Combinando GRASP con *Iterated Greedy* para el problema de la máxima intersección de *k*-conjuntos.

Alejandra Casado-Ceballos
Dept. Computer Science
Universidad Rey Juan Carlos
Móstoles, España
a.casadoce.2017@alumnos.urjc.es
0000-0003-3417-6859

Sergio Pérez-Peló
Dept. Computer Science
Universidad Rey Juan Carlos
Móstoles, España
sergio.perez.pelo@urjc.es
0000-0002-1915-4160

Jesús Sánchez-Oro
Dept. Computer Science
Universidad Rey Juan Carlos
Móstoles, España
jesus.sanchezoro@urjc.es
0000-0003-1702-4941

Abraham Duarte
Dept. Computer Science
Universidad Rey Juan Carlos
Móstoles, España
abraham.duarte@urjc.es
0000-0002-4532-3124

***Resumen*—El problema de la selección de individuos de una población en función de las características que comparten ha suscitado el interés de la comunidad científica en los últimos años. En este artículo se propone un algoritmo basado en la metaheurística *Iterated Greedy* para resolver el problema de la máxima intersección de k-conjuntos (kMIS). En concreto, se trata de una variante donde se deben seleccionar k individuos de una población, con el objetivo de maximizar el número de características que dichos individuos tienen en común. Para aumentar la eficiencia del algoritmo propuesto, se plantea una nueva representación de la solución basada en bitsets. Esta representación reduce notablemente la complejidad de la evaluación de la función objetivo. La propuesta algorítmica se compara con el mejor trabajo encontrado en la literatura reciente. Los resultados computacionales muestran que el algoritmo propuesto ofrece un rendimiento muy superior al previo tanto en calidad como en tiempo de cómputo, emergiendo como uno de los algoritmos más competitivos en el contexto del kMIS.**

***Index Terms*—kMIS, *GRASP*, *Iterated Greedy*, metaheurísticas**

I. Introducción

La selección de individuos de una población siguiendo diferentes criterios es uno de los problemas de optimización más extendidos. Entre ellos, destaca una familia de problemas cuyo cometido es seleccionar un conjunto de individuos de manera que se maximicen las características que estos tienen en común. Esta familia de problemas ha resultado en una gran variedad de problemas de optimización combinatoria, donde el objetivo suele consistir en maximizar las relaciones entre dos conjuntos diferentes de elementos, sin considerar las relaciones entre los elementos del mismo conjunto.

Uno de los problemas más estudiados de este tipo es el *Maximum edge biclique (MEB)* [1], donde el objetivo es encontrar un subgrafo bipartito completo (*biclique*) con el máximo número de aristas dado un grafo bipartito cualquiera. Este problema ha sido estudiado desde distintas perspectivas, usando tanto aproximaciones heurísticas como exactas [2], [3]. Además, existen numerosas variantes de este problema, considerando nuevas restricciones como el equilibrio entre conjuntos [4] o los pesos de los elementos para seleccionarlos según su relevancia [2].

Esta investigación está financiada por el Ministerio de Ciencia, Innovación y Universidades con el proyecto con referencia PGC2018-095322-B-C22, y por la Comunidad de Madrid y los Fondos Estructurales de la Unión Europea con los proyectos con referencias S2018/TCS-4566 y Y2018/EMT-5062.

Esta investigación está centrada en el problema que maximiza la intersección entre k elementos (kMIS), y es muy similar a la familia de problemas del MEB. Dado un conjunto de elementos $E = \{e_1, e_2, \ldots, e_n\}$, donde cada e tiene un conjunto de características $F_e \subseteq F$, siendo F el conjunto de las características posibles. Siendo así, dado un número entero k, el objetivo de kMIS es seleccionar un subconjunto de elementos de E que tengan el máximo número de características de F en común.

Una solución factible para el problema kMIS está formada por un conjunto de k elementos de E. Dada una solución $S \subseteq E$, donde $|S| = k$, el valor de la función objetivo $k\text{MIS}(S)$ se evalúa como:

$$k\text{MIS}(S) = \left| \bigcap_{e \in S} F_e \right|$$

El objetivo del kMIS es encontrar una solución $S^\star$ con el máximo valor de $k\text{MIS}(S^\star)$. Más formalmente se podría definir como

$$S^\star = \arg\max_{S \in \mathbb{S}} k\text{MIS}(S)$$

donde $\mathbb{S}$ es el conjunto completo de soluciones que se pueden explorar, es decir, el espacio de búsqueda.

La figura 1(a) muestra una instancia del problema kMIS compuesta por 4 elementos y 5 posibles características con las que ser relacionados. Las características de cada elemento están definidas por una línea entre ellos. Para facilitar su identificación, el color de las líneas se corresponde con el del elemento al que pertenece la relación. Por ejemplo, las características asociadas al elemento e_1 son $F_{e_1} = \{f_1, f_2, f_3\}$, las de e_2 son $F_{e_2} = \{f_1, f_2, f_3, f_5\}$, y así con cada elemento.

Para esta instancia, se han calculado dos posibles soluciones S_1 y S_2. Por un lado, S_1, representada en la figura 1(b), está formada por los elementos $S_1 = \{e_1, e_3, e_4\}$ y el valor de la función objetivo es $k\text{MIS}(S_1) = |F_{e_1} \cap F_{e_3} \cap F_{e_4}| = 1$. Por otro lado, S_2, que se muestra en la figura 1(c), contiene los elementos $S_2 = \{e_1, e_2, e_3\}$ y tiene como valor de la función objetivo $k\text{MIS}(S_2) = |F_{e_1} \cap F_{e_2} \cap F_{e_3}| = 3$. Es por esto que,

sabiendo que tratamos de maximizar la función objetivo, está claro que S_2 es mejor que S_1.

Este problema tiene múltiples aplicaciones en distintas áreas en la vida real. Los autores de [5] muestran que el kMIS es útil para controlar la privacidad de ciertos datos. Además, en el contexto de la genética [6], el objetivo es seleccionar k genes con el máximo número de características en común. [7] propone una aplicación donde cada elemento es un artista musical y cada característica una persona. Ambos tienen relación en caso de que la persona sea admiradora del músico. El objetivo es encontrar el conjunto de k músicos maximizando los admiradores en común para organizar un festival.

El kMIS fue definido por primera vez por [5], y se trata de un problema $\mathcal{NP}$-difícil como se demostró en [8]. Aun así, para ciertas instancias especiales, el kMIS puede ser resuelto de forma óptima en tiempo polinomial. [9] propuso un algoritmo con complejidad polinomial que enumeraba todos los conjuntos cercanos de una solución factible representando la instancia como un grafo bipartito y mostrando todos los bicliques máximos del grafo si el valor de k está limitado por una constante.

[7] propuso un método exacto basado en un modelo de programación entera para el kMIS que comenzaba la búsqueda partiendo de una solución de buena calidad obtenida a través de un método de preprocesamiento rápido. Sin embargo, como un método exacto, requiere mucho tiempo de computación para solucionar de forma óptima instancias incluso de pequeño tamaño. Adicionalmente proponen tres nuevos modelos de programación entera lineal y un método de construcción heurística para el kMIS, donde dos de las fórmulas presentadas fueron adaptadas del problema MEB [10].

A pesar de la dificultad asociada a encontrar soluciones óptimas para el kMIS, el problema ha sido ignorado desde un punto de vista heurístico. Actualmente el mejor trabajo previo que utiliza un enfoque heurístico, [11] propone la combinación de distintos algoritmos heurísticos y metaheurísticos para proporcionar soluciones de alta calidad en tiempos de computación reducidos. La propuesta está basada en *Variable Neighborhood Search (VNS)*, propuesto por [12], con el objetivo de aprovechar los cambios de vecindad para mejorar las soluciones encontradas durante la búsqueda. En concreto, amplían el procedimiento constructivo propuesto en [7] para tratar varias soluciones simultáneamente combinándolas mediante Path Relinking. Además, proponen un *VNS* reactivo que modifica la fase de perturbación incluyendo una selección voraz aleatorizada de soluciones para ser seleccionadas. Finalmente, el método utiliza el cambio de vecindad para escapar de óptimos locales en caso de estancarse.

Este artículo está estructurado de la siguiente manera: la sección II describe la representación de la solución usada en esta investigación, la sección III presenta la propuesta algorítmica basada en *GRASP*, la sección IV describe el algoritmo basado en *Iterated greedy* utilizado como post-proceso para mejorar las soluciones encontradas por *GRASP*, la sección V muestra la experimentación realizada para analizar el comportamiento del algoritmo propuesto y, finalmente, la sección VI resume una serie de conclusiones derivadas de esta investigación.

II. Representación de la solución

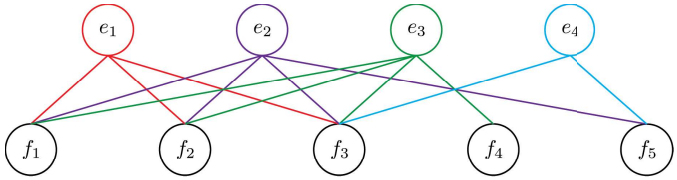
Dada una instancia $I = (B, k)$, donde $B = (E, F)$ es un grafo bipartito formado por los conjuntos de vértices E y F, una implementación directa para el kMIS consideraría que la solución se representa por un conjunto de elementos seleccionados de E. Sin embargo, es importante tener en cuenta el rendimiento de las operaciones más comunes sobre esta representación de la solución para reducir el tiempo de cómputo. Si consideramos esta representación, la evaluación de la solución se calcula mediante la intersección de las características comunes de los elementos seleccionados, que puede ser computacionalmente exigente para instancias medianas y grandes. Con el objetivo de acelerar el proceso, proponemos el uso de bitsets como adaptación a la idea original de bit-strings presentada en [11].

Un bitset se define como un array de elementos donde el valor de cada uno solo puede ser 0 o 1. Para modelar el kMIS, cada elemento $e \in E$ tiene su propio bitset B_e de tamaño $|F|$, donde el valor 1 en una posición p indica que F_e tiene la característica f_p. Además, para evaluar las características en común entre dos elementos e_i y e_j, solo es necesario realizar la operación lógica `AND` entre B_i y B_j, que se ejecuta en tiempo constante. Este comportamiento nos permite acelerar el proceso de búsqueda. La tabla I muestra la representación del bitset y la evaluación de la función objetivo de las soluciones S_1 y S_2 que aparecen en la figura 1.

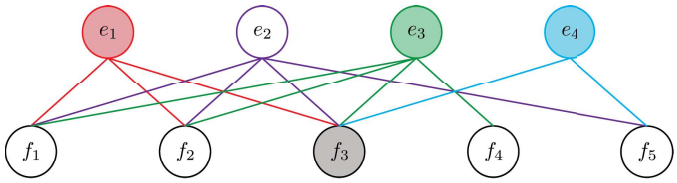
	S_1					S_2				
	f_1	f_2	f_3	f_4	f_5	f_1	f_2	f_3	f_4	f_5
e_1	1	1	1	0	0	1	1	1	0	0
e_2	1	1	1	0	1	1	1	1	0	0
e_3	1	1	1	1	0	1	1	1	0	0
e_4	0	0	1	0	1	0	0	1	0	1
AND	0	0	1	0	0	1	1	1	0	0

Tabla I: Representación de la solución S_2 con un bitset mostrada en la figura 1.

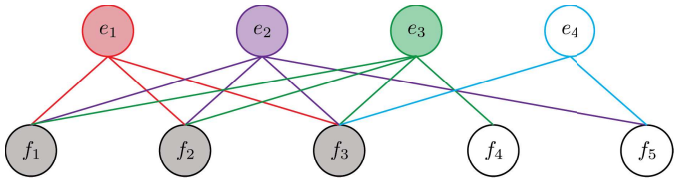
Con respecto a la representación de la solución, el elemento e_1 tiene las características f_1, f_2 y f_3, que están representadas con un 1 en la tabla. Las características f_4 y f_5 tienen un 0 en la tabla debido a que no son características que pertenezcan a e_1. Podemos hacer el mismo análisis con los elementos e_2, e_3 y e_4. Las filas resaltadas en gris indican los elementos seleccionados en cada solución. La última fila representa el resultado de realizar la operación lógica `AND` entre los bitsets de los elementos seleccionados. Para la solución S_1 el único bit activo tras la operación corresponde a la característica f_3, lo que supone que es la única característica que e_1, e_3, y e_4 tienen en común. En el caso de S_2, las características en común son f_1, f_2, y f_3, es por ello que en el resultado de realizar la operación lógica `AND` encontramos los bits de estas tres características activos. Para calcular el valor de la función objetivo a partir de esta información solo se tiene que evaluar la cardinalidad del bitset resultante, que viene dada por el número de bits activos. En el caso de S_1 el valor de la



(a) Instancia compuesta de 4 elementos y 5 características.



(b) Solución S_1



(c) Solución S_2

Figura 1: Instancia compuesta de 4 elementos y 5 características, 1(a), y dos soluciones factibles para esta instancia: S_1, 1(b), y S_2, 1(c).

función objetivo sería 1, mientras que S_2 tiene un valor de 3, siendo esta última mejor ya que se trata de un problema de maximización.

En términos de complejidad computacional, esta operación se realiza en tiempo constante, resultando en una complejidad de $O(1)$. Esta complejidad es menor que la que requiere calcular la intersección de características de los elementos seleccionados, ya que tiene una complejidad de $O(n)$.

III. *Greedy Randomized Adaptive Search Procedure*

Este trabajo propone un algoritmo basado en la metodología *Greedy Randomized Adaptive Search Procedure (GRASP)*. Se trata de una metaheurística multiarranque presentada en [13], que no fue definida formalmente hasta [14]. Está formada por dos fases: construcción y búsqueda local. La primera fase intenta generar una solución diversa y de alta calidad desde cero [15]. Por otro lado, la segunda fase es la responsable de encontrar un óptimo local respecto a cierta vecindad. La idea principal de *GRASP* recae en añadir aleatoriedad durante la fase de construcción, de forma que se diversifica la búsqueda. Al contrario que en una construcción plenamente voraz, cada construcción de *GRASP* genera una solución diferente, explorando una gran parte del espacio de búsqueda.

III-A. Constructivo

En este trabajo se propone un procedimiento constructivo que aprovecha la representación de la solución descrita en la sección II con el objetivo de generar soluciones prometedoras de las cuales partirá la búsqueda local. El procedimiento constructivo sigue el esquema tradicional del constructivo en *GRASP*. El algoritmo 1 muestra el pseudocódigo del constructivo propuesto.

Algoritmo 1 `Constructivo`$(B = (E, F), k, \alpha)$

1: $e \leftarrow Random(E)$
2: $S \leftarrow \{e\}$
3: $CL \leftarrow E \setminus \{e\}$
4: **while** $|S| < k$ **do**
5: $\quad g_{\text{mín}} \leftarrow \text{mín}_{c \in CL}\, g(c)$
6: $\quad g_{\text{máx}} \leftarrow \text{máx}_{c \in CL}\, g(c)$
7: $\quad \mu \leftarrow g_{\text{máx}} - \alpha \cdot (g_{\text{máx}} - g_{\text{mín}})$
8: $\quad RCL \leftarrow \{c \in CL : g(c) \geq \mu\}$
9: $\quad e \leftarrow Random(RCL)$
10: $\quad S \leftarrow S \cup \{e\}$
11: $\quad CL \leftarrow CL \setminus \{e\}$
12: **end while**
13: **return** S

Como suele ocurrir en *GRASP*, el primer elemento es seleccionado de forma aleatoria, favoreciendo la diversidad en las soluciones (paso 1). Tras ello, el elemento se incluye en la solución (paso 2) y la lista de candidatos *CL* se construye con todos los elementos de E menos el primer elemento seleccionado (paso 3). Iterativamente, el método añade elementos a la solución usando el criterio voraz elegido que se describirá posteriormente (pasos 4-12).

En cada iteración son evaluados el mínimo ($g_{\text{mín}}$) y el máximo ($g_{\text{máx}}$) valor de la función voraz, considerados para

guiar la construcción eligiendo entre los candidatos (pasos 5-6). Es recomendable usar una función voraz que permita calcular el valor de esta con un tiempo computacional bajo, ya que se debe evaluar en cada iteración. En el contexto de kMIS, la representación eficiente de la solución presentada en la sección II permite usar directamente la función objetivo como función voraz. El valor de la función voraz para un elemento c, denotado como $g(c)$, se calcula como el número de características que c y los elementos ya en la solución tienen en común. En términos de la representación de la solución, el valor de la función voraz se obtiene realizando la operación lógica `AND` entre las características de los elementos de la solución parcial y las características de c, y posteriormente calculando la cardinalidad del bitset resultante.

El umbral μ se calcula para determinar qué elementos deben entrar en la lista de candidatos restringida *RCL* (pasos 7-8). El umbral depende de un parámetro de entrada α, que puede tener valores en el rango entre 0 y 1. Es importante destacar que en el caso de que $\alpha = 0$ el valor de $\mu = g_{\text{máx}}$. Es por esto que solo los elementos con un valor de la función voraz igual a $g_{\text{máx}}$ son contemplados en la *RCL*, resultando en un algoritmo totalmente voraz. Por otro lado, en caso de que $\alpha = 1$, μ es igual al valor de $g_{\text{mín}}$, por lo que todos los elementos se incluyen en la *RCL* resultando un procedimiento totalmente aleatorio. Por lo tanto, cuanto mayor sea el valor de α más voraz será el algoritmo. En la sección V se analiza el valor más apropiado para este parámetro.

El siguiente elemento que se añade en la solución es seleccionado de forma aleatoria de la *RCL* (paso 9). Finalmente, la *CL* es actualizada eliminando el elemento seleccionado de forma que no se pueda elegir dos veces el mismo elemento. El método termina cuando son seleccionados k elementos de E, devolviendo la solución construida (paso 13).

III-B. Búsqueda local

La segunda fase de *GRASP* consiste en encontrar un óptimo local respecto a cierta vecindad empezando por la solución generada en el constructivo. La literatura de *GRASP* ha considerado tanto búsquedas locales simples como otras más complejas. Algunos trabajos han llegado a incluir metahurísticas completas usando *Variable Neighborhood Search* [16] o incluso algoritmos genéticos [17] en la segunda fase de *GRASP* con el objetivo de explorar más allá en el espacio de búsqueda.

En este trabajo se propone una simple pero efectiva búsqueda local, con el objetivo de encontrar un óptimo local en un tiempo computacional reducido. El primer elemento que se debe identificar en una búsqueda local es el movimiento que se aplica en la búsqueda. Debido a que una solución para el kMIS está formada por exactamente k elementos y con el objetivo de mantener la factibilidad de las soluciones exploradas en el vecindario, se propone un movimiento de intercambio. Dado un elemento $e_i \in S$ y un elemento $e_j \in E \setminus S$, se intercambia e_i con e_j. En términos matemáticos,

$$\textit{Intercambio}(S, e_i, e_j) = (S \setminus e_i) \cup e_j$$

La definición de un movimiento nos permite establecer la vecindad $N(S)$ que se explora en la búsqueda local propuesta como un conjunto de soluciones que se pueden generar a partir de S realizando un único movimiento de intercambio. Más formalmente,

$$N(S) = \{S' \leftarrow \textit{Intercambio}(S, e_i, e_j),\ \forall e_i \in S \wedge \forall e_j \in E \setminus S\}$$

Habiendo definido el movimiento y la vecindad generada a partir de este, el último elemento que falta por definir es la manera en la que se recorre esta vecindad. Tradicionalmente se consideran dos formas de recorrer una vecindad: *best improvement* y *first improvement*. Por un lado *best improvement* explora la vecindad de una solución al completo, realizando aquel movimiento que genera una solución de mayor calidad. Por otro lado, *first improvement* realiza el primer movimiento que mejora la solución de partida.

Debido a que *best improvement* requiere una exploración completa de la vecindad en cada iteración, suele ser computacionalmente más lenta que *first improvement*. En kMIS, elegimos *first improvement* con el objetivo de mantener la rapidez del algoritmo al incluir la búsqueda local. Al elegir *first improvement* el orden en el que las soluciones son exploradas es importante por lo que, para evitar sesgar la búsqueda, se ordenan de forma aleatoria en cada iteración. El algoritmo 2 muestra el pseudocódigo de la búsqueda local propuesta.

Algoritmo 2 `Búsqueda_local`$(I = (E, F, k), S)$

```
1:  best ← kMIS(S)
2:  for e_i ∈ S do
3:      for e_j ∈ E \ S do
4:          S ← Intercambio(S, e_i, e_j)
5:          if kMIS(S) > best then                ▷ Mejora
6:              go to 1
7:          end if
8:          S ← Intercambio(S, e_j, e_i)
9:      end for
10: end for
11: return S
```

La función comienza recorriendo los elementos de S de forma aleatoria (paso 2). Tras esto, para cada elemento $e_i \in S$ se realiza un movimiento de intercambio con cada elemento $e_j \in E \setminus S$ (paso 4). En caso de encontrar una mejora (paso 5) se reinicia la búsqueda (paso 6). Si no, el movimiento de intercambio se deshace, volviendo a la solución inicial. El procedimiento termina cuando no se encuentra ninguna mejora, devolviendo un óptimo local de la solución inicial S (paso 11).

La búsqueda local puede mejorar considerablemente ya que realizar un intercambio implica revaluar la solución en cada iteración. Para reducir el tiempo empleado en la búsqueda local, proponemos una búsqueda local más eficiente. Esta nueva búsqueda local es capaz de decidir si un intercambio generará una mejor solución o no realizando una única operación lógica `AND`. Para ello, el paso 4 se reemplaza por la operación `AND`

entre el bitset de la solución, cuando el elemento e_i ha sido borrado, y el bitset del elemento e_j. Si la cardinalidad del bitset resultante es mayor que la original, se realiza el movimiento. En caso contrario, el intercambio se omite, eliminando el paso 8 del algoritmo. La sección V analiza los efectos que tiene esta búsqueda local eficiente en el algoritmo completo *GRASP*.

IV. *Iterated Greedy*

Iterated greedy (IG) es una metaheurística propuesta por primera vez en 2007 [18], que se ha convertido en una de las más usadas en los últimos años. Las bases de *IG* residen en explorar un espacio de búsqueda alternando intensificación y diversificación iterativamente gracias a sus dos fases, la de destrucción y la de reconstrucción.

En esta propuesta se aplica *IG* sobre la mejor solución encontrada tras ejecutar las dos fases de *GRASP*. El algoritmo consiste en una primera fase de destrucción que resulta en una solución parcial, seguida de una reconstrucción en la que se consigue volver a obtener una solución completa con k elementos en ella. Finalmente, a la solución reconstruida se le aplica un proceso de optimización local. Estos tres pasos se ejecutan iterativamente en caso de que no se haya superado el número máximo de iteraciones sin mejora γ, parámetro que se analiza en la sección V.

Para poder realizar la destrucción se debe establecer el parámetro de entrada β, que determina el porcentaje de elementos de la solución que será eliminado y cuyo valor elegido se determina de forma experimental en la sección V.

Por otro lado, se debe especificar cómo será cada una de las fases del *IG* implementado. Para ello se proponen dos alternativas: aleatoria y voraz. Es por esto que el algoritmo *IG* propuesto tiene 4 combinaciones diferentes variando el método aleatorio y el voraz tanto en el caso de la destrucción como en el de la reconstrucción. La combinación elegida y el motivo se analizan en la sección V.

En el caso del método aleatorio para la destrucción se realiza seleccionando elementos aleatorios de forma iterativa para quitarlos de la solución hasta eliminar $\beta \times k$ elementos. La reconstrucción aleatoria actúa de manera similar, seleccionando elementos iterativamente de forma aleatoria hasta volver a obtener una solución factible.

Para aplicar el procedimiento voraz se vuelve a usar la representación de la solución explicada en la sección II, realizando la operación lógica `AND` para determinar el mejor elemento a añadir en cada iteración en el caso de la reconstrucción. Por otro lado, para destruir la solución se eligen los elementos que menos características tengan en común con el resto de elementos de la solución.

V. Experimentos y resultados

Esta sección tiene dos objetivos principales. Por un lado, determinar qué parámetros son los más adecuados para el algoritmo propuesto. Por otro, es necesario evaluar el rendimiento del algoritmo desarrollado comparándolo con el mejor método encontrado en el estado del arte. Para cumplir los dos objetivos se dividen los experimentos en dos fases: los experimentos preliminares y los finales.

El algoritmo propuesto se ha implementado en Java 11, y todos los experimentos se han realizado en un AMD EPYC 7282 (2.8 GHz) y 8 GB RAM. El conjunto de instancias a la hora de realizar las comparaciones son las usadas en [11], que es el mejor método encontrado en la literatura, para poder llevar a cabo una comparativa justa. Consiste en un total de 238 instancias donde el número de elementos y características va de 32 hasta 300 (ver [11] para más detalles sobre estas instancias).

Todos los experimentos se evalúan según las siguientes métricas: Promedio, la media del valor de la función objetivo; Tiempo (s), el tiempo total de cómputo medido en segundos; Desv. (%), la desviación porcentual media con respecto al mejor resultado del experimento; y # Mejores, el número de soluciones mejores encontradas en el experimento.

La fase de experimentos preliminares está diseñada para ajustar los parámetros de entrada de cada algoritmo. Con el objetivo de no sobreajustar los parámetros, en estos experimentos se considera un subconjunto representativo de 27 del total de 238 instancias.

El primer parámetro estudiado es α del constructivo *GRASP*, donde se han evaluado $\alpha = \{0.25, 0.50, 0.75, \textit{RND}\}$. En concreto, se han ejecutado 1000 iteraciones del algoritmo *GRASP* completo. La Tabla II muestra los resultados obtenidos en este experimento.

α	Promedio	Tiempo (s)	Desv. %	# Mejores
0.25	24.26	**14.66**	5.31 %	18
0.50	22.37	17.26	25.97 %	11
0.75	21.63	17.80	28.94 %	10
RND	**25.07**	15.99	**0.66 %**	**25**

Tabla II: Comparativa de los diferentes valores de α considerados construyendo y mejorando 1000 soluciones independientes.

En este experimento, el mejor valor obtenido es $\alpha = \textit{RND}$ con una desviación del 0.66 % y 25 de las 27 mejores soluciones encontradas. Una vez determinado esto, se analiza el impacto de la búsqueda local eficiente, comparando esta variante con la básica. Los resultados indican que la búsqueda eficiente es 10 veces más rápida, reduciendo su tiempo de ejecución a 1.20 segundos frente a los 15.99 segundos originales, obteniendo los mismos resultados respecto a la calidad.

En cuanto a *Iterated Greedy*, se deben establecer los parámetros β y el número de iteraciones sin mejora. Cabe destacar que la solución de partida de *Iterated Greedy* es siempre la mejor de las 1000 iteraciones GRASP realizadas. A la hora de ajustar el primer parámetro se evalúa $\beta = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, llegando a la conclusión de que el mejor valor se obtiene con $\beta = 0.2$, produciéndose a partir de este valor un estancamiento en la calidad mientras continúa aumentando el tiempo de ejecución. Por otro lado, en cuanto al número de iteraciones sin mejora, se estudian los valores $\gamma = \{5, 10, 15, 20\}$, convergiendo el algoritmo con $\gamma = 10$.

Tal y como se indica al inicio de la sección, el experimento final tiene como objetivo el análisis del rendimiento del algoritmo propuesto frente al mejor resultado encontrado en la literatura. En concreto, el mejor algoritmo del estado del arte se propone en [11], donde los autores presentan el algoritmo *Variable Neighborhood Search (VNS)* reactivo (*Reactive VNS*) descrito en la Sección I. La Tabla III muestra los resultados obtenido por el algoritmo propuesto (*GRASP+IG*) frente a *Reactive VNS*, donde el mejor resultado de cada métrica se resalta en negrita.

Algoritmo	Promedio	Tiempo (s)	Desv. (%)	# Mejores
Reactive VNS	32.59	21.01	2.04	186
GRASP+IG	**33.00**	**0.85**	**0.57**	**225**

Tabla III: Comparativa entre el algoritmo propuesto (*GRASP+IG*) y el estado del arte (*Reactive VNS*) sobre el total de 238 instancias.

Como se puede observar, *GRASP+IG* obtiene mejores resultados en todas las métricas evaluadas. En concreto, obtiene 225 mejores soluciones frente a las 186 obtenidas por *Reactive VNS*. Cabe destacar que, de esas mejores soluciones encontradas, *GRASP+IG* es estrictamente mejor que *Reactive VNS* en 52 de ellas, mientras que *Reactive VNS* solo obtiene 14 soluciones estrictamente mejores. Además, la desviación obtenida del 0.57 % respecto al 2.04 % de *Reactive VNS* indica que, en aquellas soluciones en las que no es capaz de alcanzar el mejor valor, se queda muy cerca del mismo. Por último, y no menos importante, hay que resaltar la relevancia de la representación de la solución descrita en la Sección II, así como el impacto de la búsqueda local eficiente, que llevan a *GRASP+IG* a requerir, en promedio, menos de un segundo por instancia, mientras que *Reactive VNS* necesita más de 20 segundos por cada instancia analizada. Finalmente, se ha ejecutado el test no-paramétrico de Wilcoxon para confirmar que hay diferencias estadísticamente significativas entre los resultados de ambos algoritmos. El p-valor obtenido, menor que 0.001, confirma la hipótesis inicial. Estos resultados confirman la superioridad del algoritmo propuesto para resolver el problema del kMIS.

VI. Conclusiones

El algoritmo propuesto, basado en *Iterated Greedy*, con una construcción inspirada en *Greedy Randomized Adaptive Search Procedure*, es capaz de obtener resultados prometedores tanto en calidad como en tiempo de cómputo. La representación alternativa de la solución propuesta, junto con un diseño inteligente de la búsqueda local, consigue llevar al algoritmo a obtener resultados muy prometedores en tiempos de cómputo despreciables (menos de un segundo por instancia), lo que convierte al algoritmo propuesto en uno de los métodos más competitivos para resolver el kMIS.

Como trabajo futuro se propone la propuesta de funciones objetivo alternativas que permitan enriquecer el espacio de búsqueda. Además, debido al tiempo tan reducido de cómputo requerido por el algoritmo propuesto, sería interesante probar nuevas estrategias de búsqueda, como Tabú Search, que permita ampliar la región del espacio de búsqueda explorado. Estas nuevas estrategias eventualmente llevarán al algoritmo a encontrar mejores soluciones.

Referencias

[1] R. Peeters, "The maximum edge biclique problem is np-complete," *Discrete Applied Mathematics*, vol. 131, no. 3, pp. 651–654, 2003.

[2] A. Pandey, G. Sharma, and N. Jain, "Maximum weighted edge biclique problem on bipartite graphs," in *Conference on Algorithms and Discrete Applied Mathematics*. Springer, 2020, pp. 116–128.

[3] Y. Wang, S. Cai, and M. Yin, "New heuristic approaches for maximum balanced biclique problem," *Information Sciences*, vol. 432, pp. 362–375, 2018.

[4] M. Li, J.-K. Hao, and Q. Wu, "General swap-based multiple neighborhood adaptive search for the maximum balanced biclique problem," *Computers & Operations Research*, vol. 119, p. 104922, 2020.

[5] S. A. Vinterbo, "A Note on the Hardness of the k-Ambiguity Problem," Decision Systems Group/Harvard Medical School, 75 Francis Street, Boston, MA 02115, USA, Tech. Rep. DSG-TR-2002-006, May 2002.

[6] D. Nussbaum, S. Pu, J.-R. Sack, T. Uno, and H. Zarrabi-Zadeh, "Finding maximum edge bicliques in convex bipartite graphs," in *International Computing and Combinatorics Conference*. Springer, 2010, pp. 140–149.

[7] E. T. Bogue, C. C. de Souza, E. C. Xavier, and A. S. Freire, "An integer programming formulation for the maximum k-subset intersection problem," in *Combinatorial Optimization*, P. Fouilhoux, L. E. N. Gouveia, A. R. Mahjoub, and V. T. Paschos, Eds. Cham: Springer International Publishing, 2014, pp. 87–99.

[8] E. C. Xavier, "A note on a maximum k-subset intersection problem," *Information Processing Letters*, vol. 112, no. 12, pp. 471–472, 2012.

[9] B. Ganter and K. Reuter, "Finding all closed sets: A general approach," *Order*, vol. 8, no. 3, pp. 283–290, 1991.

[10] V. Acuña, C. E. Ferreira, A. S. Freire, and E. Moreno, "Solving the maximum edge biclique packing problem on unbalanced bipartite graphs," *Discrete Applied Mathematics*, vol. 164, pp. 2–12, 2014.

[11] F. C. Dias, W. A. Tavares, and J. R. de Freitas Costa, "Reactive VNS algorithm for the maximum k-subset intersection problem," *Journal of Heuristics*, vol. 26, no. 6, pp. 913–941, 2020.

[12] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[13] 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.

[14] 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.

[15] M. G. C. Resende and C. C. Ribeiro, *Greedy Randomized Adaptive Search Procedures: Advances and Extensions*. Cham: Springer International Publishing, 2019, pp. 169–220.

[16] Y. Gao, X. Gao, X. Li, B. Yao, and G. Chen, "An embedded GRASP-VNS based two-layer framework for tour recommendation," *IEEE Transactions on Services Computing*, 2019.

[17] A. Saad, A. Kafafy, O. Abd-El-Raof, and N. El-Hefnawy, "A GRASP-Genetic metaheuristic applied on multi-processor task scheduling systems," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*. IEEE, 2018, pp. 109–115.

[18] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European journal of operational research*, vol. 177, no. 3, pp. 2033–2049, 2007.