# XIII Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (XIII MAEB)

MAEB 2.2:
SCATTER SEARCH Y VNS

# Encontrando grafos bipartitos completos mediante Búsqueda de Vecindad Variable

Juan David Quintana
*Dept. Computer Sciences*
*Universidad Rey Juan Carlos*
Madrid, España
juandavid.quintana@urjc.es

Jesús Sánchez-Oro
*Dept. Computer Sciences*
*Universidad Rey Juan Carlos*
Madrid, España
jesus.sanchezoro@ujrc.es

Abraham Duarte
*Dept. Computer Sciences*
*Universidad Rey Juan Carlos*
Madrid, España
abraham.duarte@urjc.es

***Resumen*—El Problema del Máximo Biclique Balanceado, o *Maximum Balanced Biclique Problem* (MBBP), consiste en encontrar el biclique de tamaño máximo inducido por un grafo bipartito no completo. Se tiene la restricción adicional de que tanto el tamaño del grafo de partida como el biclique inducido tienen el mismo número de vértices en cada una de sus capas, por lo tanto se dice que son balanceados. Algunas de sus aplicaciones se dan en el diseño de circuitos en integración a gran escala (VLSI) o el diseño de sistemas nanoelectrónicos, entre otras. Se ha demostrado que este problema de optimización combinatoria es *NP*-Duro. En la literatura se han propuesto diversas heurísticas para solucionarlo, generalmente basadas en la eliminación de vértices, y más recientemente se ha propuesto un algoritmo evolutivo que tiene los mejores resultados en la literatura. En este trabajo se propone el uso de la metodología RVNS para abordar este problema. Esta elección se justifica en la díficultad para diseñar una búsqueda local efectiva para este problema.**

***Index Terms*—biclique, Reduced VNS, bipartito.**

## I. Introducción

Sea $G(L, R, E)$ un grafo bipartito balanceado en el que $L$ y $R$ son dos conjuntos (o capas) de vértices con la misma cardinalidad (i.e., $|L| = |R| = n$) y $E$ es el conjunto de aristas. Al ser un grafo bipartito, $L \cap R = \emptyset$, una arista solo puede conectar un vértice $v \in L$ con otro $u \in R$, i.e., $\forall (v, u) \in E\ v \in L \wedge u \in R$. Además, definimos un biclique $B(L'R', E')$ como el grafo inducido por $G$, en el que $L' \subset L$, $R' \subset R$, de forma que todo vértice $v \in L'$ está conectado a todo vértice $u \in R'$. Dicho de otra forma, $B$ es un grafo bipartito completo, también llamado biclique.

Dado un grafo bipartito balanceado $G(L, R, E)$, este trabajo se centra en resolver el Problema del Máximo Biclique Balanceado, más conocido como *Maximum Balanced Biclique Problem* (MBBP) en inglés, que consiste en identificar un biclique balanceado $B^\star(L', R', E')$ con el mayor número de vértices por capa. En otras palabras, el objetivo del MBBP es maximizar la cardinalidad de los conjuntos $L'$ y $R'$.

La Figura 1 presenta un ejemplo de grafo bipartito con 8 vértices, 13 aristas, y dos posibles soluciones para el MBBP. La figura 1(b) muestra una solución $B_1(L_1, R_1, E_1)$ con dos vértices en cada capa. En este caso, $L_1 = \{\texttt{A}, \texttt{B}\}$, y $R_1 = \{\texttt{F}, \texttt{G}\}$. Las aristas que pertenecen al biclique inducido se representan con una línea continua, mientras que aquellas con un extremo fuera de la solución se representan con una línea discontinua. Obsérvese que no es posible insertar nuevos vértices en la solución, ya que en dicho caso el grafo bipartito inducido resultante no sería un biclique balanceado. Por ejemplo, no es posible añadir los vértices `E` o `H` porque no son adyacentes a los vértices `B` y `A`, respectivamente. Además, no es posible añadir vértices en la capa $L_1$ ya que el biclique inducido no sería balanceado (i.e., $|L_1| \neq |R_1|$).

La figura 1(c) muestra una solución $B_2(L_2, R_2, E_2)$ de mejor calidad, ya que tiene 3 vértices en cada capa. En concreto, $L_2 = \{\texttt{B}, \texttt{C}, \texttt{D}\}$, y $R_2 = \{\texttt{F}, \texttt{G}, \texttt{H}\}$. De nuevo, no es posible añadir más vértices sin violar la restricción de tener un biclique balanceado.

Se ha demostrado que este problema es $\mathcal{NP}$-Duro en varios trabajos previos [1]–[3]. Algunos resultados teóricos proponen cotas para el tamaño máximo que la solución óptima puede alcanzar [4], y se demostrado difícil de aproximar dentro de un determinado factor [5].

Los grafos biclique han demostrado ser de utilidad en varias aplicaciones prácticas, la mayoría de ellas en el campo de la biología: agrupación de datos de microarrays [6]–[8], optimización de la reconstrucción del arbol filogenético [9], entre otras [10]–[12]. En particular, el MBBP tiene aplicaciones adicionales en diversos campos: diseño eficiente de circuitos en integración a gran escala (VLSI) [13], o el diseño de nuevos sistemas nanoelectrónicos [14]–[16], entre otras.

A pesar de las aplicaciones prácticas del MBBP, no se han propuesto muchos algoritmos eficientes para solucionarlo, principalmente debido a la dificultad del problema. Sin embargo, si del MBBP eliminamos la restricción de que la solución deba ser balanceada, entonces el problema resultante sería resoluble en tiempo polinómico [17], pero la mayoría de las soluciones serían desbalanceadas, haciendo que los resultados no sean adaptables al problema que estamos considerando.

La mayoría de los planteamientos previos siguen un enfoque destructivo en el que la solución inicial contiene todos los vértices y la heurística elimina iterativamente vértices en $L'$ y $R'$ hasta que la solución resultante es factible. Estos algoritmos se diferencian entre sí principalmente por el criterio

Este trabajo ha sido financiado parcialmente por el Ministerio de Economía y Competitividad, ref. TIN2015-65460-C2-2-P.

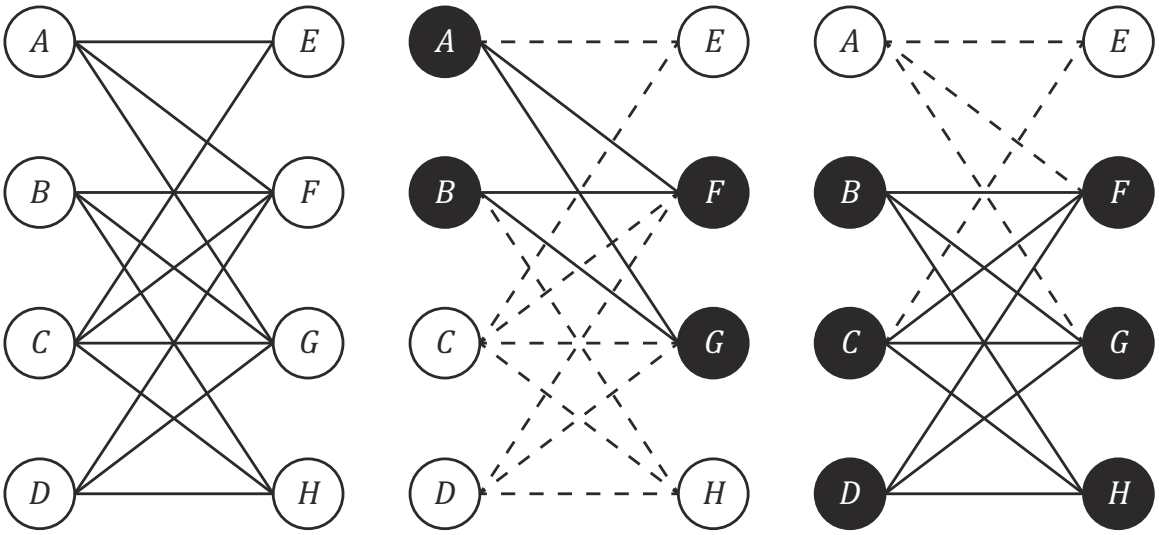


Figura 1: 1(a) Grafo bipartito con 8 vértices y 13 aristas, 1(b) una solución factible con 2 vértices en cada capa (A y B en $L'$, F y G en $R'$) , y 1(c) una solución distinta con 3 vértices en cada capa (B, C y D en $L'$, F, G, y H en $R'$)

utilizado para elegir los vértices a eliminar. En particular, [14] selecciona los vértices de mayor grado, mientras que [15] elimina los de mayor número de vértices de mínimo grado en la otra capa. Además, algunos algoritmos han intentado combinar ambos criterios [18], [19]. El mejor algoritmo encontrado en la literatura consiste en un algoritmo evolutivo [20] que propone un nuevo operador de mutación así como una nueva busqueda local para mejorar la calidad de las soluciones generadas.

El resto de este artículo está organizado de la siguiente forma: la Sección II describe la propuesta algorítmica para el MBBP; la Sección III presenta los experimentos realizados para evaluar la calidad de la propuesta algortimica; y la Sección IV plantea las conclusiones de esta investigación.

## II. Reduced Variable Neighborhood Search

La Búsqueda de Vecindad Variable (VNS por sus siglas en inglés *Variable Neighborhood Search*) [21] es una metaheuristica basada en cambios sistemáticos de vecindad. Al ser una metaheurística, no garantiza la optimalidad de las soluciones obtenidas, pero se centra en obtener soluciones de alta calidad en un tiempo razonable de cómputo. La constante evolución de VNS ha dado lugar a diversas variantes, entre la que podemos destacar *Basic VNS*, *Reduced VNS (RVNS)*, *Variable Neighborhood Descent (VND)* , *General VNS*, *Skewed VNS*, *Variable Neighborhood Decomposition Search*, entre otras.

La mayoría de las variantes se diferencian en la forma de explorar las vecindades establecidas. En particular, *Variable Neighborhood Decomposition Search* realiza una exploración totalmente determinista del espacio de soluciones, mientras que la exploración realizada por RVNS es totalmente estocástica. Algunas variantes combinan cambios de vecindad tanto deterministas como estocásticas (e.g., *Basic VNS*, *General VNS*).

Como se menciona en trabajos previos [19], [20], diseñar una búsqueda local para el MBBP es una tarea difícil principalmente por la complejidad de mantener una solución factible (i.e. un biclique balanceado) tras eliminar o añadir vértices en una solución previa. En otras palabras, el MBBP no es idóneo para proponer métodos de búsqueda local que permitan encontrar un óptimo local en relación a una solución determinada. Por lo tanto, este trabajo propone un algoritmo Reduced VNS, que se basa en la exploración aleatoria de las vecindades establecidas.

RVNS es una variante de VNS útil para instancias grandes en las que la búsqueda local es muy costosa en tiempo, o para aquellos problemas en los que no es fácil diseñar un método de búsqueda local.

**Algoritmo 1** $RVNS(B, k_{\text{máx}}, t_{\text{máx}})$

```
1: repeat
2:     k ← 1
3:     while k ≤ k_máx do
4:         B' ← Shake(B, k)
5:         k ← NeighborhoodChange(B, B', k)
6:     end while
7: until CPUTime() ≤ t_máx
8: return B
```

El algoritmo requiere de dos parámetros de entrada: $B$, una solución factible de partida, que se puede generar aleatoriamente o mediante un procedimiento constructivo más elaborado; y $k_{\text{máx}}$, la vecindad máxima a explorar.

Cada iteración de RVNS empieza desde la vecindad inicial (paso 2). A continuación, el método explora cada una de las vecindades establecidas (pasos 3-6) de la siguiente forma. En

primer lugar, el método genera una solución al azar $B'$ en la vecindad $k$ de la solución actual $B$ con el método *Shake* (paso 4). Luego el método *NeighborhoodChange* (paso 5) se encarga de seleccionar la siguiente vecindad a explorar. En particular, si la nueva solución $B'$ es mejor que la solución de partida $B$, entonces se actualiza ($B \leftarrow B'$), y reinicia la búsqueda desde la vecindad inicial ($k \leftarrow 1$). En caso contrario, la búsqueda continúa con la siguiente vecindad ($k \leftarrow k+1$). Una iteración de RVNS termina cuando se ha alcanzado la vecindad máxima establecida $k_{\text{máx}}$. Es importante mencionar que la vecindad máxima utilizada en RVNS es generalmente pequeña debido a la naturaleza aleatoria del método *Shake*, ya que un valor mayor para $k_{\text{máx}}$ produciría resultados equivalentes a reiniciar la búsqueda desde una nueva solución inicial. El método RVNS se ejecuta hasta que se alcanza un tiempo límite de cómputo $t_{\text{máx}}$.

### *II-A. Procedimiento constructivo*

La solución inicial para RVNS puede ser generada al azar o con un procedimiento constructivo más elaborado. Este trabajo propone un procedimiento constructivo basado en *Greedy Randomized Adaptive Search Procedure* (GRASP) [22]. Esta metodología considera una función voraz que evalúa la importancia de insertar cada vértice en la solución a construir. En el Algoritmo 2 se presenta el pseudocódigo del procedimiento constructivo propuesto.

**Algoritmo 2** *Construct*$(G = (L, R, E), \alpha)$

1: **function** UpdateLayer($CL_1, CL_2, \alpha$)
2: $g_{\text{mín}} \leftarrow \text{mín}_{v \in CL_1}\, g(v)$
3: $g_{\text{máx}} \leftarrow \text{máx}_{v \in CL_1}\, g(v)$
4: $\mu \leftarrow g_{\text{mín}} + \alpha \cdot (g_{\text{máx}} - g_{\text{mín}})$
5: $RCL \leftarrow \{v \in CL_1 \ ; \ g(v) \leq \mu\}$
6: $v \leftarrow Random(RCL)$
7: $CL_1 \leftarrow CL_1 \setminus \{v\}$
8: $CL_2 \leftarrow CL_2 \setminus \{u \in CL_2 \ : \ (v,u) \notin E\}$
9: **return** $v$
10: **end function**
11: $v \leftarrow Random(L)$
12: $L' \leftarrow \{v\}$
13: $CL_L \leftarrow L \setminus \{v\}$
14: $CL_R \leftarrow \{u \in R \ : \ (v,u) \in E\}$
15: **while** $CL_L \neq \emptyset$ **and** $CL_R \neq \emptyset$ **do**
16: $v_r \leftarrow$ UpdateLayer($CL_R, CL_L, \alpha$)
17: $R' \leftarrow R \cup \{v_r\}$
18: $v_l \leftarrow$ UpdateLayer($CL_L, CL_R, \alpha$)
19: $L' \leftarrow L \cup \{v_l\}$
20: **end while**
21: **return** $B = (L', R', \{(v,u)\ v \in L' \wedge u \in R'\})$

El método empieza por seleccionar un vértice al azar de la capa $L$ (paso 11), insertarlo la capa correspondiente $L'$ de la solución (paso 12). Entonces, se crean dos listas de candidatos (CL), una por cada capa del grafo (paso 13-14). Obsérvese que ambas listas de candidatos solo contienen aquellos vértices de cada capa que pueden ser seleccionados manteniendo la solución factible (i.e., la solución es un grafo bipartito completo). Por lo tanto, en este paso inicial, $CL_L$ contiene todos los vértices de $L$ exceptuando al vértice seleccionado $v$. Por otra parte, $CL_R$ contiene todos los vértices adyacentes a $v$ en $R$, ya que de otra forma la solución no sería un grafo bipartito completo. A continuación, el método añade un vértice en la capa $R'$ y luego en $L'$ iterativamente, mientras que queden candidatos en ambas capas (paso 16-19).

La selección del vértice siguiente se describe en la función *UpdateLayer* que requiere de tres parámetros: la lista de candidatos desde la que se debe elegir el vértice, $CL_1$, la lista de candidatos de la otra capa, $CL_2$, y el parámetro $\alpha$ que determina el grado de voracidad / aleatoriedad de la selección. Una función voraz $g$ que evalúa la calidad de un vértice candidato $v$ debe ser definida. En este trabajo, proponemos como función el número de vértices adyacentes en la lista de candidatos de la capa opuesta. Formalmente,

$$g(v, CL) \leftarrow \{u \in CL \ : \ (v,u) \in E\}$$

El primer paso para seleccionar el vértice siguiente consiste en obtener los valores mínimo ($g_{\text{mín}}$) y máximo ($g_{\text{máx}}$) para la función voraz (paso 2-3). A continuación, se construye (paso 5) la lista de candidatos restringida (*RCL*) con aquellos vértices cuyo valor de la función objetivo sea mayor o igual que el umbral hallado previamente $\mu$ (paso 4). Los valores para el parámetro $\alpha$ están en el rango 0–1, dónde $\alpha = 0$ implica que el método es totalmente aleatorio, y $\alpha = 1$ transforma el algoritmo en completamente voraz. El siguiente vértice se selecciona al azar de *RCL* (paso 6), y luego se actualiza cada lista de candidatos. En particular, $CL_1$ se actualiza mediante quitar el vértice seleccionado de sí mismo, mientras $CL_2$ se actualiza quitando aquellos vértices que no son adyacentes a $v$, debido a que no pueden ser elegidos en iteraciones futuras sin que la solución se vuelva infactible.

### *II-B. Shake*

El método *Shake* es una etapa dentro de la metodología VNS diseñada para escapar de óptimos locales encontrados durante la fase de búsqueda. Consiste en perturbar aleatoriamente una solución con el objetivo de explorar regiones más amplias en el espacio de búsqueda. Esta fase de la metodología VNS se centra en diversificar la búsqueda.

Dada una vecindad $k$, el método *Shake* propuesto en este trabajo elimina $k$ elementos al azar de cada capa. La solución resultante es factible pero el valor de función objetivo siempre es menor, ya que se reduce el número de vértices seleccionados.

Teniendo en cuenta las restricciones del problema, si un vértice $v$ se añade a la solución, entonces todos los vértices en la capa opuesta que no sean adyacentes a $v$ no pueden ser añadidos en iteraciones futuras, ya que en dicho caso la solución obtenida no sería un biclique. Sin embargo, quitar algunos vértices con el método *Shake* puede permitir eventualmente que nuevos vértices sean añadidos a la solución

(i.e., aquellos que no eran adyacentes a ninguno de los vértices eliminados).

Por lo tanto, proponemos una etapa de reconstrucción que se ejecuta después de cada *Shake*. En concreto, la fase de reconstrucción intenta añadir nuevos vértices a la solución, aquellos que no eran candidatos factibles antes de ejecutar el método *Shake*.

La etapa de reconstrucción siempre mejora o, al menos, mantiene la calidad de la solución obtenida tras la llamada al método *Shake*. Obsérvese que la solución reconstruida solo supera a la inicial si y solo si la etapa de reconstrucción es capaz de insertar más de $k$ vértices en cada capa.

La naturaleza aleatoria de este procedimiento hace que sea difícil obtener mejoras en la calidad de la solución. Para mitigar este efecto se proponen cuatro variantes del método *Shake*. Estas se diferencian en cómo se realizan las fases de destrucción y reconstrucción de la solución, cada fase puede ser respectivamente aleatoria (R) o voraz (G), con lo que obtenemos cuatro variantes: RR, RG, GR, GG. La Tabla I muestra el comportamiento voraz o aleatorio de cada una de las variantes propuestas. Por ejemplo, la variante de *Shake* GR realiza una destrucción voraz con una reconstrucción aleatoria.

| Variante | Destrucción | Reconstrucción |
|---|---|---|
| RR | Aleatoria | Aleatoria |
| RG | Aleatoria | Voraz |
| GR | Voraz | Aleatoria |
| GG | Voraz | Voraz |

Tabla I: Enumeración de las cuatro variantes de *Shake* propuestas.

En la fase de reconstrucción se utiliza el mismo método constructivo que genera la solución inicial, pero modificado para ser totalmente aleatorio ($\alpha = 0$) o totalmente voraz ($\alpha = 1$). En la fase de destrucción se plantea un algoritmo, con una estructura similar a la del método constructivo, que elimina vértices de una solución factible. Para que sea una destrucción voraz es necesario definir una nueva función heurística $g'$ que puntúe los vértices a eliminar. El valor de esta función heurística para cada vértice $v \in S$ se calculará por el número de vertices en la otra capa con los que no está conectados.

## III. Resultados experimentales

En esta sección se presentan dos grupos de experimentos: los experimentos preliminares, realizados para ajustar los parámetros del algoritmo propuesto; y el experimento final, diseñado para evaluar la calidad de nuestra propuesta algorítmica y compararla con el algoritmo previo de la literatura. Todos los algoritmos fueron implementados en Java y se ejecutaron en un sistema con una CPU Intel i7 (7660U) y 8 GB de RAM, en la máquina virtual de Java 8.

Se ha utilizado el mismo conjunto de datos presentado en [20], facilitado por dichos autores, que consiste en 30 grafos bipartitos balanceados con tamaños $n = \{250, 500\}$, donde $n$ representa el número de vértices en cada capa; y con diferentes probabilidades $p = \{85\,\%, 90\,\%, 95\,\%\}$ de que una arista exista en el grafo. Se trata de grafos densamente poblados en los que falta un porcentaje bajo de aristas para que lleguen a ser completos. Este conjunto de datos se utiliza para comparar el rendimiento de nuestro algoritmo con el estado del arte. A parte de las instancias utilizadas para presentar sus resultados, los autores de [20] también facilitan otras instancias generadas con los mismos parámetros, no utilizadas en su artículo, y que en este trabajo se utilizan para ajustar los parámetros del algoritmo.

En estos experimentos presentamos los siguientes datos: el tamaño promedio del biclique balanceado más grande obtenido para el conjunto de datos, Tamaño; el tiempo promedio de ejecución por instancia, Tiempo (s); el porcentaje de desviación promedio respecto de la mejor solución obtenido por instancia, %Desv.; y el número de veces que se encuentra la mejor solución para cada instancia, #Mejores.

### *III-A. Experimentos preliminares*

Los experimentos presentados a continuación fueron diseñados para elegir la mejor variante para el algoritmo propuesto. Se utilizó un grupo reducido de 6 instancias representativas, una por cada combinación posible de parámetros, seleccionadas a partir de las excluidas del experimento final para no sobreajustar el algoritmo. Estas instancias forman parte de las facilitadas por [20] y han sido generadas al azar con los mismos parámetros $n$ (tamaño) y $p$ (probabilidad) que aquellas utilizadas en la comparación de resultados. En el primer experimento se comprueba cuál es el mejor valor de $\alpha$ para el procedimiento constructivo, en el segundo se comparan cuatro variantes del procedimiento *Shake*, y finalmente se examina cuál es el valor más conveniente para la vecindad máxima a explorar. Todos los experimentos se han ejecutado iterativamente hasta alcanzar el limite de tiempo establecido, un número de segundos igual al tamaño $n$ de la instancia.

| $\alpha$ | Tamaño | Tiempo (s) | %Desv. | #Mejores |
|---|---|---|---|---|
| 0.25 | 46.33 | 375.00 | 11.57 | 0 |
| 0.50 | 48.33 | 375.00 | 7.96 | 0 |
| 0.75 | 52.00 | 375.00 | 0.96 | 2 |
| **rnd** | **52.50** | **375.00** | **0.00** | **6** |

Tabla II: Comparación del algoritmo constructivo con distintos valores del parámetro $\alpha$.

En el primer experimento ejecutamos iterativamente únicamente el procedimiento constructivo hasta que se alcanza el límite de tiempo para cada instancia y se guarda la mejor solución alcanzada en dicho tiempo. Se consideran los siguientes valores posibles para el parámetro alfa $\alpha = \{0.25, 0.50, 0.75, random\}$. En particular, el valor "random" significa que en cada iteración se elige un alfa distinto de forma aleatoria. Los resultados en la Tabla II muestran que los mejores resultados se obtienen cuando se elige aleatoriamente el parámetro alfa en cada iteración. En este caso, se obtiene un biclique balanceado de tamaño 52.50 en promedio y, además,

alcanza la mejor solución hallada en este experimento en todas las 6 instancias. Por consiguiente, tanto en los experimentos sucesivos como en el final, se utilizará este valor de alfa para el constructivo inicial que forma parte de nuestra propuesta de *RVNS*.

| Shake | Tamaño | Tiempo (s) | %Desv. | #Mejores |
|---|---|---|---|---|
| RR | 48.83 | 375.00 | 10.21 | 0 |
| RG | **54.33** | 375.00 | 0.00 | 6 |
| GR | 49.50 | 375.00 | 8.95 | 0 |
| GG | 53.17 | 375.00 | 2.37 | 1 |

Tabla III: Comparación del algoritmo *RVNS* con las cuatro variantes del método *Shake*. Se ha fijado la vecindad máxima en $K_{max} = 50\,\%$.

El siguiente experimento está diseñado para elegir la mejor variante del método *Shake*. Se ha fijado la vecindad máxima del algoritmo RVNS en $K_{max} = 50\,\%$ y se comparan las cuatro variantes propuestas para el método *Shake*. En este experimento se observa en la Tabla III que la variante RG, es decir, destrucción aleatoria con reconstrucción voraz, alcanza los mejores resultados. Esta obtiene un biclique balanceado con un tamaño de 54.33 vértices en promedio y, además, también obtiene las 6 mejores soluciones de este experimento.

| $K_{max}$ | Tamaño | Tiempo (s) | %Desv. | #Mejores |
|---|---|---|---|---|
| 10 % | 53.17 | 375.00 | 2.37 | 1 |
| 20 % | 53.33 | 375.00 | 1.85 | 2 |
| 30 % | 53.83 | 375.00 | 0.98 | 3 |
| 40 % | 53.83 | 375.00 | 0.87 | 3 |
| 50 % | **54.33** | 375.00 | 0.00 | 6 |

Tabla IV: Comparación del algoritmo *RVNS* para diferentes valores de la vecindad máxima $K_{max}$.

En el último experimento preliminar queremos evaluar la influencia que tiene el tamaño máximo de la vecindad $k_{max}$ sobre el algoritmo propuesto. Se utiliza la mejor configuración hallada en los experimentos anteriores, es decir, una construcción incial con un alfa aleatorio en cada iteración y el método *Shake* que realiza primero una destrucción aleatoria y luego una reconstrucción voraz (RG); y se ejecuta el algoritmo con diferentes tamaños para la vecindad máxima $K_{max} = \{10\,\%, 20\,\%, 30\,\%, 40\,\%, 50\,\%\}$. En este experimento se puede ver que, en general, expandir el tamaño de la vecindad permite que el algoritmo alcance mejores soluciones, disminuyendo en consecuencia la desviación. Se toma como mejor vecindad $K_{max} = 50\,\%$ con la que se han obtenido los mejores resultados de este experimento.

### *III-B. Experimento final*

En el experimento final se presenta una comparación con el algoritmo memético [20] encontrado en la literatura previa. La comparación se realiza utilizando el mismo conjunto de 30 instancias que se han utilizado en dicho trabajo. El algoritmo *RVNS* se ejecuta por iteraciones hasta que se llega a un tiempo límite, en este caso es un tiempo en segundos igual al triple del tamaño ($n$) de las instancias. En la Tabla V se puede ver que nuestros resultados son muy cercanos al algoritmo previo respecto a la calidad de las soluciones obtenidas, está a menos de un vértice de distancia en promedio, pero tiene un tiempo de ejecución que se aproxima a la mitad del tiempo requerido por el algoritmo memético.

| | Tamaño | Tiempo (s) | %Desv. | #Mejores |
|---|---|---|---|---|
| EA/SM | 55.10 | 2075.11 | 0.04 | 29 |
| RVNS | 54.33 | 1125.00 | 1.71 | 10 |

Tabla V: Comparación del RVNS con el algoritmo evolutivo EA/SM encontrado en la literatura.

## IV. Conclusiones

En este trabajo se ha propuesto el uso de la metodología RVNS para buscar soluciones eficientes para el MBBP. Se ha constatado que en este problema es difícil plantear una búsqueda local efectiva y debido a ello se ha elegido una variante de VNS que explora aleatoriamente una vecindad predefinida. Se ha conseguido plantear un algoritmo competitivo en calidad respecto al actual estado del arte que utiliza aproximadamente la mitad del tiempo que este, por lo que todavía es un enfoque prometedor que permite explorar en un futuro otras técnicas complementarias para mejorar estos resultados.

## Referencias


[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[2] D. S. Johnson, "The np-completeness column: An ongoing guide." *J. Algorithms*, vol. 13, no. 3, pp. 502–524, 1992.

[3] N. Alon, R. A. Duke, H. Lefmann, V. Rödl, and R. Yuster, "The algorithmic aspects of the regularity lemma," *J. Algorithms*, vol. 16, no. 1, pp. 80–109, 1994.

[4] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. Tayur, "On bipartite and multipartite clique problems," *Journal of Algorithms*, vol. 41, no. 2, pp. 388 – 403, 2001.

[5] U. Feige and S. Kogan, "Hardness of approximation of the balanced complete bipartite subgraph problem," Tech. Rep., 2004.

[6] Y. Cheng and G. M. Church, "Biclustering of expression data," in *Proc. of the 8th ISMB*. AAAI Press, 2000, pp. 93–103.

[7] A. Tanay, R. Sharan, and R. Shamir, "Discovering statistically significant biclusters in gene expression data." in *ISMB*, 2002, pp. 136–144.

[8] H. Wang, W. W. 0010, J. Yang, and P. S. Yu, "Clustering by pattern similarity in large data sets." in *SIGMOD Conference*, M. J. Franklin, B. Moon, and A. Ailamaki, Eds. ACM, 2002, pp. 394–405.

[9] M. J. Sanderson, A. C. Driskell, R. H. Ree, O. Eulenstein, and S. Langley, "Obtaining maximal concatenated phylogenetic data sets from large sequence databases," *Molecular Biology and Evolution*, vol. 20, no. 7, pp. 1036–1042, 2003.

[10] E. J. Chesler and M. A. Langston, "Combinatorial genetic regulatory network analysis tools for high throughput transcriptomic data." in *Systems Biology and Regulatory Genomics*, ser. Lecture Notes in Computer Science, E. Eskin, T. Ideker, B. J. Raphael, and C. T. Workman, Eds., vol. 4023. Springer, 2005, pp. 150–165.

[11] E. J. Baker, J. J. Jay, V. M. Philip, Y. Zhang, Z. Li, R. Kirova, M. A. Langston, and E. J. Chesler, "Ontological discovery environment: A system for integrating gene–phenotype associations," *Genomics*, vol. 94, no. 6, pp. 377 – 387, 2009.

[12] R. A. Mushlin, A. Kershenbaum, S. T. Gallagher, and T. R. Rebbeck, "A graph-theoretical approach for pattern discovery in epidemiological research." *IBM Systems Journal*, vol. 46, no. 1, pp. 135–150, 2007.

[13] S. S. Ravi and E. L. Lloyd, "The complexity of near-optimal programmable logic array folding." *SIAM J. Comput.*, vol. 17, no. 4, pp. 696–710, 1988.
[14] M. B. Tahoori, "Application-independent defect tolerance of reconfigurable nanoarchitectures," *JETC*, vol. 2, no. 3, pp. 197–218, 2006.
[15] A. A. Al-Yamani, S. Ramsundar, and D. K. Pradhan, "A defect tolerance scheme for nanotechnology circuits." *IEEE Trans. on Circuits and Systems*, vol. 54-I, no. 11, pp. 2402–2409, 2007.
[16] M. B. Tahoori, "Low-overhead defect tolerance in crossbar nanoarchitectures." *JETC*, vol. 5, no. 2, 2009.
[17] M. Yannakakis, "Node-deletion problems on bipartite graphs." *SIAM J. Comput.*, vol. 10, no. 2, pp. 310–327, 1981.
[18] B. Yuan and B. Li, "A low time complexity defect-tolerance algorithm for nanoelectronic crossbar," in *International Conference on Information Science and Technology*, 2011, pp. 143–148.
[19] ——, "A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 3, p. 25, 2014.
[20] B. Yuan, B. Li, H. Chen, and X. Yao, "A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem," *IEEE Trans. Cybernetics*, vol. 45, no. 5, pp. 1040–1053, 2015.
[21] P. Hansen and N. Mladenović, *Variable Neighborhood Search*. Boston, MA: Springer US, 2014, pp. 313–337.
[22] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.

# Búsqueda de Vecindad Variable para el problema de la agrupación y recogida de pedidos online en almacenes logísticos

Sergio Gil-Borrás, Abraham Duarte, Antonio Alonso-Ayuso y Eduardo G. Pardo

***Resumen*—En este trabajo se presenta la adaptación de un algoritmo de Búsqueda de Vecindad Variable Básica al problema de agrupación y recogida de pedidos *online* en almacenes logísticos. La recogida de pedidos en almacenes se ha convertido, en la actualidad, en una importante tarea operacional como parte de la cadena de suministro. En particular, debido a los costes derivados de esta actividad se ha tornado necesario tratar de optimizar el proceso de recogida, de modo que los productos asociados a los pedidos sean entregados de forma eficiente. Existen diferentes políticas de recogida, entre las que destacan: (i) la recogida de pedidos directa, es decir, cada pedido que llega al almacén es recogido directamente por un trabajador en orden de llegada; y (ii) la recogida por lotes, en la que varios pedidos son agrupados en un mismo lote y asignados a un trabajador, para ser recogidos simultáneamente. En este trabajo se aborda una variante del problema de recogida de pedidos, basada en la política de agrupación en lotes. En concreto, la variante abordada tiene en consideración que los pedidos llegan *online* al almacén, es decir, que no están todos disponibles al comienzo del turno de recogida, sino que van llegando a medida que la jornada de trabajo avanza. La función objetivo consiste en minimizar el tiempo máximo que un pedido permanece en el sistema.**

***Palabras clave*—** Búsqueda de Vecindad Variable, Agrupación en lotes, Heurísticas, *Online Order Batching Problem*.

## I. Introducción

En los últimos años las compras a través de Internet se han popularizado en la sociedad. Derivado de esto, ha surgido la necesidad de mejorar los procesos que se producen dentro de los almacenes logísticos, para llevar a cabo una rápida y eficaz gestión de los pedidos, así como una reducción de costes de procesamiento de los mismos. Esto ha motivado el estudio de diferentes problemas que se producen dentro de este contexto, tan demandado por empresas de distribución. Entre los problemas que surgen, se pueden encontrar, entre otros, problemas relacionados con la recepción de mercancías, su almacenamiento y la recogida de las mismas cuando se recibe un pedido. En este último contexto, es donde se enmarca el presente trabajo. En concreto, se estudia cómo hacer eficiente la recogida de pedidos *online* que llegan al almacén. Existen diferentes variantes de este problema, en función de factores tales como: el número de personas recogiendo los pedidos simultáneamente; si los pedidos tienen o no fecha límite de entrega; e incluso en función de la estructura concreta de los almacenes considerados (varios bloques de estanterías, número de pasillos paralelos, existencia de pasillos diagonales, etc.).

El problema que se aborda aquí es el denominado "*Online Order Batching Problem*" (OOBP). Este problema consiste en determinar el orden en el que se recogen los pedidos que van llegando al almacén de manera constante, así como la ruta necesaria para recogerlos. Para ello, se conforman una serie de lotes de recogida, formados por un conjunto de pedidos, con un tamaño máximo por lote. En este caso, cada uno de los lotes será recogido por un único trabajador. El objetivo consiste en minimizar el tiempo máximo que transcurre desde que un pedido entra en el almacén hasta que este ha sido procesado para su envío. Este tiempo es comúnmente denominado "*makespan*" en la literatura asociada a los procesos industriales. El tiempo en el que un pedido está en el almacén es, en realidad, la suma de dos tiempos. Por un lado, el tiempo de procesado del lote (compuesto a su vez por lo que tarda en generarse el lote y por el tiempo de espera hasta que este lote pase a ser recogido). Y, por otro lado, el tiempo que se tarda en recoger todo el lote por los pasillos del almacén y llevarse a la zona de entrega. Para este último proceso, la recogida de artículos, se emplean algoritmos que establecen una ruta dentro del almacén. En concreto, en este trabajo se han repasado tres de los algoritmos heurísticos más conocidos en la literatura, como son S-Shape [1], Largest-Gap [2] y Combined [3] [4], que se repasarán detalladamente en la Sección III.

En este trabajo se han empleado almacenes con dos pasillos trasversales, uno superior y otro inferior, y un conjunto variable de pasillos paralelos que los unen, donde se colocan los productos en columnas de igual tamaño, a ambos lados de cada pasillo paralelo. El número de pasillos y las dimensiones de la sala, así como de las estanterías, varía según la instancia seleccionada. En la Fig. 1 se puede ver un ejemplo de este tipo de almacén con dos pasillos transversales y cinco pasillos paralelos. Cada pasillo paralelo tiene nueve posiciones de recogida a cada uno de los lados, representadas por pequeños

Sergio Gil-Borrás, Universidad Politécnica de Madrid. Ctra. Valencia, Km. 7, 28031, Madrid, España, (e-mail: sergio.gil.borras@alumnos.upm.es).
Abraham Duarte, Universidad Rey Juan Carlos. C/Tulipán s/n, 28933, Madrid, España, (e-mail: abraham.duarte@urjc.es).
Antonio Alonso-Ayuso, Universidad Rey Juan Carlos. C/Tulipán s/n, 28933, Madrid, España, (e-mail: antonio.alonso@urjc.es).
Eduardo G. Pardo, Universidad Politécnica de Madrid. Ctra. Valencia, Km. 7, 28031, Madrid, España, (e-mail: eduardo.pardo@upm.es).

cuadrados. El almacén mostrado tiene, por lo tanto, un total de 90 posiciones de recogida. Se considera que en cada posición de recogida se almacena únicamente un posible producto. No se consideran, por lo tanto, múltiples alturas en cada posición de recogida. En el pasillo transversal inferior se sitúa, además, el depósito, que será el punto de partida para las rutas de recogida, así como el punto de entrega de los productos recogidos en cada ruta. Este depósito aparece posicionado en el centro del pasillo, si bien existen instancias en las que el depósito se encuentra en uno de los extremos del mismo.

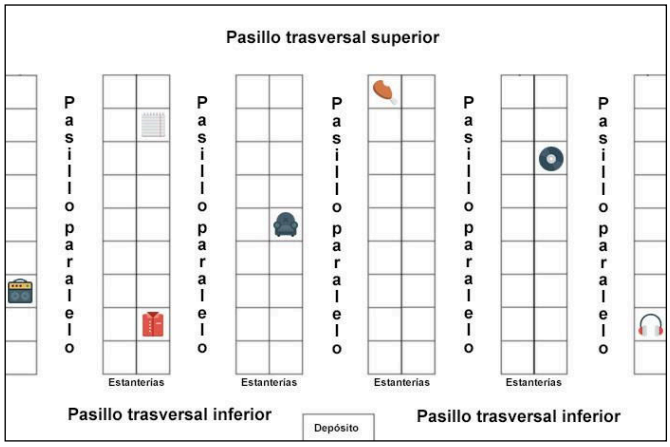


**Fig. 1 – Estructura del almacén**

El resto del artículo está estructurado de la siguiente forma: en la Sección II se repasa el estado del arte del problema abordado. En la Sección III se describen los algoritmos de enrutamiento más relevantes de la literatura. En la Sección IV se presentan los algoritmos heurísticos de agrupación propuestos, basados en la metodología de Búsqueda de Vecindad Variable. En la Sección V se muestran algunos resultados preliminares obtenidos y los conjuntos de instancias empleados. Por último, en la Sección VI se exponen las conclusiones de este trabajo.

## II. Estado del arte

Entre los últimos artículos publicados que se recogen en la literatura, relativos a esta familia de problemas de optimización, destaca el artículo de S. Henn en 2012 [5], donde los autores proponen un algoritmo basado en la metaheurística "*Iterated Local Search*" (ILS) [6] para abordar el problema. Este algoritmo está basado en el principio de búsqueda local y, en este artículo, se compara el algoritmo propuesto con las técnicas heurísticas clásicas de "*First Come, First Served*" (FCFS) y el método de "Clarke and Wright" (C&W) [7].

Más recientemente, en el año 2015, Ricardo Pérez-Rodríguez publicó un artículo [8] resolviendo también esta variante del problema. En este artículo, los autores proponen un algoritmo basado en la estimación continua de la distribución (denominado OBCEDA por sus siglas en inglés) que a su vez está basado en los bien conocidos Algoritmos de la Estimación de la Distribución (EDA) [9]. En este trabajo, se compara la propuesta realizada con un algoritmo genético y con un algoritmo basado en la metaheurística "Tabu-Search" [10]. En este caso también se usan las mismas instancias que usa Sebastián Henn en su artículo del 2012, si bien no se realiza una comparación directa entre ambos trabajos.

Sobre otras variantes del problema, íntimamente relacionadas, destaca la publicación en estos últimos años del trabajo presentado por Rubrico en [11] donde la recogida de los pedidos se hace por múltiples trabajadores (habitualmente denominados *pickers*) en lugar de únicamente por uno, tal y como se realiza en el problema tratado en este documento. En este caso, Rubrico propone dos variaciones de un "*Incremental static scheduling scheme algorithm*", que los autores denominan "*Steepest descent insertion*" y "*On multistage rescheduling*". En este caso, los autores utilizan un conjunto de instancias generadas por ellos mismos, y se compara con el "G66 *Listing algorithm*", el cual es una extensión del algoritmo "*Graham's list scheduling algorithm*" [12]. También se realiza una comparación con el denominado "*Aspnes'93 algorithm*" [13].

Otra variante interesante del problema, publicada en 2016 [14], consiste en añadir una restricción de fecha/hora límite de entrega a cada pedido y, además, la recogida de los pedidos también se realiza con múltiples *pickers*. El algoritmo que los autores proponen es un algoritmo basado en reglas para preprocesar los pedidos antes de usar el algoritmo de agrupamiento de pedidos en lotes. Con esta estrategia, los autores consiguieron mejorar los resultados. En su caso, los algoritmos que usan para compararse son los "*Seed algorithms*" [15] y el "*Recalculation saving algorithms*" [16]. Las instancias utilizadas en este artículo siguen siendo las mismas instancias que introdujo Sebastián Henn en [5].

## III. Enrutamiento

Los algoritmos de enrutamiento son los encargados de decidir el camino que seguirá el trabajador por el almacén para recoger todos los pedidos incluidos en un mismo lote. El camino comenzará en el depósito que está marcado en las siguientes figuras como "D" (del término en inglés *depot*). El camino finaliza en el mismo depósito, una vez que se han recogido todos los elementos de un mismo lote. Al igual que otros problemas de enrutamiento, este problema de optimización ha sido ampliamente trabajado con anterioridad, existiendo en la literatura algoritmos heurísticos, metaheurísticos y exactos para esta variante concreta. No obstante, los algoritmos de enrutamiento elegidos para este trabajo, y que se repasan a continuación, son algoritmos heurísticos. Estos algoritmos, aunque generan soluciones con recorridos ligeramente más largos que los algoritmos exactos o que los algoritmos metaheurísticos, son más rápidos a la hora de computar una solución. Además, en términos generales, el recorrido que ofrecen suele ser de alta calidad y más comprensible por parte de los empleados del almacén.

En la Fig. 2 se puede ver una ilustración de la estrategia de enrutamiento denominada S-Shape, también conocida en algunas referencias como estrategia Trasversal. Esta estrategia consiste en recorrer completamente todos los pasillos paralelos que contengan ítems que recoger. A excepción del último

pasillo con ítems, que dependerá de la posición del recogedor cuando deba entrar en dicho pasillo. En el caso que se acceda a él desde el pasillo trasversal inferior, entonces se recorrerá el pasillo únicamente hasta llegar al último ítem a recoger del pasillo y luego, el recogedor dará media vuelta para recorrer ese mismo pasillo de vuelta hasta llegar de nuevo al pasillo trasversal inferior y, posteriormente, volver al depósito para entregar las mercancías. No obstante, en el caso de que el *picker* se encuentre en el pasillo transversal superior, en el momento de entrar en el último pasillo, entonces este se recorrerá completamente, volviendo al pasillo transversal inferior, que contiene el depósito, para la entrega de mercancías. En la Fig. 2, están representados con color compacto negro los puntos de recogida de los ítems de un mismo lote.

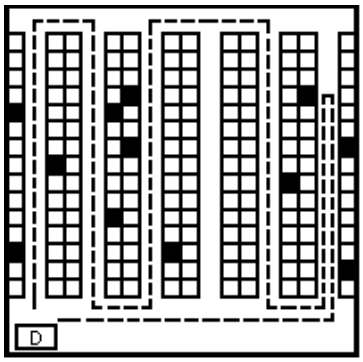


**Fig. 2 – Recorrido basado en S-shape[1]**

Análogamente, en la Fig. 2, se puede ver la estrategia de enrutamiento denominada Largest-Gap. En esta estrategia solo se recorrerán completamente el primer y último pasillo paralelo con ítems. El resto de pasillos paralelos solo se recorrerán, en el caso que tenga ítems a recoger. Además, el tramo de pasillo a recorrer será el comprendido entre el pasillo transversal del que se parta (inferior o superior según el caso) y el denominado "*largest-gap*". Este término se utiliza para referirse al espacio más grande que hay entre cada par de ítems consecutivos en un mismo pasillo. Una vez alcanzado el *largest-gap*, el operario encargado de la recogida debe dar media vuelta sobre sus pasos y volver al pasillo transversal del que partía. En el caso de que quedasen ítems por recoger (que se encuentren más allá del *largest-gap*) estos serán recogidos en la ruta de vuelta desde el pasillo transversal contrario. Por lo tanto, desde el pasillo trasversal superior se accederá solo a recoger los ítems que se encuentran en la zona situada antes del *largest-gap*, en la mitad superior de los pasillos paralelos. De manera similar, desde el pasillo trasversal inferior solo se recogerán los ítems que se encuentran entre el pasillo transversal inferior y la última posición de un ítem dentro del pasillo paralelo, antes del denominado *largest-gap*.

Por último, en la Fig. 3, se puede ver la estrategia denominada Combined. En esta estrategia se combinan las técnicas de las dos estrategias anteriormente explicadas.

1. La Fig. 2, Fig. 3 y Fig. 4 han sido obtenidas de la página web de la Universidad de Rotterdam: https://www.erim.eur.nl/material-handling-forum/research-education/tools/calc-order-picking-time/what-to-do/routing-strategies/, el día 13 Julio de 2018.

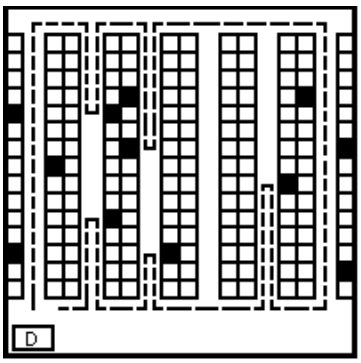


**Fig. 3 – Recorrido basado en Largest Gap[1]**

En la estrategia Combined, para recorrer cada pasillo paralelo se evalúa qué recorrido es más corto de hacer para recoger todos los ítems de un mismo pasillo, si hacer una estrategia S-Shape o si hacer una estrategia Largest-Gap. De este modo, a medida que se va avanzando se toma la mejor decisión para cada pasillo. De manera global, se puede afirmar que los resultados obtenidos con esta estrategia son mejores que aquellos obtenidos aplicando las anteriores estrategias expuestas de manera independiente. En la Fig. 4, se presenta un ejemplo de un recorrido empleando la estrategia Combined, para la misma instancia representada en la Fig. 2 y en la Fig. 3.

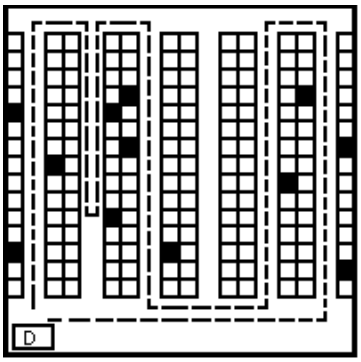


**Fig. 4 – Recorrido basado en Combined[1]**

## IV. Agrupamiento

Para abordar el problema del agrupamiento de pedidos en lotes, se propone la combinación de dos algoritmos: un algoritmo constructivo voraz y una metaheurística, como método de mejora de la solución inicial formada por el constructivo.

El método constructivo ordena los pedidos en función del orden de llegada y parte de un lote vacío. A continuación, toma el primer pedido según el orden establecido y trata de añadirlo al último lote disponible (en el caso inicial, el lote vacío). Cuando este lote no tiene espacio suficiente crea un nuevo lote. Este proceso se repite hasta que todos los pedidos han sido asignados a un lote. En lo sucesivo se denominará a este método constructivo como *First-Come-First-Serve* (FCFS). Se propone también una variante de este método consistente en tratar de añadir el siguiente pedido en el orden establecido a cada uno de los lotes previos (no solo en el último) intentando, de ese modo, completar el espacio vacío en los lotes. A este método se le denominará en lo sucesivo como *First-Come-First-Serve* Completo (FCFS-Completo).

Además de los métodos constructivos anteriormente descritos, se propone la utilización de un método basado en la metaheurística Búsqueda de Vecindad Variable (VNS) [17]. VNS es un algoritmo metaheurístico que ayuda a resolver problemas complejos tanto de optimización combinatoria como de optimización global. En concreto, partiendo de una solución inicial, intenta mejorarla visitando distintos vecindarios. Mediante la utilización de una búsqueda local explora, en cada vecindario, las posibles soluciones prometedoras, obteniendo un óptimo local en cada uno de ellos.

Existen diferentes variantes dentro de la metodología VNS. En este trabajo, se ha escogido la variante denominada "Búsqueda de Vecindad Variable Básica" (BVNS, del inglés *Basic Variable Neighborhood Search*). El algoritmo BVNS se basa en tres métodos: un método consistente en una búsqueda local para alcanzar un óptimo local en un vecindario dado; un método de perturbación de la solución y un método que determina si debe o no cambiarse la vecindad a explorar. En Algoritmo I se presenta el pseudocódigo del método BVNS.

ALGORITMO I
FUNCIÓN BVNS

```
Función BVNS (x, k_max, t_max);
 1: repeat
 2:    k ← 1;
 3:    repeat
 4:       x' ←Shake(x, k) /* Perturbación */;
 5:       x" ← PrimeraMejora (x') /* Búsqueda local */;
 6:       x← CambioVecindad (x, x", k)
          /* Cambio de vecindad */;
 7:    until k = k_max ;
 8:    t ←CpuTime()
 9: until t > t_max ;
```

Cada iteración del método BVNS repite la ejecución de los tres métodos anteriormente mencionados, mientras que no se alcance el valor máximo de *k* (denotado por el parámetro *k_max*), que determina el máximo número de vecindades a explorar. Una vez alcanzado el valor de *k_max*, se permite realizar una nueva exploración completa si el tiempo máximo de ejecución (*t_max*) no se ha superado.

El método de búsqueda local propuesto dentro del algoritmo BVNS se presenta en el pseudocódigo de la función PrimeraMejora (ver Algoritmo II). Este método, dada una solución de partida *x* explora las soluciones en el vecindario de *x,* denotado como N(*x*), hasta que encuentra una solución de mejor calidad, en cuyo caso, utiliza esta nueva solución como punto de partida para la siguiente iteración del algoritmo. El procedimiento se repite hasta que toda la vecindad ha sido explorada. La vecindad, en el caso de esta búsqueda local, está definida por movimientos de inserción factibles, es decir, se toma un pedido de un lote y se trata de insertar en cada uno de los otros lotes disponibles, realizando la inserción únicamente si el pedido cabe en el lote. En concreto, se realizará el primer movimiento factible que suponga una mejora en valor de la función objetivo de la nueva solución vecina. El procedimiento se repite hasta que ningún movimiento mejora la solución.

ALGORITMO II
FUNCIÓN PRIMERAMEJORA

```
Función PrimeraMejora(x)
 1: repeat
 2:    x' ← x;
 3:    i←0;
 4:    repeat
 5:       i←i+1
 6:       x←argmin{ f (x), f (x^i)}, x^i  ∈ N(x)
 7:    until ( f (x) < f (x^i) or i = |N(x)|)
 8: until ( f (x) ≥ f (x'))
 9: return x
```

El método encargado de realizar una perturbación de la solución (denominado Shake) se utiliza para escapar del óptimo local actual mediante el movimiento a una nueva vecindad. En este caso, el método de perturbación se basa en movimientos de intercambio de pedidos. Es decir, se toman dos pedidos de dos lotes distintos, al azar, y se intercambian, siempre y cuando al realizar el intercambio la solución continúe siendo factible (no se sobrepase la capacidad de los lotes involucrados). Este mecanismo se aplica tantas veces como indique la variable *k*. Por último, se incluye un mecanismo para determinar si se debe explorar una vecindad de mayor tamaño, o bien se deben explorar de nuevo vecindades próximas a la que se acaba de explorar. Este mecanismo se implementa en el método CambioVecindad, recogido en el pseudocódigo de Algoritmo III.

ALGORITMO III
FUNCIÓN CAMBIOVECINDAD

```
Función CambioVecindad (x, x', k)
 1: if f (x') < f(x) then
 2:    x ← x' /* Realizar un movimiento */
 3:    k ← 1 /* Volver a vecindad inicial */
 4: else
 5:    k ← k+1 /* Siguiente vecindad */
```

Tal y como se ha indicado anteriormente, los pedidos no están disponibles, en su totalidad, al comienzo de la jornada de trabajo. Durante la ejecución del algoritmo los pedidos entran en el sistema siguiendo una distribución uniforme. El algoritmo BVNS se ejecuta durante un tiempo *t_max*, considerando, en cada iteración, los pedidos que hayan llegado hasta ese momento. Una vez alcanzado el valor de *t_max*, el método comienza de nuevo, realizando una nueva construcción y añadiendo los nuevos pedidos que hayan llegado durante el tiempo de la última ejecución de BVNS. En cada ejecución del algoritmo se devuelve la mejor solución encontrada. Cuando el trabajador encargado de la recogida ha terminado con un último lote, se le asigna, de la última solución disponible, uno de los lotes aún no recogidos. Para determinar qué lote será el siguiente en recogerse se emplea un algoritmo de selección. En concreto, este algoritmo, elegirá el lote que tiene un menor valor del cociente entre el tiempo estimado que tarda en recogerse el lote y el peso del mismo.

## V. Resultados

En esta sección se describe el conjunto de instancias utilizadas para la validación de los algoritmos propuestos y se recogen, además, los resultados por instancia, de entre el conjunto de instancias seleccionadas.

### A. *Instancias*

Las instancias usadas para hacer esta investigación son las publicadas por Albareda-Sambola et al. [18], en 2009. Las instancias constan de 4 tipos de almacenes rectangulares con el depósito colocado en la esquina inferior izquierda o en el centro inferior del almacén. La distribución de los ítems en los almacenes de las instancias puede ser de dos tipos: ABC y aleatoria. Las distribuciones de tipo ABC dividen los productos en tres categorías, situando los productos más demandados en la categoría A, y los menos demandados en las categorías B y C respectivamente. Una vez clasificados los productos, aquellos que caen dentro de la categoría A se sitúan más próximos al *depot,* los pertenecientes a la categoría B, se sitúan a continuación y, por último, se almacenan los productos pertenecientes a la categoría C. Además, cada instancia tiene distinto número de pedidos (50, 100, 150, 200 y 250). Las instancias empleadas para el problema son una pequeña selección de 10 instancias de entre todas las publicadas en [18]. Es importante destacar que, en este artículo no se emplean las instancias de tamaño 50 pedidos, ya que no generan la congestión suficiente para que sistema tenga sentido. Al no producirse congestión, para este tipo de instancias basta con seguir una política de recogida *First-Come-First-Serve* (FCFS). Para hacer la experimentación presentada en este documento, las instancias, con origen en la versión *offline* del problema, han tenido que ser adaptadas a instancias aptas para la recepción de pedidos *online*, generando un sistema de entrega de paquetes que distribuya la llegada de los pedidos. En concreto, el proceso encargado de entregar los paquetes al sistema sigue una distribución uniforme y los pedidos son entregados en un horizonte temporal de 4 horas.

### B. *Resultados*

Por último, se ha comparado la propuesta algorítmica con los resultados obtenidos por el algoritmo del estado del arte "*Iterated Local Search"* propuesto por Henn [5] en su artículo de 2012. La comparación se ha realizado entre dicho algoritmo y dos variantes del BVNS propuesto, donde se combina un BVNS con cada uno de los dos tipos de construcciones iniciales. Un FCFS estándar y un FCFS-Completo, donde siempre se intenta completar la capacidad del lote a la llegada de cada pedido, con independencia de que dicho lote se hubiera dado por lleno en la iteración anterior. El BVNS se ejecuta en iteraciones de 60 segundos, permitiendo a los algoritmos constructivos, reconstruir la solución con los nuevos pedidos llegados al sistema antes de continuar con la búsqueda. Como selector, para determinar el siguiente lote a ser recogido, se emplea el criterio del lote que tiene un menor valor en el cociente obtenido al dividir el tiempo que tarda en recogerse el lote entre el peso del mismo.

En la Tabla I y en la Tabla II de resultados, respectivamente, se observa como la diferencia entre los resultados arrojados por los algoritmos es muy pequeña. En concreto, se reportan en la Tabla I los valores de función objetivo por instancia, obtenidos por cada método comparado y, en la Tabla II, la desviación obtenida respecto a la mejor solución del experimento. En este caso, ILS aparece como el mejor método, siendo capaz de obtener el mayor número de mejores soluciones y la menor desviación. Los mejores resultados, por cada instancia, están resaltados en negrita en las tablas.

TABLA I
Comparación Con El Estado Del Arte
Tiempo Máximo De Los Pedidos En El Sistema

| *Instancia* | *ILS* | *BVNS - FCFS* | *BVNS – FCFS Completo* |
|---|---|---|---|
| **A1_100_000** | 1721232 | 1666627 | **1578027** |
| **A1_100_030** | **1215180** | 2081531 | 2621530 |
| **A1_100_060** | **1515669** | 1945930 | 1885316 |
| **A1_100_090** | **1099946** | 1409978 | 1443535 |
| **A1_150_000** | **2876411** | 4215847 | 3375709 |
| **A1_150_030** | **1081205** | 1426639 | 1645529 |
| **A1_150_060** | **2651687** | 2855052 | 3770849 |
| **A1_150_090** | **1232818** | 1437467 | 1422996 |
| **A1_200_000** | 5578399 | 2273878 | **1479272** |

TABLA II
Comparación Con El Estado Del Arte
Desviación Respecto a la Mejor Solución Del Experimento

| *Instancia* | *ILS* | *BVNS - FCFS* | *BVNS – FCFS Completo* |
|---|---|---|---|
| **A1_100_000** | 9,1% | 5,6% | **0,0%** |
| **A1_100_030** | **0,0%** | 71,3% | 115,7% |
| **A1_100_060** | **0,0%** | 28,4% | 24,4% |
| **A1_100_090** | **0,0%** | 28,2% | 31,2% |
| **A1_150_000** | **0,0%** | 46,6% | 17,4% |
| **A1_150_030** | **0,0%** | 31,9% | 52,2% |
| **A1_150_060** | **0,0%** | 7,7% | 42,2% |
| **A1_150_090** | **0,0%** | 16,6% | 15,4% |
| **A1_200_000** | 277,1% | 53,7% | **0,0%** |

En la Tabla III, se muestran los resultados obtenidos de manera resumida. En concreto, se presenta el promedio de desviación de las 10 instancias consideradas y la suma del número de veces que un algoritmo ha encontrado el mejor valor.

TABLA III
Comparación Con El Estado Del Arte
Resultados Agrupados

| *Instancia* | *ILS* | *BVNS - FCFS* | *BVNS – FCFS Completo* |
|---|---|---|---|
| **Promedio** | **31,8%** | 32,2% | 33,2% |
| **Número de mejores** | **7** | 0 | 2 |

## VI. Conclusiones

En este trabajo se ha estudiado el problema de agrupación y recogida de pedidos *online* en almacenes logísticos. Este problema de optimización, consiste, por un lado, en la agrupación en lotes de los pedidos que llegan al sistema y, por otro, en el diseño de rutas eficientes para recoger cada lote. La variante abordada considera que únicamente se dispone de un

trabajador y que los pedidos llegan al almacén de manera *online*, es decir, no todos los pedidos están disponibles al comienzo de la jornada de trabajo, sino que van llegando a medida que avanza el proceso de recogida.

Para abordar el problema se ha repasado el estado del arte de la variante *online* del mismo y se han descrito los métodos de *routing* más comunes en la literatura. A continuación, se ha propuesto un algoritmo basado en la metodología de Búsqueda de Vecindad Variable, en concreto, su versión Básica. Se ha implementado también el algoritmo más destacado del estado del arte. Para validar el algoritmo propuesto se ha adaptado un subconjunto de las instancias presentes en la literatura para la versión *offline* del problema, estableciendo una distribución uniforme para la llegada de los pedidos al almacén, y se han ejecutado ambos algoritmos en la misma máquina.

Pese a ser aún un trabajo en desarrollo, los resultados obtenidos son prometedores, siendo el algoritmo propuesto competitivo con la mejor variante en el estado del arte de la actualidad. Si bien, el método del estado del arte, propuesto por Henn en 2012, es el mejor, en promedio, de los algoritmos estudiados.

Como trabajos futuros, en primer lugar, se plantea la posibilidad de mejorar el algoritmo constructivo voraz utilizado como punto de partida inicial. Además, parece recomendable probar diferentes vecindades a la propuesta en este trabajo, de modo que se puedan conformar nuevas búsquedas locales para embeber dentro del esquema de Búsqueda de Vecindad Variable Básica. De manera similar, varias búsquedas locales pueden ser utilizadas en un esquema de Búsqueda de Vecindad Variable General. Por otro lado, las estrategias más prometedoras propuestas, pueden ser empleadas también para resolver otras variantes del problema tales como: disponer de más de un trabajador para recoger los pedidos en el almacén; o establecer una fecha/hora límite de entrega de cada pedido que se debe cumplir para evitar penalizaciones.

## Agradecimientos

Esta investigación ha sido parcialmente financiada por el Ministerio de Economía y Competitividad, Referencias "TIN2015-65460-C2-2-P y MTM2015-63710-P".

## Referencias

[1] R. Hall., «Distance approximation for routing manual pickers in a warehouse», *IIE Transactions,* nº 25, pp. 77–87, 1993.

[2] C. Petersen, «Routeing and storage policy interaction in order picking operations», *Decision Science institute Proceedings,* nº 31, pp. 1614 - 1616, 1995.

[3] R. De Koster, E. Van der Poort y K. Roodbergen, «When to apply optimal or heuristic routing of orderpickers, in: Advances in Distribution Logistics», *Springer* , pp. 375-401, 1998.

[4] K. Roodbergen y C. Petersen, «How to improve order picking efficiency with routing and storage policie», *Progress in Material Handling Practice,* pp. 107–124, 1999.

[5] S. Henn, «Algorithms for On-line Order Batching in an Order- Picking Warehouse», *Computers &OperationsResearch,* nº 39, pp. 2549–2563, 2012.

[6] H. R. Lourenco, O. C. Martin y T. Stutzle, «Iterated Local Search: framework and applications», *Handbook on MetaHeuristics,* pp. 363-397, 2001.

[7] C. G. y W. J., «Scheduling of vehicles from a central depot to a number of delivery points», *Operations Research,* vol. 4, nº 12, pp. 568-581, 1964.

[8] R. Pérez-Rodríguez, A. Hernández-Aguirre y S. Jöns, «A continuous estimation of distribution algorithm for the online order-batching problem», *Int J Adv Manuf Technol,* pp. 569–588, 2015.

[9] P. Larrañaga y J. A. Lozano, Estimation of Distribution Algorithms a New Tool for Evolutionary Computation, Boston, MA: Springer, 2002.

[10] F. Glover, «Future paths for integer programming and links to artificial intelligence», *Computers and Operations Research,* vol. 13, nº 5, pp. 533-549, 1986.

[11] J. Rubrico, T. Higashi, H. Tamura y J. Ota, «Online rescheduling of multiple picking agents for warehouse management», *Robotics and Computer Integrated Manufacturing,* nº 27, pp. 62–71, 2011.

[12] R. Graham, «Bounds for certain multi-processing anomalies», *Bell Syst Tech,* pp. 1563–1581, 1966.

[13] J. Aspnes, Y. Azar, F. A., S. Plotkin y O. Waarts, «On-line load balancing with applications to machine scheduling and virtual circuit routing», *Proceedings of the 25th A C Mannual symposium on the theory of computing,* pp. 623–631, 1993.

[14] J. Zhang, X. Wanga y K. Huang, «Integrated on-line scheduling of order batching and delivery under B2C e-commerce», *Computers & Industrial Engineering,* nº 94, pp. 280–289, 2016.

[15] Y. C. Ho y Y. Y. Tseng, «A study on order-batching methods of order-picking in a distribution centre with two cross-aisles», *International Journal of Production Research,* vol. 44, pp. 3391-3417, 2006.

[16] R. De Koster, E. S. Van der Poort y M. Wolters, «Efficient order batching methods in warehouses», *International Journal of Production Research,* vol. 37, pp. 1479–1504, 1999.

[17] N. Mladenovic y P. Hansen, «Variable neighborhood search», *Computers and Operations Research,* nº 24, pp. 1097–1100, 1997.

[18] M. Albareda-Sambola, A. Alonso-Ayuso y E. Molina, «Variable neighborhood search for order batching in a warehouse», *Asia-Pacific Journal of Operational Research,* vol. 26, nº5, pp. 655-683, 2009.

# Una metaheurística paralela para grandes problemas de optimización dinámica entera mixta, con aplicaciones en la biología computacional

David R Penas
*Departamento de Estadística, Análisis matemático y Optimización, y Instituto de Matemáticas (IMAT)*
Universidade de Santiago de Compostela (USC), España
david.rodriguez.penas@usc.es

David Henriques
*Instituto de Investigaciones Marinas (IIM) Consejo Superior de Investigaciones Científicas (CSIC)*
Vigo, España

Patricia González
*Grupo de Arquitectura de Computadores (GAC)*
Universidade da Coruña (UDC)
Coruña, España

Ramón Doallo
*Grupo de Arquitectura de Computadores (GAC)*
Universidade da Coruña (UDC)
Coruña, España

Julio Saez-Rodriguez
*Joint Research Centre for Computational Biomedicine (JCR-Combine)*
RWTH Aachen University
Aquisgrán, Alemania

Julio R Banga
*Instituto de Investigaciones Marinas (IIM) Consejo Superior de Investigaciones Científicas (CSIC)*
Vigo, España

***Abstract*—Esto es un resumen de nuestro artículo publicado en PLoS ONE [1] para su presentación en la Multiconferencia CAEPIA'18 Key Works.**

***Index Terms*—Optimización MIDO-MINLP, programación paralela, metaheurística cooperativa, Scatter Search.**

## I. Introducción

Muchos problemas clave dentro de la bioinformática, la biología computacional o la biología de sistemas [2], se pueden resolver a través de formulaciones basadas en la optimización matemática. Un ejemplo es el desarrollo de modelos matemáticos dinámicos, donde se mapean comportamientos que varían en el tiempo de determinados procesos biológicos, como por ejemplo el de rutas metabólicas, o la regulación de determinados genes.

En este trabajo nos hemos centrado en los conocidos en bioinformática como problemas de ingeniería inversa [3], los cuales consisten en reconstruir la estructura de una red celular a partir de sus datos experimentales. Este proceso puede ser formulado como un problema de optimización dinámica entera mixta (MIDO), debido a las variables dependientes en el tiempo y al uso de loggic-ODEs [4] para mapear ciertos comportamientos en el modelo, y donde además parte de las variables de decisión son discretas (binarias o enteras). Adicionalmente, este tipo de optimización también pertenece a la programación no linear entera mixta (MINLP), dando lugar a problemas extremadamente difíciles de resolver debido a que presentan no linealidad, no convexidad, mal acondicionamiento y multimodalidad.

Las metaheurísticas son una buena alternativa a tener en cuenta en este contexto, ya que generalmente alcanzan una región cercana a la solución global en un tiempo de ejecución aceptable. Sin embargo, cuando estos problemas tienen un tamaño o complejidad considerable, el costo computacional puede llegar a ser muy alto [5], [6].

Por ello, en este trabajo se ha presentado saCeSS2 [7]: una metaheurística cooperativa y auto-adaptativa con capacidad para resolver problemas MIDO-MINLP de manera satisfactoria en un tiempo de cálculo razonable.

## II. Métodos

Una de las contribuciones de este trabajo es la actualización de saCeSS [8], la cual es una herramienta distribuida, cooperativa y auto-adaptativa del método de búsqueda dispersa, en concreto del enhanced Scatter Search (eSS) [9], [10]. Así, saCeSS es un optimizador global orientado a trabajar con problemas NLP que implementa las siguientes funcionalidades:

- *Paralelización a grano fino combinando el modelo maestro-esclavo con un modelo en islas*: cada esclavo/isla ejecuta de manera aislada una búsqueda con eSS, y cuando se encuentra una solución prometedora, se envía al maestro, el cual es el encargado de gestionar las comunicaciones entre las diferentes islas, evitando así un exceso de compartición de resultados que podría degradar la diversidad del método.
- *Cooperación de soluciones basado en la calidad de la solución*: el intercambio de información entre islas solo se realiza cuando una isla encuentra una solución lo suficientemente buena.
- *Comunicaciones asíncronas*: el protocolo de comunicaciones diseñado es asíncrono, para evitar retardos y

esperas, haciendo que la búsqueda de soluciones de cada isla nunca se pare.

- *Esquema autoadaptativo*: se ha implementado un mecanismo de auto-configuración de los parámetros de la metaheurística. Cada isla comenzará el proceso de optimización con unas opciones de configuración diferentes. A medida que la búsqueda avance, el maestro permitirá que los esclavos que hayan aportado menos soluciones a la búsqueda global adquieran la configuración de aquellos que hayan sido más exitosos.

Por lo tanto, la principal aportación de este trabajo es la presentación de un nuevo método, saCeSS2, que resulta de las modificaciones del método saCeSS original con el principal objetivo de resolver problemas MIDO-MINLP. Esta extensión de funcionalidades ha seguido tres direcciones principales:

1) *Inclusión de un solucionador local especializado en la resolución de problemas de optimización entera mixta*: se ha combinado el eSS con un método de programación cuadrática, llamado Mixed-Integer Sequential Quadratic Programming (MISQP) [11], en cada isla.
2) *Modificación del mecanismo de auto-adaptación con el objetivo de evitar estancamiento prematuro de la convergencia*: en este tipo de problemas de optimización, encontrar nuevas soluciones candidatas a ser compartidas no es fácil, por lo tanto se han relajado las condiciones de adaptación del método.
3) *Implementación de nuevos mecanismo para mantener la diversidad durante el proceso de cooperación*: se ha observado que en este tipo de problemas hay que ser muy cautos durante la compartición de soluciones, ya que una cooperación agresiva daña muy rápidamente la diversidad de la isla. Por eso, cuando se realice un cambio en los parámetros de configuración de una isla que no ha contribuido lo suficiente, se produce un reinicio aleatorio de la mayoría de soluciones de su población.

Otra de las contribuciones de este trabajo ha sido la extensión y generalización de la formulación considerada por Henriques et al [4] al adoptar un enfoque genérico del problema del control óptimo entero mixto, pero sin utilizar relajaciones o transformaciones del problema original durante su resolución.

## III. Resultados

Se ha probado el método propuesto en tres casos de estudio de ingeniería inversa muy complejos relacionados con la señalización celular: (1) un problema que considera una vía de señalización sintética, con 84 variables de decisión continuas y 34 binarias; (2) un modelo dinámico de señalización en cáncer de hígado que utiliza datos de alto rendimiento, con 135 variables de decisión continuas y 109 binarias; y (3) un problema extremadamente difícil relacionado con el cáncer de mama, con 690 variables de decisión continuas y 138 binarias.

Se reportaron resultados computacionales obtenidos en diferentes infraestructuras, incluyendo un clúster local, un gran superordenador y una plataforma de Cloud pública. Curiosamente, los resultados muestran cómo la cooperación de búsquedas paralelas individuales modifica las propiedades sistémicas del algoritmo secuencial, logrando aceleraciones superlineales en comparación con una búsqueda individual (por ejemplo, aceleraciones de 15 con 10 núcleos) y mejorando significativamente (por encima del 60%) el rendimiento con respecto a un esquema paralelo no cooperativo. La escalabilidad del método también es buena (las pruebas se realizaron usando hasta 300 núcleos). Adicionalmente, se ha obtenido que la utilización de saCeSS2 en una infraestructura pública de Cloud tiene un buen rendimiento en configuraciones que usan 10 núcleos, pero el rendimiento se degrada considerablemente cuando se trata de utilizar más procesadores.

## IV. Conclusiones

Los resultados mostrados en este trabajo demuestran que saCeSS2 puede ser útil en los problemas de ingeniería inversa en modelos dinámicos de complejas rutas biológicas. Además, los éxitos obtenidos mediante esta herramienta pueden ser fácilmente exportados a otros casos de estudios que también generen problemas de optimización MIDO-MINLP a gran escala, como por ejemplo la ingeniería metabólica, la biología sintética, o la programación de fármacos.

Por otro lado, desde el punto de vista computacional, también se han demostrado que las estrategias de virtualición y computación en Cloud, pueden ser una buena alternativa, aunque presentan unas bien conocidas limitaciones debido principalmente a sobrecargas en la compartición de recursos.

## References


[1] D.R. Penas, D. Henríques, P. Gónzalez, R. Doallo, J. Saez-Rodriguez and J.R. Banga, "A parallel metaheuristic for large mixed-integer dynamic optimization problems, with applications in computational biology" PLoS ONE, 12(8): e0182186, 2017.

[2] J.R. Banga, "Optimization in computational systems biology" BMC Systems Biology, 2 (1), 47, 2008.

[3] A.F. Villaverde and J.R. Banga, "Reverse engineering and identification in systems biology: strategies, perspectives and challenges" Journal of The Royal Society Interface, 11(91), 20130505, 2014.

[4] D. Henriques, A.F. Villaverde, M. Rocha, J. Saez-Rodriguez , J.R. Banga "Data-driven reverse engineering of signaling pathways using ensembles of dynamic models" PLoS Computational Biology, 13(2), e1005379, 2017.

[5] P. Mendes and D. Kell "Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation", Bioinformatics, 14 (10), 869–883, 1998.

[6] C.G. Moles, P. Mendes and J.R. Banga, "Parameter estimation in biochemical pathways: a comparison of global optimization methods", Genome Research, 13 (11), 2467–2474, 2003.

[7] saCeSS2 - global mixed-integer optimization library version 2017A, https://zenodo.org/record/290219

[8] D.R. Penas, P. González, J.A. Egea, R. Doallo and J.R. Banga "Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy", BMC Bioinformatics, 18:52, 2017.

[9] J.A. Egea, R. Martí, J.R. Banga, "An evolutionary method for complex-process optimization", Computers & Operations Research, 37(2), 315-324, 2010.

[10] J.A. Egea, E. Balsa-Canto, MS. G. García, J.R. Banga, "Dynamic optimization of nonlinear processes with an enhanced scatter search method", Industrial & Engineering Chemistry Research, 48(9), 4388–4401, 2009.

[11] O. Exler, K. Schittkowski, "A trust region sqp algorithm for mixed-integer nonlinear programming", Optimization Letters, 1(3), 269—280, 2007.