocked. Furthermore, threads enable parallelism by dividing an algorithm into smaller units of work that can be executed concurrently, which may speed up the execution in a multi-processor environment.

Memory models in parallelism play a critical role in determining how data is shared, accessed, and synchronized among multiple threads or processes in a parallel computing environment. These models define the rules and guarantees governing memory consistency and visibility. We use a shared-memory model [18] where all the threads share the same memory space. In this model, we are responsible of determining how updates made by one thread become visible to other threads, ensuring correctness and avoiding data races.

With the aim of providing a general design for parallel algorithms, we use threads within a shared-memory model, for designing parallel strategies in the context of scatter search. Therefore, we do not link our work to a specific technology, allowing the reader to adapt the designs to their preferred programming language and parallelization tool. Specifically, for each proposed algorithmic strategy, we assume that a pool of threads is available, each one being able to execute any of the tasks under consideration. The number of available threads depends only on the hardware specification, making the proposed parallel designs scalable. If there are more tasks than available threads, then some tasks must wait until a thread becomes available to continue the process.

We point out that, in the proposed designs, the time reduction will not be directly proportional to the number of available threads. This is mainly because the thread creation and management, and the context switching, have an associated overhead which must be taken into account when designing parallel algorithms.

3.2. Hoarding parallel scatter search

Our first parallel approach, named Hoarding Parallel Scatter Search (HPSS), aims at reducing the elapsed time to execute SS by parallelizing those phases that can be simultaneously executed without communication. We analyze the structure of SS to identify such phases.

The generation of the initial population P, consisting of $|P|$ independent applications of DGM followed by IMP, requires no sharing of data or information. Since IMP is applied to solutions generated by DGM, IMP depends on DGM and these methods cannot be executed simultaneously during the generation of P. The HPSS parallelization of the generation of the initial population consists of distributing the construction of each solution among the available threads. With this approach, it is expected that the time required to generate the initial population would be reduced by a factor close to T (depending on the associated overhead), where T is the number of available threads.

Once the population has been generated, the next step consists of building the *RefSet*, with a mixture of high-quality and most diverse solutions. This step is not suitable for parallelization, since the selection of the next solution depends on the solutions already in *RefSet*. In particular, when selecting the next high-quality solution, it is necessary to check which solutions are already in the *RefSet* to avoid repetition, while the diversity measure is usually performed with a distance metric between the candidate solution and the ones already in the *RefSet*. Therefore, the logic used for selection does not allow the simultaneous addition of several solutions to the *RefSet*. In the *RefSet* UPD phase, the only part that can be parallelized is the evaluation of the distance from each candidate solution to the *RefSet* under construction. However, this is usually a fast process and the overhead associated with the thread management would be more time consuming than the process itself. Therefore, it is not recommended to parallelize this step unless the calculation of diversity is a rather computationally demanding evaluation.

Next, the tasks associated with SGM are relatively simple, consisting of generating all pairs of solutions. The time required for this step is usually negligible, so it is not a good candidate for parallelization.

The solution combination method is one of the critical elements of SS. In a traditional implementation, SCM consists of combining all pairs of solutions, with path relinking, a computationally demanding procedure, being one of the most common combination approaches. Fortunately, each combination is independent from the others and it is not necessary to perform any synchronization among them. Therefore, SCM is an excellent candidate for parallelization. The same rationale applies to the improvement method. The application of IMP to each trial solution resulting from SCM can be done independently from each other on an available thread.

SS requires that each solution resulting from SCM be subjected to improvement by IMP. Therefore, if SCM is parallelized independently from IMP, then IMP would have to wait until all solutions have been combined to start its parallelization. In order to avoid unnecessary waiting times, the parallelization of both phases is performed jointly. Specifically, each combination is launched in an independent thread and, every time a new trial solution is created out a combination of reference solutions, the trial solution is subjected to the improvement method in a new thread. In other words, combination and improvement are executed concurrently, minimizing the elapsed time to generate new solutions.

Finally, each improved trial solution is evaluated as a candidate for entering in the *RefSet*. This evaluation cannot be performed in parallel, since the acceptance criterion depends on the current composition of the *RefSet*. Therefore, this step is done sequentially and the improved solutions are enqueued until the previous candidates have been evaluated. Fig. 2 shows the scheme of HPSS based on the original SS scheme depicted in Fig. 1.

The tasks that are executed in parallel are depicted over a cross hatched background. HPSS starts by launching a new task for each solution, where each task will execute an independent DGM and IMP step, generating a new solution for the initial population. Once the initial population is created, the *RefSet* is built following a sequential approach. Then, while new solutions are included in the *RefSet*, the SGM sequentially generates all the pairs of solutions to be combined. Each subset (typically a pair) of solutions is combined with the SCM in parallel, and for each solution resulting from the combination, a new task is launched to improve it. Finally, all the improved solutions are enqueued to be sequentially evaluated as candidates to enter the *RefSet*.

3.3. Agglomerative parallel scatter search

Our second parallel designed, named Agglomerative Parallel Scatter Search (APSS), focuses on exploring a wider region of the search space

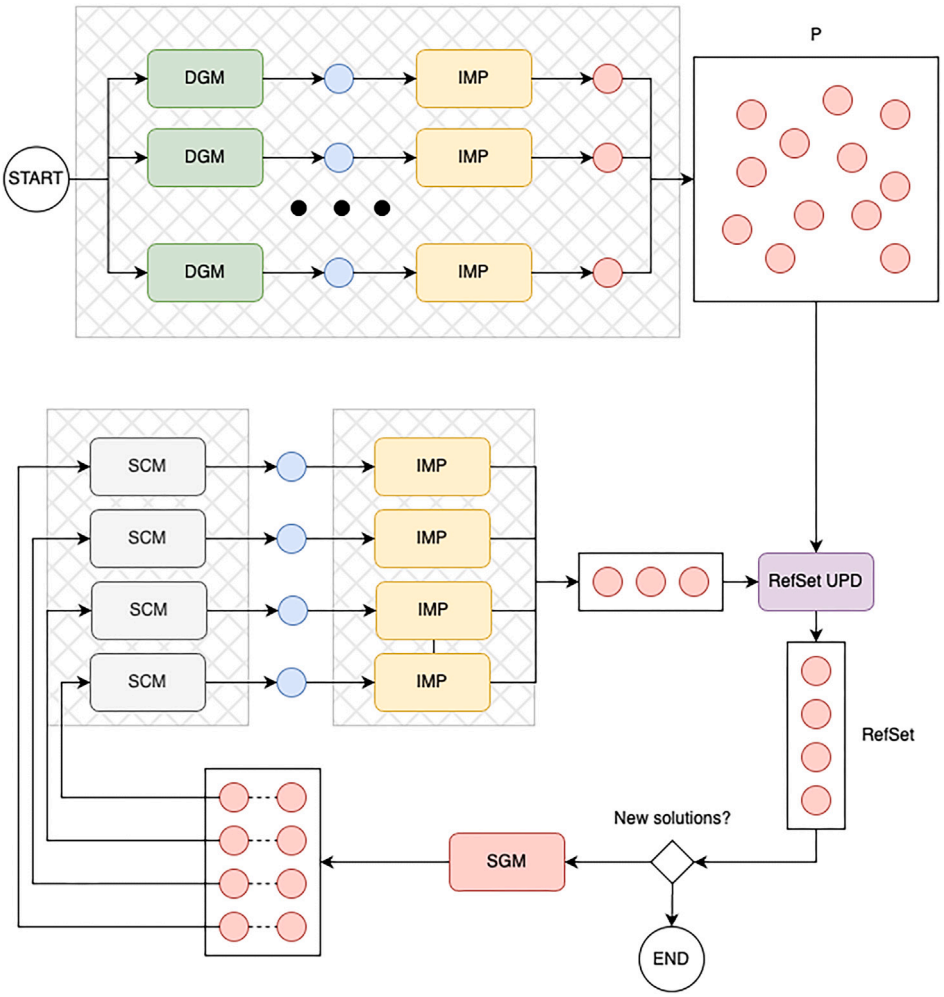


Fig. 2. Scheme of hoarding parallel scatter search.

than its sequential counterpart. Most published SS research proposes several constructive procedures, improvement methods, and even combination mechanisms. Researchers employ scientific experimentation to select the best constructive procedure, the best improvement method, and the best combination mechanism based on empirical evidence. The notion of "best" is associated with overall or average performance. This means that when considering individual outcomes, mechanisms that are not the "best" overall may perform better in some instances.

The hypothesis behind the design of APSS is the following: does the quality of the SS algorithm improve if all the proposed constructive procedures, improvement methods, and combination mechanisms are included in the final version of the algorithm? In a sequential approach, it is not feasible to validate or reject this hypothesis, since the computational effort to evaluate all possible combinations is not affordable to produce results in reasonable computing times. However, the parallel capabilities of computers allows us to include all those elements in the same SS implementation, obtaining the advantages of each constructive, improvement, and combination method originally proposed. Obviously, this variant will not reduce the computing time and, even in some cases, it can be slightly increased, but APSS will allow the search to explore new regions of the search space that could eventually lead to finding better solutions. The reason the computer time could increase with APSS when compared to its sequential counterpart is simple. When selecting the best DGM, IMP, and SCM, researchers usually consider both solution quality and computing time. Therefore, an alternative may be discarded in the sequential version due to its high computational time requirement. However, computationally-expensive alternatives may be included in APSS, leading to an increase in the elapsed time required to complete the search. Fig. 3 shows the APSS scheme.

Without loss of generality, it is assumed that there are C procedures for generating initial solutions, $DGM_1 \dots DGM_C$, I improvement methods $IMP_1 \dots IMP_I$, and S combination mechanisms, $SCM_1 \dots SCM_S$. In the sequential SS, each solution of the initial population is generated by constructing a solution with the selected DGM and then improved by IMP. Instead of considering a single DGM and IMP, APSS leverages parallelism to generate a solution with each available DGM and then improve each generated solution with all IMPs. Therefore, in each iteration, instead of generating a single solution, a total of $C \times I$ solutions are constructed and added to the initial population. It is expected that due to parallelization, the computing time to generate a single solution in SS is similar to the time required by APSS to generate $C \times I$ solutions. As before, the *RefSet* is created sequentially and the SGM generates all pair of solutions as it is customary in SS.

Continuing with the idea of applying several alternative strategies, APSS includes all combinations of SCMs and IMPs to generate new trial solutions. In particular, each subset (usually a pair) of solutions is combined with every proposed combination method and subjected to every proposed improvement method. Therefore, APSS generates $S \times I$ new trial solutions that become candidates for entering the *RefSet*. Since the *RefSet* UPD must be performed sequentially, all the generated solutions are enqueued to be evaluated sequentially.

Since the combination and improvement methods are performed in parallel, the elapsed time is expected to be similar to the one of the

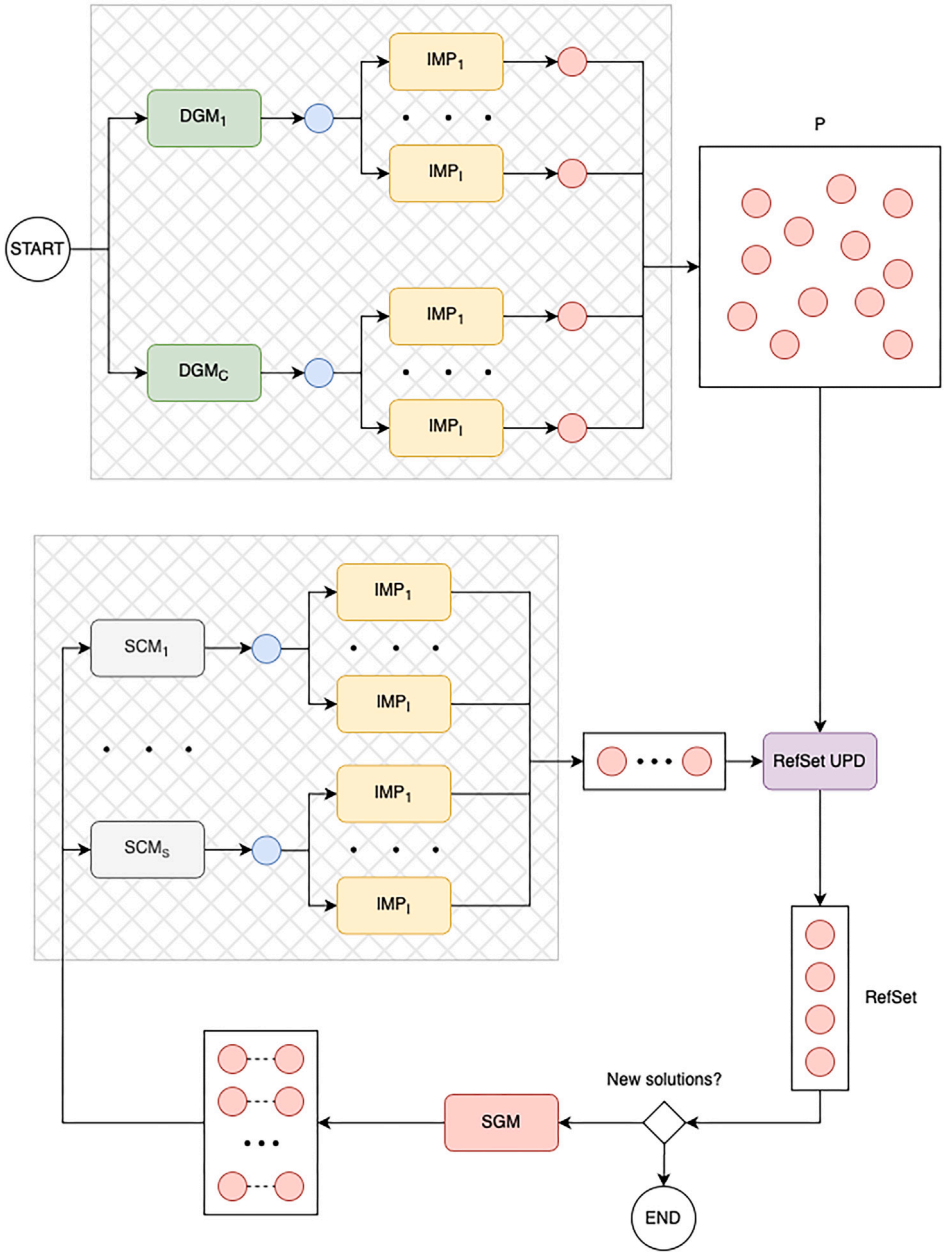


Fig. 3. Scheme of agglomerative parallel scatter search.

sequential version. However, the number of solutions enqueued within APSS will be considerably larger, so the total computing time of the algorithm could be larger than the sequential SS. APSS stops when no new solutions are included in the *RefSet*, as in SS and HPSS.

3.4. Cooperative parallel scatter search

Our third approach is named Cooperative Parallel Scatter Search (CPSS). This variant is a compromise between the reduction of time of HPSS and the increase in the search space exploration of APSS. The idea is to make the components of the algorithm cooperate in order to generate high-quality and diverse solutions.

CPSS is based on APSS, where an intermediate step called *Select Best and Diverse* (SBD) is added. Given a set of solutions, SBD selects the best solution according to the objective function value and the most diverse. Diversity is measured as the minimum distance between a solution and all other solutions in a set. The most diverse solution is the one whose closest solution has the maximum distance. Let S be a set of solutions generated either DGM, IMP, or SCM. The best solution $s^{best} \in S$ in a minimization problem is such that:

$$s^{best} = \arg\min_{s \in S} f(s)$$

and the most diverse solution $s^{diverse} \in S$ is such that:

$$s^{diverse} = \arg\max_{s \in S} \min_{r \in S} d(s, r)$$

where $f(s)$ is the objective function value of solution s and $d(s, r)$ is a metric of distance between solution s and r that depends on the problem context and solution representation. For instance, d can be the Euclidean distance for solutions represented by continuous values. Fig. 4 shows a graphical representation of CPSS.

The algorithm starts with the parallel generation of a set of initial solutions applying all the proposed DGMs. Then, this set is used as input to SBD, obtaining the best and the most diverse solution, i.e., s^{best} and $s^{diverse}$. These two solutions are used as input to every improvement method available in parallel, generating a set of improved solutions.

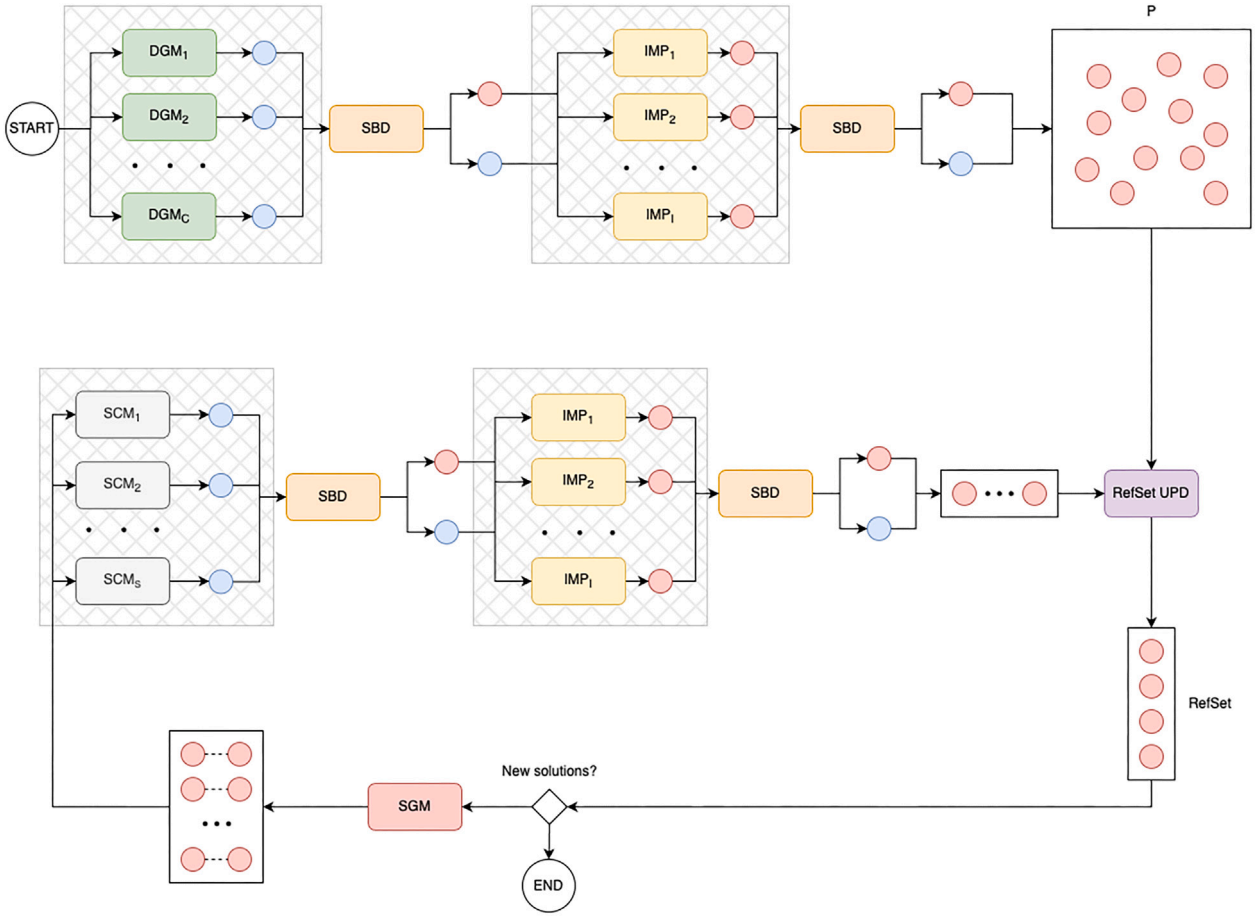


Fig. 4. Scheme of cooperative parallel scatter search.

This set is used as input to SBD, which again produces the best and the most diverse solutions to be included in the initial population.

We point out that CPSS does not apply IMP to all solutions generated by DGM. Instead IMP is only applied to s^{best} and $s^{diverse}$ which are the output of SBD. Therefore, the computational effort is reduced from APSS while keeping the philosophy of balancing intensification and diversification.

The tandem SCM and IMP within CPSS mimics the construction of P. That is, SCM generates a set of trial solutions S that are then the input of SBD. The output of SBD (i.e., the best and most diverse solutions in S) become the input of all the IMPs. This produces the trial solutions that are candidates to become reference solutions if admitted to *RefSet* by the *RefSet* UPD method.

Similarly to the construction of the initial P, the combination/improvement steps are less computationally demanding than in APSS since CPSS does not require to improve every solution obtained by the SCM with every available improvement method. Additionally, the queue of solutions to be evaluated in *RefSet* UPD is smaller than in APSS since the output of this step consists of two solutions only. CPSS retains all the proposed combination and improvement methods but are applied to fewer solutions than APSS. The goal is for CPSS to produce similar quality as APSS with less computational effort. As in the other variants, the algorithm ends when no new solutions enter the *RefSet*.

4. Benchmark problems

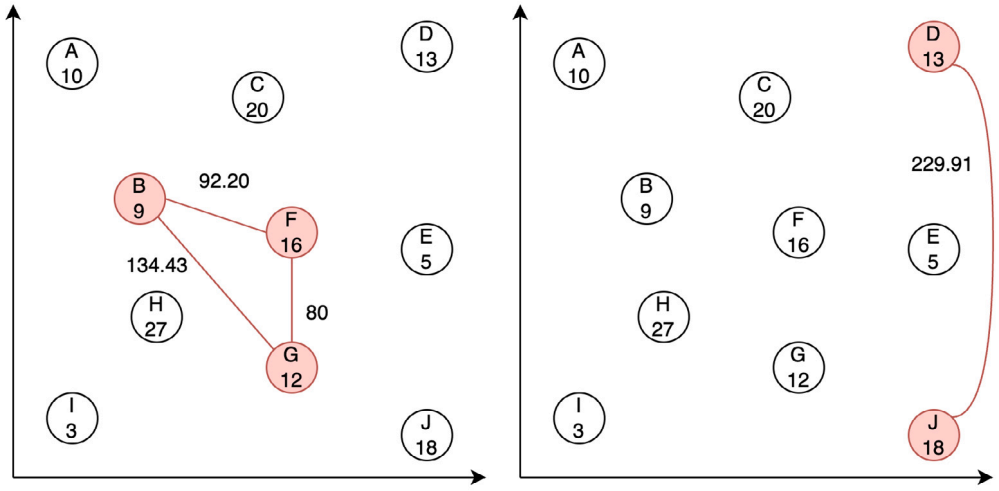
We test the proposed designs on three different optimization problems for which a sequential scatter search exists and whose performance has been published in the literature. For each problem, we provide a brief description and review the DGM, IMP, and SCM of the published sequential SS. We also discuss the parallelization that we have implemented.

4.1. Capacitated dispersion problem

The Capacitated Dispersion Problem (CDP) belongs to the family of dispersion problems that consist of selecting a subset of elements from a given set to maximize the distance among the selected elements. A variant of the CDP considers the number of elements to be selected as fixed, rendering this version unsuitable for several real-life applications. Instead, we consider a version in which a capacity requirement constraint dictates the number elements to be selected. This has direct applications in location problems when establishing the minimum capacity level is required to provide a service.

Given a set of elements I, each element $i \in I$ has an associated capacity c_i, while the total capacity required is represented by B. The CDP consists of selecting a subset $S \subseteq I$ such that $\sum_{i \in S} c_i \geq B$ and the minimum distance between elements in S, i.e., $\min_{i,j \in S} d_{ij}$, is maximized. Fig. 5 shows two possible solutions for an instance with 10 points where each point contains an identification and an associated capacity. The numbers next to the connecting lines are the euclidean distance between the points. We assume a capacity requirements of $B = 30$.

As mentioned above, the number of nodes to be selected is not fixed but determined by the minimum capacity requirement of $B = 30$. Therefore, solution S_1 consisting of elements $\{B, F, G\}$ has a capacity of 37 while S_2 with elements $\{D, J\}$ has a capacity of 31. Note that the number of selected elements is not directly related to the quality of



(a) Solution $S_1 = \{B, F, G\}$ with a capacity of 37. (b) Solution $S_2 = \{D, J\}$ with a capacity of 31.

Fig. 5. Two solutions for a CDP instance with 10 nodes. Numbers in nodes are capacities and numbers next to edges are distances.

the solution, i.e., it is possible to find a solution with a smaller number of elements and better objective function value. In this example, the objective function of S_1 is evaluated as $\min(92.20, 134.43, 80) = 80$, while the objective function of S_2 is evaluated as $\min(229.91) = 229.91$. Therefore, S_2 is better than S_1 in terms of quality, even though it has one fewer element.

Martí et al. [19] developed a sequential SS for the CDP. Their traditional SS implementation includes one DGM, one IMP, and a SCM. The DGM is a semi-greedy constructive method that includes randomization in the selection of elements in order to increase diversity, parameterized by the value of α. In particular, the smaller the value of α, the more random the method becomes. The authors performed a preliminary experimentation in with values of $\alpha = \{RND, 0.25, 0.50, 0.75\}$, where *RND* indicates that the value is selected at random in each construction. Their tuning process yielded RND as the best option. Our parallel design uses four DGMs, one for each α value.

The IMP in [19] is based on selecting one of the critical elements (those with the minimum distance between them), and compute the candidate elements that could replace it, i.e., those which will increase the minimum distance between elements in the solution. With the aim of avoiding marginal improvements, a parameter β is considered, so an element is replaced if the new minimum distance is, at least, $\beta \cdot d_{min}$, where $d_{\min}$ is the current minimum distance between elements. The authors tested $\beta = \{RND, 1, 1.25, 1.5\}$, where *RND* indicates a random value selected between 1 and 1.5. No significant differences in performance were detected among the tested values, with $\beta = 1$ being slightly better. In our parallel design, four IMPs are considered, one for each β value.

The SCM is based on path relinking [20], which generates new solutions by exploring paths that connect two high-quality solutions. Path relinking constructs a path between an initiating solution and a guiding solution with the goal of finding solutions in the path with an improved objective function value.

The path is created by replacing an element from the initiating solution which is not in the guiding solution with another element which is in the guiding solution but not in the initiating solution. The initiating solution eventually becomes the guiding solution, having generated a set of new solutions in the process. Note that intermediate solutions may not satisfy the minimum capacity constraint. In that case, new elements are added until reaching a feasible solution. In this work, a single combination method is proposed, so no multiple versions are considered in the parallel design. Therefore, the parallel SS proposed for this problem considers four DGMs, four IMPs and one SCM.

4.2. Max-cut problem

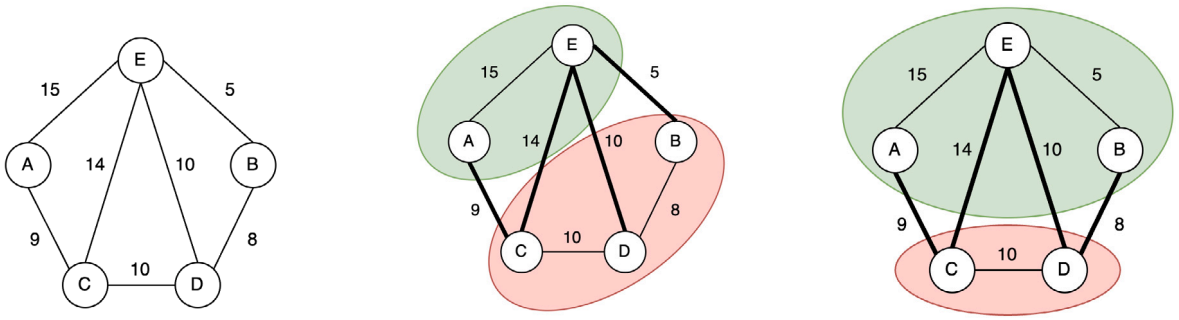
The Max-Cut (MCP) is a classical $\mathcal{NP}$-hard [21] optimization problem that has attracted the attention of the scientific community for decades. Let $G = (V, E)$ be a weighted undirected graph, where V represents the set of vertices, with $|V| = n$, and E represents the set of edges, with $|E| = m$. Each edge $(u, v) \in E$, with $u, v \in V$, has an associated weight w_{uv}. Then, a cut of G, $cut(S, S')$ is defined as a partition of V into two sets S and $S' = V \setminus S$, with its value defined as:

$$cut(S, S') = \sum_{\substack{u \in S \\ v \in S'}} w_{uv}$$

The MCP then consists of finding a cut in G with the maximum value among all possible cuts. Fig. 6 shows a graph with 5 nodes and 7 edges along with two possible solutions for the MCP. The weights are the numbers next to the edges. Solution 1 consists of the cut $S_1 = \{\mathtt{A},\mathtt{E}\}$ and $S_1' = \{\mathtt{B},\mathtt{C},\mathtt{D}\}$, as shown in Fig. 6(b). The edges connecting nodes from different sets are shown with thick lines, i.e., `(A,C)`, `(C,E)`, `(D,E)`, `(B,D)`. The associated cut-value is the sum of the weights of these edges, resulting in $9 + 14 + 10 + 5 = 38$. The second solution is depicted in Fig. 6(c), with the cut $S_2 = \{\mathtt{A},\mathtt{B},\mathtt{E}\}$ and $S_2' = \{\mathtt{C},\mathtt{D}\}$. In this case, the edges connecting the subsets are `(A,C)`, `(C,E)`, `(D,E)`, `(B,D)`, with a cut-value of $9+14+10+8 = 41$. Since the MCP is a maximization problem, solution 2 emerges as a better solution than solution 1 in this example graph.

The best scatter search for the MCP in the optimization literature is Martí et al. [22]. The proposed SS follows the standard framework with population rebuilding. When the main loop of SS finishes, the rebuilding step removes all but the best reference solution and restarts the search by rebuilding the *RefSet*. The stopping criterion is a predefined number of rebuilding steps. The article describes 2 DGMs, 2 IMPs, and 3 SCMs, which we test within or parallel designs.

The first DGM is a GRASP-based constructive procedure in which a solution is generated from scratch by iteratively adding one of the most promising elements to the solution, alternating S and S' as destination set in each iteration. Instead of selecting the best element in each iteration, the method is slightly randomized to increase the diversity of the search. The greedy function used to evaluate the candidate elements to be included in the solution is the sum of the weights of edges that connect the candidate node with those in the opposite set of the solution. The second method initially considers a solution where all the nodes are included in S', and iteratively moves from S' to S the node which produces the largest increment in the cut-value of the solution.



(a) Example graph with 5 nodes and 7 edges.

(b) Solution 1, with $S_1 = \{\mathtt{A},\mathtt{E}\}$ and $S'_1 = \{\mathtt{B},\mathtt{C},\mathtt{D}\}$.

(c) Solution 2, with $S_2 = \{\mathtt{A},\mathtt{B},\mathtt{E}\}$ and $S'_2 = \{\mathtt{C},\mathtt{D}\}$.

Fig. 6. Example graph and two possible solutions to the MCP.

The first IMP is based on transferring nodes from S to S' and vice versa. The method calculates the value associated with each move and selects the one that leads to the largest improvement in the objective function value, following a best improvement strategy. The second IMP employs ejection chain moves. This special type of move depends on a maximum length parameter and it starts by moving a node from its current set to the other one. If the move results in an improvement, it is performed and the chain stops. Otherwise, a node adjacent to the one that moved is selected to be transferred in the opposite direction. If the move of both nodes results in an improvement, the chain stops at depth 2 (which is equivalent to a swap move). Otherwise, the chain of moves continues until it improves the current objective function value or a maximum chain length is reached.

The first SCM calculates a score for each element by taking into account the objective function value of the two solutions being combined. The node score is translated into a probability of being assigned to S or S'. This probabilistic assignment is carried out until all elements are assigned and a new solution is generated. The second SCM constructs a partial solution resulting from the intersection of the two solutions being combined. Then, each unassigned node is included in the set that produces the largest increase in the objective function value. A second solution is generated by randomly assigning the unassigned nodes to either S or S', favoring diversity. Therefore, this second SCM generates two trial solutions instead of only one. The third SCM uses path relinking to transform an initiating solution by moving each node to the set to which it belongs in the guiding solution, generating a path of intermediate feasible solutions.

4.3. Profile minimization problem

The Profile Minimization Problem (PMP) was originally proposed by Tewarson [23] as a mechanism to reduce the required space for storing sparse square matrices, but it has been applied to several problems in areas such as biology [24,25], archeology [26], and manufacturing [27]. The problem is based on the so-called *bandwidth* of a row in a matrix. The *profile* of a matrix is calculated as the sum of the bandwidth values and the goal is to arrange the columns of the matrix in such a way that the profile is minimized. The problem can be viewed as a graph, where the nodes are the columns and the edges indicate whether a nonzero element exist in the corresponding cell. Since the idea is to find an optimal arrangement of columns, a solution can be represented as a permutation. Fig. 7 shows two solutions for a 6×6 matrix with 7 nonzero entries, namely, (A,B), (A,D), (B,C), (B,D), (C,D), (D,E), and (E,F).

The objective is to minimize the sum of the distance (i.e., the profile) between connected nodes. The problem can be represented on a graph $G = (V, E)$, where V is the set of vertices (with $|V| = n$) and E is the set of edges (with $|E| = m$). To evaluate the quality of a solution for the PMP, the profile of each vertex must be calculated. Let π be an ordering of the vertices in V, then the profile of vertex v is given by:

$$\delta_v = \pi(v) - f_v$$

where $\pi(v)$ represents the position of vertex v in the ordering π and f_v is the position of the left-most vertex adjacent to v. If for a given ordering π there is no vertex $u : (u, v) \in E$ to the left of v, then $f_v = \pi(v)$ and therefore $\delta_v = 0$. The profile of graph G is the sum of the profiles of all its vertices:

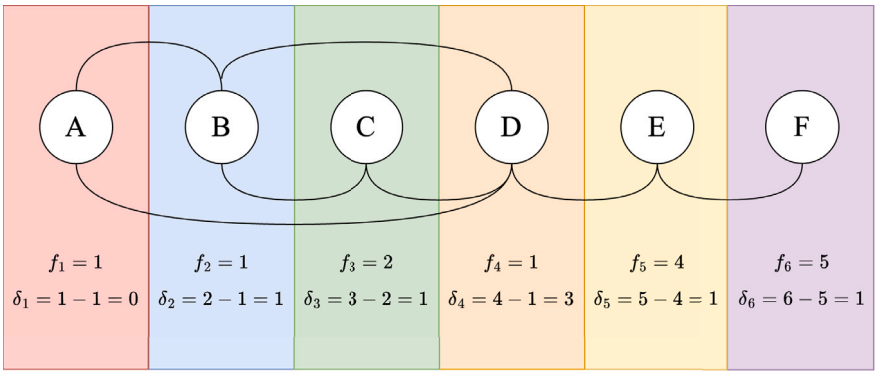
$$Pr(\pi) = \sum_{v=1}^{n} \delta_v = \sum_{v=1}^{n} \pi(v) - f_v$$

An optimal solution for the PMP is an ordering π^* that results in the smallest profile value, that is, $Pr(\pi^*) = min_{\pi \in \Pi} \sum_{v=1}^{n} \delta_v = \sum_{v=1}^{n} \sigma(v) - f_v$, where Π is the solution space consisting of all orderings. The vertices in the solution shown in Fig. 7(a) are ordered alphabetically, with the corresponding positions given by $\pi_1 = (1, 2, 3, 4, 5, 6)$. The positions of the vertices in the solution of Fig. 7(b) are given by $\pi_2 = (4, 2, 5, 3, 6, 1)$.

The best scatter search in literature for the PMP is the one proposed by Sánchez-Oro et al. [28]. The SS implementation in [28] is standard with four DGMs, two IMPs and two SCMs. The DGMs are GRASP-based constructive procedures with two different greedy functions. The first greedy function assigns an urgency score to unassigned vertices in G. At every step of the construction process, the urgency of assigning a position to vertex v is the difference between the number of vertices adjacent to v that have already been assigned a position minus the number of vertices adjacent to v that have not yet been assigned a position. The second greedy function modifies the first one by multiplying the sum of the absolute differences between the candidate position for v and the positions of all its adjacent vertices that have already been included in the solution. Four DGMs were configured using the two greedy functions combined with two different pseudo-greedy ways of selecting the next vertex to be assigned a position.

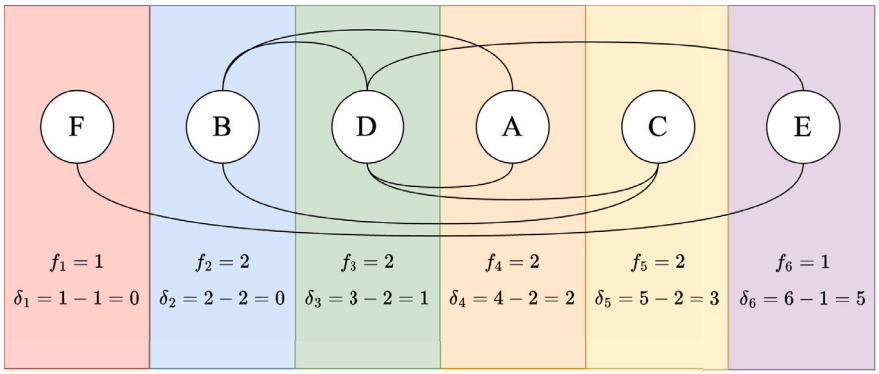
For the improvement methods, two neighborhoods were defined. The first performs $swap(u, v)$ moves of the position of vertices u and v, that is, $\pi(v) \leftrightarrow \pi(u)$. The second is an $insert(u, \pi(v))$ that moves vertex u to the position occupied by vertex v, causing vertex v and all other vertices at higher positions to shift.

The two solution combination methods are based on path relinking, where the transformation process is done with swap moves. At each step, all swaps that move the current solution (starting from the initiating solution) closer to the guiding solution are evaluated. The first SCM selects the best move at each step and the second selects the move at random. This is done until the guiding solution is reached.



$$Pr(\pi_1) = 0 + 1 + 1 + 3 + 1 + 1 = 7$$

(a) Solution $\pi_1 = (1, 2, 3, 4, 5, 6)$ with a profile value of 7.



$$Pr(\pi_2) = 0 + 0 + 1 + 2 + 3 + 5 = 11$$

(b) Solution $\pi_2 = (4, 2, 5, 3, 6, 1)$ with a profile value of 11.

Fig. 7. Two solutions for an instance with $n = 6$ and $m = 7$. Vertices are identified by letters.

5. Computational results

The goal of the computational experiments is to provide a direct performance comparison between the original sequential designs with the proposed parallel versions. In order to make a fair comparison, we implemented all procedures in Java 21 and the experiments were conducted on an AMD Ryzen 9 5950x with 32 threads and 128 GB RAM.

We point out that a single run of a parallel design may be difficult to replicate because different interleaving of threads during execution may lead to slightly different values of the key metrics. To make the comparison as robust as possible, we executed all variants 10 times. For each experiment, we report the following metrics:

- Best, the best objective function value for all runs
- Avg., the average objective function value
- Std. Dev., the standard deviation of the objective function value
- Avg. Time (s), the average time, in seconds, required by a run
- #Best, the number of times that the procedure reaches the best solution
- Dev(%), the average deviation with respect to the best solution

The last two metrics are computed with respect to the best solution found at the end of the 10 runs. Our tables summarize the results across all instances. We did not fine tuned the procedures because we assumed that this was done in the original work and the best parameter values are the ones reported in the corresponding published articles. Therefore, the specific parameter values used for each algorithm are exactly the same as the ones indicated in the original articles. In order to ease further comparisons, the source code of all the considered algorithms and the instances used is publicly available at https://grafo.etsii.urjc.es/ParallelScatterSearch.

For the sake of clarity, in all the experiments the best value for each metric is highlighted in bold font. Additionally, the acronyms used for each variant in the tables are: SSS, Sequential Scatter Search; HPSS, Hoarding Parallel Scatter Search; APSS, Agglomerative Parallel Scatter Search; and CPSS, Cooperative Parallel Scatter Search.

Finally, a speedup graph is shown for each analyzed problem. In particular, HPSS is the only variant designed to reduce the computing time instead of increasing the exploration of the search space, so the speedup can only be measured when comparing the sequential version with HPSS. The speedup is evaluated as $Speedup = t_1/t_p$ where t_1 is the time required by the sequential version and t_p is the time required when considering p processors in HPSS. For the sake of simplicity, a set of 5 representative instances has been selected for each problem to evaluate the speedup. The number of processors evaluated is $p = \{1, 2, 4, 8, 16\}$, and, in each speedup graph, the x-axis represents the number of processors and the y-axis represents the speedup obtained. Each instance is represented in a different line.

5.1. Capacitated dispersion problem

For this experiment, we use the instances with 50, 150, and 500 elements introduced by Martí et al. [19]. We divide the results in three tables, one for each number of elements. Table 1 shows the results for instances with 50 elements. The instances with 50 elements are too small to notice any significant performance difference among the procedures. Clearly, they all perform at the same level. The parallelization of SS is not needed to tackle CDP instances of this size.

Table 1
Summary of results for instances with 50 elements for the CDP.

$n = 50$	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	105.05	104.83	0.00069	≤ 0.01	18	0.07
HPSS	104.94	104.75	**0.00058**	≤ 0.01	16	0.27
APSS	**105.08**	**105.03**	0.00105	0.02	**19**	**0.03**
CPSS	**105.08**	104.98	0.00076	0.02	**19**	0.06

Table 2
Summary of results for instances with 150 elements for the CDP.

$n = 150$	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	112.80	112.18	0.00294	0.10	7	0.30
HPSS	112.73	112.12	**0.00087**	**0.01**	6	0.34
APSS	**113.02**	**112.59**	0.01263	0.24	**16**	**0.10**
CPSS	112.97	112.41	0.00804	0.22	13	0.12

Table 3
Summary of results for instances with 500 elements for the CDP.

$n = 500$	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	21.61	20.69	0.04731	3.89	17	2.70
HPSS	21.54	20.53	**0.00701**	**0.34**	18	2.76
APSS	**22.19**	**21.12**	0.27494	7.39	**28**	**1.11**
CPSS	22.14	21.03	0.21332	7.10	26	1.13

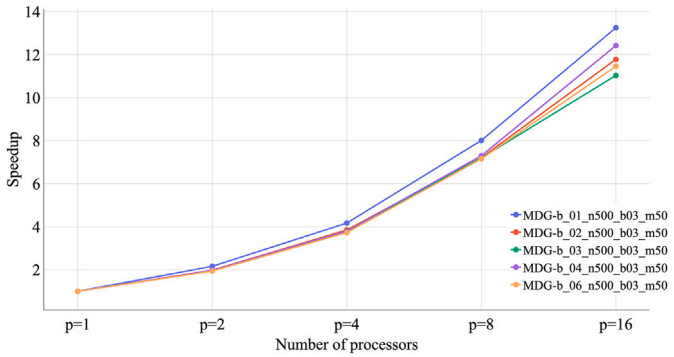


Fig. 8. Speedup obtained for the CDP.

For instances with 150 elements, Table 2 shows that all exhibit low variation across runs. HPSS produces similar results as SSS in terms of solution quality but it does so ten times faster. For these larger instances, APSS and CPSS, which are designed to expand exploration, find significantly more best solutions than the sequential version. Due to the design goals of HPSS (reduce computational time), APSS and CPSS (expand the search), it is expected that as the problem size increases, HPSS will continue to produce solutions of similar quality as SS but much faster, while APSS and CPSS will improve quality with an increase in the computational effort. The results for instances with 500 elements confirm this trend (see Table 3).

The instances with 500 elements help us evaluate the performance of the parallel versions. HPSS confirms that is able to produce solution quality similar to SSS. However, HPSS is ten times faster than the sequential version. Both APSS and CPSS outperform SSS in terms of solution quality with a modest increase in computational time. Finally, Fig. 8 shows the speedup graph for the CDP.

As it can be seen in the graph, the speedup polynomially grows with the number of processors available. Additionally, it can be seen that the thread management does not incur in a heavy overload affecting the computing time, since the speedup is approximately equivalent to the number of processors. This also indicates that the synchronization of the threads is efficient, without unnecessary stops during the search in order to wait for solution updates. When the number of processors is 16, the speedup is approximately 12 on average, indicating that, when considering a large number of processors, the thread management and synchronization does not allow to obtain directly the speedup of 16.

Table 4
Summary of results for instances from Set 1 consisting of a total of 54 instances for the MCP.

Set 1	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	5121.07	5106.35	30.34	910.60	1	1.03
HPSS	5114.83	5097.15	13.53	**408.59**	1	1.26
APSS	**5149.91**	**5132.47**	17.92	1283.99	**41**	**0.07**
CPSS	5106.52	5089.24	**1.82**	1760.44	16	1.12

5.2. Max-cut problem

In this experiment, we consider two sets of MCP instances derive from the original sequential scatter search [22]:

- Set 1: These 54 instances were constructed in [29] and come from a graph generator with a number of nodes ranging from 800 to 3000. The instances correspond to toroidal, planar, and random graphs with weight values of −1, 0, or 1. This set was later used in [30,31].
- Set 2: This set consists of 30 low-density instances proposed in [31]. The number of nodes ranges from 128 to 2744 and weight values are -1, 0, or 1.

Table 4 shows the results obtained when comparing all the SS variants over the Set 1 of instances.

If we first analyze the HPSS variant, which is focused on reducing the computing time with respect to the original SSS. The results show how HPSS requires approximately half of the computing time to perform the same operations as SSS. As expected, in terms of quality, both algorithms perform in a similar way. The small variations between them can be partially explained due to the non-deterministic nature of the algorithms since both of them include some random choices during the search.

Regarding APSS, it clearly obtains the best results in terms of quality, reaching 41 out of 54 best solutions, with a deviation which is close to zero. The rationale behind this is that this variant considers all the constructive, local search, and combining methods proposed in the original work. The inclusion of all the methods allow the algorithm to perform a deeper search, resulting in better solutions. However, this exhaustive search requires an increase in the computational demand, which cannot be totally overcome by the parallelization of the algorithm. Therefore, the computing time of this variant is considerably larger than SSS, since it is focused on improving the quality, not the computational effort.

Finally, CPSS is able to reach better results than SSS and HPSS but it is slightly worse than APSS in terms of quality. It is expected that CPSS performs better than APSS in terms of computing time but the results are not reflecting this behavior. This is mainly because in the context of MCP, some of the components of the algorithm which were discarded in the SSS are extremely computationally demanding. This penalizes the cooperative version which has to wait to every single component until resorting to the next phase, unlike in APSS which does not need to wait for every component to finish. These results arise an interesting conclusion from the parallel design: there is not a *best* parallel design, and all of them should be carefully studied when dealing with a new problem.

Table 5 shows the results obtained when comparing the SS variants over the Set 2 of instances. Notice that this set is smaller than Set 1 and, therefore, it is expected that the differences among the parallel variants also become smaller.

Again, when comparing HPSS with SSS in terms of computational effort, the parallel version requires less than half of the computing time than the sequential one. Analyzing the quality of both variants, we

Table 5
Summary of results for instances from Set 2 consisting of a total of 30 instances for the MCP.

Set 2	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	1105.00	**1099.67**	**3.32**	656.69	16	0.31
HPSS	1101.47	1094.89	8.97	**270.39**	12	0.60
APSS	**1105.20**	1097.13	4.75	698.40	**20**	**0.11**
CPSS	1081.80	1073.79	13.91	1139.54	13	1.17

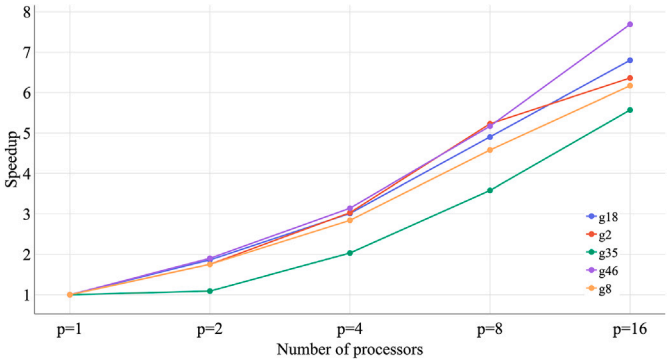


Fig. 9. Speedup obtained for the MCP.

can see that the differences between them are negligible, showing that HPSS is obtaining successful results.

The behavior of APSS and CPSS is similar to the one presented in Set 1. In particular, APSS requires from slightly larger computing times than SSS but it is able to reach 20 out of 30 best solutions, emerging as the best variant in terms of objective function value. In the case of CPSS, it is worth mentioning that, in this particular set of instances, it is not an adequate design, since it requires from large computing times without being able to produce better results than the remaining versions. The speedup graph for the MCP is shown in Fig. 9.

In this case, the speedup obtained is slightly slower than the one obtained in the CDP, reaching a maximum value of 8. This can be partially explained since the complexity of the problem requires from more synchronization, and the algorithm performs a larger number of waits. A special case can be seen in instance g35, where the parallelization is not able to reduce the computing time when considering 2 processors, requiring a minimum number of processors of 4 in order to leverage the potential of HPSS.

5.3. Profile minimization problem

We tested our procedures with one of the instances group employed by Sánchez-Oro et al. [28]. In that work, there are three groups of instances in the test set: D4-trees, K-Graphs and HB (Harwell Boeing Sparse Matrix Collection [32]). In this work, only the HB collection has been used. We have not included D4-trees and K-Graphs since they are easy to solve for the sequential version of SS and, therefore, they would not allow us to analyze the performance of the parallel designs. To help the analysis, the set has been subdivided into four different subsets, taking into account the size of the input graphs. More specifically:

- Instances with $|V| < 100$: 14 instances with number nodes ranging from 24 to 96 and number of edges ranging from 34 to 2145.
- Instances with $100 \leq |V| < 250$: 13 instances with number nodes ranging from 100 to 245 and number of edges ranging from 179 to 1758.
- Instances with $250 \leq |V| < 500$: 17 instances with number nodes ranging from 256 to 494 and number of edges ranging from 586 to 2712.
- Instances with $|V| \geq 500$: 17 instances with number nodes ranging from 503 to 960 and number of edges ranging from 1282 to 3422.

Table 6
Results obtained with the different SS variants for the HB instances with less than 100 nodes for the PMP.

$\|V\| < 100$	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	460.14	462.51	2.24	0.93	10	0.64
HPSS	460.50	465.86	2.88	**0.13**	9	0.27
APSS	**459.21**	**460.87**	**1.11**	2.36	**12**	**0.10**
CPSS	459.57	462.08	1.68	3.54	10	0.15

The results of this experiment summarized in four different tables, one for each group of instances. Table 6 shows the results obtained for the set of HB instances with less than 100 nodes for each SS implementation. It can be seen that the implemented variants are robust with respect to the obtained results for the other two problems. Regarding computational time, HPSS provides solutions with much lower effort and small loss of quality compared to SSS. Standard deviations show that all variants are robust when tackling small instances of similar characteristics. The number of times that the best objective function value of the experiment is found points to APSS as the better approach, reaching the best result in 12 out of 14 instances. This implementation obtains the lowest average objective function value, as well as the lowest deviation percentage when it is not able to match the best objective function value for the instance being solved. In the case of CPSS, it is worth mentioning that, although it is not able to reach a larger number of best solutions than SSS, it considerably reduces the average deviation with respect to the best solution found.

Table 7 shows the results for the second set of instances. This time, the contribution HPSS with respect to the computational time is more evident: it requires an order of magnitude less time than the other implementations. However, as the complexity of the problem to be solved increases, it also becomes more evident that the price of smaller computational time is reflected in the quality of the solutions. Even so, this version of the algorithm obtains a percentage deviation value of less than 1 percent on average when it is not able to reach the best solution for a given instance. This makes the scheme suitable for situations where computational time is a critical feature of the algorithm's application context, without paying too high a toll on the quality of the solutions obtained. Again, the APSS implementation emerges as the best of the experiment. In this case, CPSS is able to outperform the sequential version in both number of best solutions and average deviation, but with a considerably larger computing time.

An analysis of Table 8 provides us with valuable and significant information regarding the contributions of each variant. First, we observe that SSS requires a lot more computational time than HPSS. However, HPSS solution quality is on average inferior to SSS. APSS emerges as the best option for solving PMP. It obtains the largest number of best solutions, with a small deviation when it is not capable of matching the best result. It is interesting to observe that, when the complexity of the instances increases, CPSS improves its solution quality performance when compared to SSS and HPSS.

Table 9 helps us validate our hypothesis regarding the contribution of parallelism in the context of SS. CPSS obtains best results in 7 of the 17 instances. However, APSS is the preferred choice when measuring performance with the average objective function value and the average of the best value found. For these instances, the computational time for HPSS and SSS is of the same order of magnitude. However, the average time required by HPSS is approximately 8 times less than that required by SSS, so it is still an effective alternative when reduced computation times are desired without a significant sacrifice of solution quality. Fig. 10 shows the speedup obtained in this problem.

The speedup obtained in PMP is equivalent to the one obtained in MCP, resulting in slightly larger than $p/2$ on average. Again, the complexity of the problem requires from several synchronization points that blocks the algorithm until a solution is obtained. In this case, all the instances present a similar behavior, showing the robustness of the method.

Table 7
Results obtained with the different SS variants for the HB instances with a number of nodes ranging from 100 to 250 nodes (exclusive) for the PMP.

$100 \leq \|V\| < 250$	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	2132.31	2156.35	15.98	18.71	6	0.31
HPSS	2140.31	2178.86	31.29	**2.75**	3	0.73
APSS	**2124.85**	**2147.50**	**15.25**	46.14	**9**	**0.09**
CPSS	2125.77	2150.49	16.24	66.58	6	0.15

Table 8
Results obtained with the different SS variants for the HB instances with a number of nodes ranging from 250 to 500 nodes (exclusive) for the PMP.

$250 \leq \|V\| < 500$	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	6890.29	7067.24	102.01	117.22	4	1.01
HPSS	6985.41	7164.05	120.79	**16.45**	2	1.65
APSS	**6847.12**	**6978.65**	71.54	322.30	**10**	**0.93**
CPSS	6867.76	6983.64	**71.49**	405.74	6	0.61

Table 9
Results obtained with the different SS variants for the HB instances with a number of nodes greater than 500 for the PMP.

$\|V\| \geq 500$	Best	Avg.	Std. dev.	Avg. time (s)	#Best	Dev(%)
SSS	16 992.88	17 315.11	187.75	854.42	3	0.47
HPSS	17 153.35	17 554.63	292.61	**114.35**	4	1.84
APSS	**16 981.76**	**17 222.32**	**149.95**	1447.62	6	**0.52**
CPSS	16 984.53	17 283.02	205.63	1718.21	**7**	0.57

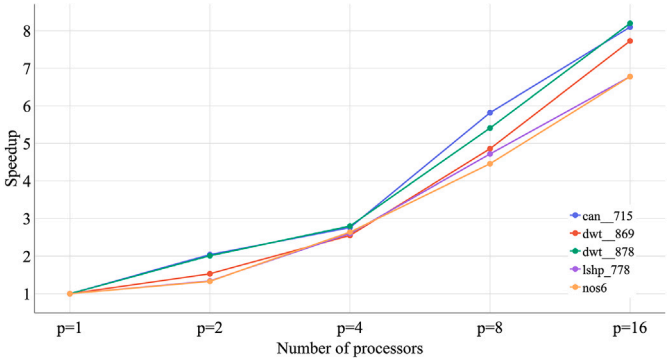


Fig. 10. Speedup obtained for the PMP.

6. Conclusions

We propose three parallel designs for the scatter search methodology, each focusing on a different goal. On one hand, Hoarding Parallel Scatter Search focuses on reducing the computational effort of traditional scatter search without sacrificing solution quality. On the other hand, Agglomerative Parallel Scatter Search aggregates search components to explore a wider portion of the search space with the goal of increasing solution quality at the expense of additional computational effort. Finally, Cooperative Parallel Scatter Search is configured to balance efficiency and efficacy by a collaboration among various search components.

The proposed designs have been tested in three problems with different solution representation, namely, the profile minimization problem, the max-cut problem, and the capacitated dispersion problem. The experimental results show the advantages and disadvantages of each parallel design when dealing with different families of problems of several sizes. In particular, HPSS emerges as a fast alternative when computing time is limited. When computing time is not a limiting factor, both APSS and CPSS show competitive results in terms of quality, outperforming the results of the sequential version. The main difference between them relies on the components available for the search (constructive and local search methods), enabling a wide variety of configurations. Our experiments show that the larger the instance to be solved, the better the performance of the parallel designs. Therefore, problem size can be used to determine whether it is worth the effort of designing and implementing a parallel SS version in practice.

As future work, it would be interesting to establish the effectiveness of parallel designs for other metaheuristic methodologies. The nature of these designs should also be general to provide the research community with a solid base for creating additional problem-specific designs. It would also be interesting to evaluate the performance of proposed parallel strategies in various hardware environments, such as GPU computing.

CRediT authorship contribution statement

A. Casado: Writing – review & editing, Writing – original draft, Validation, Software, Investigation. **S. Pérez-Peló:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation. **J. Sánchez-Oro:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Funding acquisition, Conceptualization. **A. Duarte:** Writing – review & editing, Writing – original draft, Project administration, Investigation, Funding acquisition, Conceptualization. **M. Laguna:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Sergio Perez-Pelo reports financial support was provided by Spain Ministry of Science and Innovation. Jesus Sanchez-Oro reports financial support was provided by Spain Ministry of Science and Innovation. Alejandra Casado reports financial support was provided by Spain Ministry of Science and Innovation. Abraham Duarte reports financial support was provided by Spain Ministry of Science and Innovation. Jesus Sanchez-Oro reports financial support was provided by Spain Ministry for Digital Transformation and the Public Function. Abraham Duarte reports financial support was provided by Ministry for Digital Transformation and the Public Function. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors acknowledge support from the Spanish Ministry of "Ciencia e Innovación", (MCIN/AEI/10.13039/501100011033/FEDER, UE) under grant ref. PID2021-125709OA-C22 and PID2021-125709OB-C21, and "Ministerio para la Transformación Digital y de la Función Pública" (Grant Ref. TSI-100930-2023-0003, AI4DDS: Artificial Intelligence for Data Driven Solutions).

Data availability

We have shared a link with the source code and the dataset used for this research.

References

[1] F. Glover, A template for scatter search and path relinking, in: European Conference on Artificial Evolution, Springer, 1997, pp. 1–51.
[2] F. Glover, M. Laguna, R. Marti, Scatter search and path relinking: Advances and applications, Handb. Metaheuristics (2003) 1–35.
[3] J. Behnamian, S. Memar Dezfooli, H. Asgari, A scatter search algorithm with a novel solution representation for flexible open shop scheduling: a multi-objective optimization, J. Supercomput. 77 (2021) 13115–13138.
[4] J. Sánchez-Oro, M. Laguna, R. Martí, A. Duarte, Scatter search for the bandpass problem, J. Global Optim. 66 (2016) 769–790.
[5] J. Sánchez-Oro, A. Martínez-Gavara, M. Laguna, R. Martí, A. Duarte, Variable neighborhood scatter search for the incremental graph drawing problem, Comput. Optim. Appl. 68 (2017) 775–797.
[6] E. Alba, Parallel Metaheuristics: a New Class of Algorithms, John Wiley & Sons, 2005.
[7] T.G. Crainic, M. Gendreau, P. Hansen, N. Mladenović, Cooperative parallel variable neighborhood search for the p-median, J. Heuristics 10 (2004) 293–314.
[8] F. García-López, B. Melián-Batista, J.A. Moreno-Pérez, J.M. Moreno-Vega, The parallel variable neighborhood search for the p-median problem, J. Heuristics 8 (2002) 375–388.
[9] W. Bożejko, M. Wodecki, Parallel scatter search algorithm for the flow shop sequencing problem, in: Parallel Processing and Applied Mathematics: 7th International Conference, PPAM 2007, Gdansk, Poland, September 9-12, 2007 Revised Selected Papers 7, Springer, 2008, pp. 180–188.
[10] F. Garcıa-López, B. Melián-Batista, J.A. Moreno-Pérez, J.M. Moreno-Vega, Parallelization of the scatter search for the p-median problem, Parallel Comput. 29 (5) (2003) 575–589.
[11] F.G. López, M.G. Torres, B.M. Batista, J.A.M. Pérez, J.M. Moreno-Vega, Solving feature subset selection problem by a parallel scatter search, European J. Oper. Res. 169 (2) (2006) 477–489.
[12] A. López-Sánchez, J. Sánchez-Oro, M. Laguna, A new scatter search design for multiobjective combinatorial optimization with an application to facility location, INFORMS J. Comput. 33 (2) (2021) 629–642.
[13] D.R. Penas, P. González, J.A. Egea, J.R. Banga, R. Doallo, Parallel metaheuristics in computational biology: An asynchronous cooperative enhanced scatter search method, Procedia Comput. Sci. 51 (2015) 630–639.
[14] E. Sonuç, E. Özcan, CUDA-based parallel local search for the set-union knapsack problem, Knowl.-Based Syst. (2024) 112095.
[15] Z. Zhuang, J.-S. Pan, J. Li, S.-C. Chu, Parallel binary arithmetic optimization algorithm and its application for feature selection, Knowl.-Based Syst. 275 (2023) 110640.
[16] M. Laguna, R.C. Martí, Scatter Search: Methodology and Implementations in C, Springer Science & Business Media, 2003.
[17] P. Chaudhuri, Parallel Algorithms: Design and Analysis, Prentice-Hall, Inc. 1992.
[18] S.V. Adve, K. Gharachorloo, Shared memory consistency models: A tutorial, Computer 29 (12) (1996) 66–76.
[19] R. Martí, A. Martínez-Gavara, J. Sánchez-Oro, The capacitated dispersion problem: An optimization model and a memetic algorithm, Memetic Comput. 13 (2021) 131–146.
[20] F. Glover, M. Laguna, R. Martí, Fundamentals of scatter search and path relinking, Control Cybernet. 29 (3) (2000) 653–684.
[21] R.M. Karp, Reducibility Among Combinatorial Problems, Springer, 2010.
[22] R. Martí, A. Duarte, M. Laguna, Advanced scatter search for the max-cut problem, INFORMS J. Comput. 21 (1) (2009) 26–38.
[23] R. Tewarson, Sparse Matrices, 99, Academic Press, 1973, iii–xv, 1–160.
[24] R.M. Karp, Mapping the genome: some combinatorial problems arising in molecular biology, in: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, 1993, pp. 278–285.
[25] F. Alizadeh, R.M. Karp, L.A. Newberg, D.K. Weisser, Physical mapping of chromosomes: A combinatorial problem in molecular biology, Algorithmica 13 (1995) 52–76.
[26] D. Kendall, Incidence matrices, interval graphs and seriation in archeology, Pacific J. Math. 28 (3) (1969) 565–570.
[27] Y.-C. Lai, V.I. Weingarten, H. Eshraghi, Matrix profile and wavefront reduction based on the graph theory and wavefront minimization, Internat. J. Numer. Methods Engrg. 39 (7) (1996) 1137–1159.
[28] J. Sánchez-Oro, M. Laguna, A. Duarte, R. Martí, Scatter search for the profile minimization problem, Networks 65 (1) (2015) 10–21.
[29] C. Helmberg, F. Rendl, A spectral bundle method for semidefinite programming, SIAM J. Optim. 10 (3) (2000) 673–696.
[30] S. Burer, R.D. Monteiro, Y. Zhang, Rank-two relaxation heuristics for max-cut and other binary quadratic programs, SIAM J. Optim. 12 (2) (2002) 503–521.
[31] P. Festa, P.M. Pardalos, M.G. Resende, C.C. Ribeiro, Randomized heuristics for the MAX-CUT problem, Optim. Methods Softw. 17 (6) (2002) 1033–1058.
[32] R. Boisvert, R. Pozo, K. Remington, B. Miller, R. Lipman, Matrix Market, National Institute of Standards and Technology (NIST), Gaithersburg (Maryland), 2004, [January, 2011]. http://math.nist.gov/MatrixMarket.

Parte III

Publicaciones adicionales durante el desarrollo de la tesis

Capítulo 10

Trabajos presentados en congresos nacionales e internacionales

En este capítulo se recogen los trabajos expuestos en congresos nacionales e internacionales. El capítulo cuenta con dos secciones: la primera en la que están enumerados aquellos trabajos asociados a un capítulo de libro de Lecture Notes in Computer Science y en la segunda los que no.

10.1. Trabajos con capítulo de libro de Lecture Notes in Computer Science asociado

1. Casado, A., Bermudo, S., López-Sánchez, A. D., & Sánchez-Oro, J. (2023, Febrero). A Fast Metaheuristic for Finding the Minimum Dominating Set in Graphs. In Metaheuristics International Conference (pp. 554-559). Cham: Springer International Publishing.
2. Casado, A., Mladenović, N., Sánchez-Oro, J., & Duarte, A. (2023, Mayo). A Metaheuristic Approach for Solving Monitor Placement Problem. In International Conference on Variable Neighborhood Search (pp. 1-13). Cham: Springer Nature Switzerland.
3. Casado, A., Sánchez-Oro, J., Martínez-Gavara, A., & Duarte, A. (2024, Junio). Improving Biased Random Key Genetic Algorithm with Variable

Neighborhood Search for the Weighted Total Domination Problem. In Metaheuristics International Conference (pp. 377-382). Cham: Springer Nature Switzerland.

10.2. Trabajos sin capítulo de libro de Lecture Notes in Computer Science asociado

1. Casado-Ceballos, A., Pérez-Peló, S., Sánchez-Oro, J., & Duarte, A. Iterated Greedy para resolver el problema de localización de regeneradores resiliente a fallos. XX Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2024). Organizado por la Asociación Española para la Inteligencia Artificial (AEPIA). A Coruña, España. Del 19 al 21 de junio de 2024. ISBN del Libro de Actas: 978-84-09-62724-0. Páginas: 468-472.
2. Martínez-Gavara, A., Sánchez-Oro, J. & Casado, A. Biased GRASP con VNS para el problema de dominación total de grafos ponderados. XVI Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2025). Del 28 al 30 de mayo de 2025.
3. Casado, A., Pérez-Peló, S., Sánchez-Oro, J., Duarte, A., & Laguna, M. A novel parallel framework for scatter search. European Conference on Operational Research (EURO 2025). Del 22 al 25 de junio de 2025.
4. Casado, A., Pérez-Peló, S., Sánchez-Oro, J., Duarte, A., & Laguna, M. A novel parallel framework for scatter search. Congreso Nacional de Estadística e Investigación Operativa (SEIO 2025). Del 10 al 13 de junio de 2025.

Bibliografía

[1] D. Chalupa, “An order-based algorithm for minimum dominating set with application in graph mining,” *Information Sciences*, vol. 426, pp. 101–116, 2018.

[2] A. Duarte, J. Pantrigo, and M. Gallego, “Metaheurísticas,” *Madrid: Dykinson*, 2007.

[3] A. Culotta, “Maximizing cascades in social networks: An overview,” 2003.

[4] P. Domingos and M. Richardson, “Mining the network value of customers,” KDD ’01, (New York, NY, USA), pp. 57–66, Association for Computing Machinery, 2001.

[5] I. Lozano-Osorio, J. Sánchez-Oro, and A. Duarte, “An efficient and effective grasp algorithm for the budget influence maximization problem,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 15, no. 3, pp. 2023–2034, 2024.

[6] R. Mueller-Bady, R. Gad, M. Kappes, and I. Medina-Bulo, “Using genetic algorithms for deadline-constrained monitor selection in dynamic computer networks,” in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 867–874, 2015.

[7] R. Mueller-Bady, M. Kappes, I. Medina-Bulo, and F. Palomo-Lozano, “Optimization of monitoring in dynamic communication networks using a hybrid evolutionary algorithm,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1200–1207, 2017.

[8] R. Mueller-Bady, M. Kappes, I. Medina-Bulo, and F. Palomo-Lozano, “An evolutionary hybrid search heuristic for monitor placement in communication networks,” *Journal of Heuristics*, vol. 25, no. 6, pp. 861–899, 2019.

[9] S. P. Borgatti and M. G. Everett, “A graph-theoretic perspective on centrality,” *Social networks*, vol. 28, no. 4, pp. 466–484, 2006.

[10] M. E. Newman, “A measure of betweenness centrality based on random walks,” *Social networks*, vol. 27, no. 1, pp. 39–54, 2005.

[11] D. Zhang, E. K. Cetinkaya, and J. P. Sterbenz, “Robustness of mobile ad hoc networks under centrality-based attacks,” in *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pp. 229–235, IEEE, 2013.

[12] F. Wang, H. Du, E. Camacho, K. Xu, W. Lee, Y. Shi, and S. Shan, “On positive influence dominating sets in social networks,” *Theoretical Computer Science*, vol. 412, no. 3, pp. 265–269, 2011.

[13] M. A. Henning and N. Jafari Rad, “Locating-total domination in graphs,” *Discrete Applied Mathematics*, vol. 160, no. 13, pp. 1986–1993, 2012.

[14] V. Zverovich, *Modern applications of graph theory*. Oxford University Press, 2021.

[15] C. Alba, E. Alba, and F. Chicano, “Parallel metaheuristics: A new class of algorithms,” *Wiley-Interscience*, 2005.

[16] S. F. Crainic and M. Toulouse, “Parallel strategies for meta-heuristics,” *Handbook of Metaheuristics*, vol. 57, pp. 475–513, 2004.

[17] F. Glover and M. Laguna, “Scatter search and path relinking: Advances and applications,” *Handbook of Metaheuristics*, pp. 1–35, 2002.

[18] D. Karaboga and B. Basturk, “A parallel implementation of the artificial bee colony algorithm,” *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 2396–2401, 2008.

[19] J. García, E. Alba, and F. Chicano, “Parallel scatter search for multiobjective optimization,” in *Euro-Par 2003 Parallel Processing*, pp. 601–610, Springer, 2003.

[20] J. A. López and E. Alba, “Parallel metaheuristics for multiobjective optimization,” *Computing and Informatics*, vol. 25, no. 4, pp. 355–381, 2006.

[21] J. A. López, F. Luna, and E. Alba, “Design of a general scatter search algorithm for multiobjective optimization,” *Soft Computing*, vol. 25, pp. 9117–9136, 2021.

[22] J. Peña, J. García, and E. Alba, “Parallel scatter search for solving combinatorial optimization problems,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 925–932, ACM, 2015.

[23] Y. Song, L. Li, and Y. Wang, “A cuda-accelerated local search for the set-union knapsack problem,” *Journal of Parallel and Distributed Computing*, 2024. In press.

[24] X. Zhu, L. Gao, and L. Zhang, “A parallel binary arithmetic optimization algorithm for feature selection,” *Applied Soft Computing*, vol. 140, p. 110334, 2023.

[25] K. Sörensen and F. Glover, “Metaheuristics,” *Encyclopedia of operations research and management science*, vol. 62, pp. 960–970, 2013.

[26] P. Hansen and N. Mladenović, “First vs. best improvement: An empirical study,” *Discrete Applied Mathematics*, vol. 154, no. 5, pp. 802–817, 2006.

[27] T. A. Feo and M. G. Resende, “A probabilistic heuristic for a computationally difficult set covering problem,” *Operations Research Letters*, vol. 8, no. 2, pp. 67–71, 1989.

[28] T. A. Feo, M. G. Resende, and S. H. Smith, “A greedy randomized adaptive search procedure for maximum independent set,” *Operations Research*, vol. 42, no. 5, pp. 860–878, 1994.

[29] F. W. Glover and G. A. Kochenberger, *Handbook of metaheuristics*, vol. 57. Springer Science & Business Media, 2006.

[30] M. Prais and C. C. Ribeiro, “Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment,” *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 164–176, 2000.

[31] M. G. Resende and C. C. Ribeiro, *Optimization by GRASP - Greedy Randomized Adaptive Search Procedures*. Springer Nature, 2016.

[32] N. Mladenović and P. Hansen, “Variable neighborhood search,” *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[33] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi, “Variable neighborhood search: basics and variants,” *EURO Journal on Computational Optimization*, vol. 5, no. 3, pp. 423–454, 2017.

[34] K. Fleszar and K. S. Hindi, “Solving the resource-constrained project scheduling problem by a variable neighbourhood search,” *European Journal of Operational Research*, vol. 155, no. 2, pp. 402–413, 2004. Financial Risk in Open Economies.

[35] R. Ruiz and T. Stutzle, “A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem,” *European Journal of Operational Research*, vol. 177, no. 3, 2007.

[36] F. Glover, “A template for scatter search and path relinking,” in *European conference on artificial evolution*, pp. 1–51, Springer, 1997.

[37] F. Glover, M. Laguna, and R. Marti, “Scatter search and path relinking: Advances and applications,” *Handbook of metaheuristics*, pp. 1–35, 2003.

[38] M. Laguna and R. C. Martí, *Scatter search: methodology and implementations in C*. Springer Science & Business Media, 2003.

[39] P. Chaudhuri, *Parallel algorithms: design and analysis*. Prentice-Hall, Inc., 1992.

[40] S. V. Adve and K. Gharachorloo, “Shared memory consistency models: A tutorial,” *computer*, vol. 29, no. 12, pp. 66–76, 1996.

[41] A. Potluri and A. Singh, “Two hybrid meta-heuristic approaches for minimum dominating set problem,” pp. 97–104, 12 2011.

[42] A. Potluri and A. Singh, “Hybrid metaheuristic algorithms for minimum weight dominating set,” *Applied Soft Computing*, vol. 13, pp. 76–88, 2013.

[43] E. Álvarez-Miranda and M. Sinnl, “Exact and heuristic algorithms for the weighted total domination problem,” *Computers & Operations Research*, vol. 127, p. 105157, 2021.

[44] R. Martí, A. Martínez-Gavara, and J. Sánchez-Oro, “The capacitated dispersion problem: An optimization model and a memetic algorithm,” *Memetic Computing*, vol. 13, pp. 131–146, 2021.

[45] R. Martí, A. Duarte, and M. Laguna, “Advanced scatter search for the max-cut problem,” *INFORMS Journal on Computing*, vol. 21, no. 1, pp. 26–38, 2009.

[46] C. Helmberg and F. Rendl, “A spectral bundle method for semidefinite programming,” *SIAM Journal on Optimization*, vol. 10, no. 3, pp. 673–696, 2000.

[47] S. Burer, R. D. Monteiro, and Y. Zhang, “Rank-two relaxation heuristics for max-cut and other binary quadratic programs,” *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 503–521, 2002.

[48] P. Festa, P. M. Pardalos, M. G. Resende, and C. C. Ribeiro, “Randomized heuristics for the max-cut problem,” *Optimization methods and software*, vol. 17, no. 6, pp. 1033–1058, 2002.

[49] J. Sánchez-Oro, M. Laguna, A. Duarte, and R. Martí, “Scatter search for the profile minimization problem,” *Networks*, vol. 65, no. 1, pp. 10–21, 2015.

[50] R. Boisvert, R. Pozo, K. Remington, B. Miller, and R. Lipman, “Matrix market,” *National Institute of Standards and Technology (NIST), Gaithersburg (Maryland)*, 2004.

Tesis realizada