Búsqueda de Vecindad Variable para el problema de la Minimización del Número de Errores en ordenaciones lineales

Eduardo G. Pardo[1], Borja Menéndez[2], Abraham Duarte[2], and Marc Sevaux[3]

[1] Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, Madrid, España,
{eduardo.pardo@upm.es}
[2] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles, España,
{borja.menendez, abraham.duarte}@urjc.es
[3] Lab-STICC, Université de Bretagne-Sud, Lorient, Francia,
{marc.sevaux@univ-ubs.fr}

Resumen El problema de la minimización del número de errores en ordenaciones lineales se define como un problema sobre un grafo con signos, donde las aristas están etiquetadas con signos positivos y negativos. De manera general, el objetivo del problema consiste en encontrar una ordenación lineal, que sitúe a cada vértice lo más cerca posible de sus vértices adyacentes con conexiones positivas y lo más lejos posible de sus vértices adyacentes con conexiones negativas. Específicamente, la función objetivo tratará de minimizar el número de conexiones negativas situadas antes que una conexión positiva. En este trabajo, se propone un algoritmo basado en la metodología Búsqueda de Vecindad Variable, particularmente en la variante Básica, para abordar el problema. El algoritmo propuesto ha sido comparado sobre diversos conjuntos de instancias y resulta competitivo con métodos previos en el estado del arte.

Keywords: Búsqueda de Vecindad Variable, Ordenación lineal, Grafos con signos

1. Introducción

Existe una gran cantidad de problemas de optimización donde la instancia de entrada es un grafo con signos. Esto quiere decir un grafo donde las aristas están etiquetadas con un signo positivo o un signo negativo. Esta clase de grafos son de gran utilidad a la hora de representar diferentes tipos de redes. Por ejemplo, han sido ampliamente utilizados en la representación de redes sociales, donde un signo positivo significa amistad entre los dos extremos de la arista, y un signo negativo significa enemistad. Otras aplicaciones relevantes del problema tienen que ver con: el establecimiento de canales de radio donde los vértices puede

redistribuir mensajes de algunos de sus vecinos (en su rango de comunicación); la predicción del signo de una hipotética conexión entre dos vértices; o bien simplemente con la manera de distribuir una mesa en una celebración social, donde se pretende que los invitados estén lo más próximos posible a sus amigos y lo más alejados de sus enemigos.

Dado un grafo con signos, una cuestión importante es cómo embeberlo en un espacio métrico de modo que cada vértice esté más próximo a sus vecinos positivos que de sus vecinos negativos. Este problema fue originalmente introducido en [7], quienes demostraron que si el grafo de entrada es completo, y el espacio métrico es la línea Euclídea (recta que, establecidos un origen y una unidad, permite asociar biyectivamente a cada punto un número real denominado abscisa del punto) entonces el problema es decidible en tiempo polinómico. Posteriormente, se demostró que la versión de decisión del problema era NP-Completa cuando el grafo con signos no está restringido a un grafo completo y este debe ser embebido en una línea Euclídea [1]. El estudio teórico de este problema ha sido más tarde extendido en [8]. En concreto, la variante abordada en este trabajo es la versión de optimización introducida en [12] y que posteriormente ha sido estudiada en [2]. En ella se define el concepto de error como aquellas situaciones en las cuales, dada una ordenación lineal de los vértices del grafo, un vértice tiene alguno de sus vecinos conectados por aristas positivas, más allá de algún vecino con una arista negativa. La función objetivo de este problema de optimización consiste en minimizar el número de errores de una ordenación dada.

Lo que resta de documento esta estructurado de la siguiente manera: en la Sección 2 se define formalmente el problema a resolver. En la Sección 3 se describe el algoritmo propuesto para abordar el problema. Por último, en la Sección 4 y en la Sección 5 se presentan respectivamente los resultados y conclusiones asociados a esta investigación.

2. Descripción del problema

Para describir el problema, en primer lugar, es necesario introducir el concepto de error. De manera informal, se podría decir que dada una ordenación lineal de los vértices de un grafo con signos, un error se produce cuando un vértice, que tiene como adyacente otro vértice conectado por una arista positiva se encuentra más alejado en la ordenación, que un segundo adyacente, conectado con arista con signo negativo. En este sentido, es importante tener en cuenta que para determinar los errores que se producen en un vértice, por un lado se consideran los vértices adyacentes que ocupan posiciones superiores y, por otro, los vértices adyacentes que ocupan posiciones inferiores.

En la Figura 1 se puede ver un ejemplo de ordenación linear de un grafo con 5 vértices y 4 aristas. En dicho ejemplo el vértice A es adyacente a los vértices B, C, D, E, teniendo dos conexiones positivas (A,C) y (A,E) y dos negativas (A,B) y (A,D). Dada la disposición de esta figura, los vértices B, C, D y E no generan ningún error, debido a que únicamente tienen un adyacente (por lo que da igual cuál sea el signo de la arista que los une), sin embargo, el vértice A, sí que genera

un error porque, a su derecha, el vértice E (unido por una arista positiva con A), se encuentra más lejos (posición 5) que el vértice D (también adyacente a A, con una arista negativa y más próximo a A). Sin embargo, a la izquierda de A, no se genera ningún error, ya que el único vértice adyacente positivo por ese lado de la ordenación, C, está más próximo que el único vértice adyacente negativo, B. Es importante destacar que, cuando se consideran los posibles errores que genere A, la disposición de los vértices B y C no afecta a los vértices D y E, y viceversa, a la hora de contabilizar los errores. Por lo tanto, el valor de la función objetivo de esa ordenación es 1.

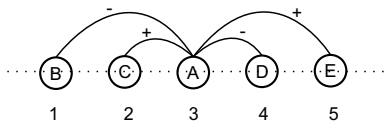

Figura 1. Ejemplo de ordenación lineal.

De manera formal, sea $G = (V, E^+ \cup E^-)$ la representación de un grafo donde V es el conjunto de vértices y E^+ y E^- los conjuntos de aristas positivas y negativas respectivamente. Se denotan como n y m la cardinalidad de los conjuntos V y $E = \{E^+ \cup E^-\}$, respectivamente. Dado un vértice $x \in V$ y sus vértices adyacentes denotados como $N(x)$, se especifican como $N^+(x)$ y $N^-(x)$ a los adyacentes unidos por una arista positiva y negativa respectivamente.

Dada la anterior definición de un grafo con signos, G, y una ordenación, π, de los vértices de G, se define el error en un vértice $x \in V$ producido en π como el par de vértices $\{y, z\} \in V$ tales que $(x, y) \in N^+(x)$ y $(x, z) \in N^-(x)$ y $\pi(y) < \pi(z) < \pi(x)$ o bien, $\pi(x) < \pi(z) < \pi(y)$.

Se denotará $\mathcal{E}_x^\pi$ como el número de errores en un vértice x producidos en una ordenación π. De este modo, el cálculo de la función objetivo de una ordenación π para un grafo G se denota como:

$$\mathcal{E}_\pi(G) = \sum_{x \in V} \mathcal{E}_x^\pi$$

Por lo tanto, el problema de optimización de la Minimización de Errores en ordenaciones lineales se define como la minimización de $\mathcal{E}_\pi(G)$ considerando todas las posibles ordenaciones $\pi \in \Pi$ de los vértices de G.

3. Propuesta algorítmica

En este trabajo se propone la utilización de la metodología Búsqueda de Vecindad Variable (VNS, del inglés *Variable Neighborhood Search*) para abordar el problema anteriormente descrito. VNS es una metaheurística basada en cambios sistemáticos de estructuras de vecindad como mecanismo para escapar de óptimos locales. La metaheurística original fue propuesta por Hansen y

Mladenović en 1997 [10], si bien existen numerosas extensiones. Entre las más conocidas se encuentran: *Reduced Variable Neighborhood Search* (RVNS), *Basic Variable Neighborhood Search* (BVNS), *Variable Neighborhood Descent* (VND) y *General Variable Neighborhood Search* (GVNS). En [5] y [6] se pueden ver una extensa descripción de las variantes más conocidas así como diversos tutoriales de uso. Más recientemente, VNS ha sido extendida mediante su aplicación a contextos más específicos, como son el caso de problemas multiobjetivo [3], implementaciones paralelas [4] [9] y contextos para problemas mín-máx y máx-mín [11] donde existen numerosas soluciones con el mismo valor de función objetivo asociado, aunque estructuralmente diferentes. En este trabajo se ha escogido la variante Búsqueda de Vecindad Variable Básica para abordar el problema y que, a continuación, pasa a describirse.

El Algoritmo 1 presenta el pseudocódigo del método propuesto. Comienza su ejecución con una solución factible, S, dada por un algoritmo constructivo. Es común en la literatura relacionada con VNS que el algoritmo comience desde una solución construida aleatoriamente, si bien un método más elaborado puede ayudar notablemente a la metaheurística o suponer un importante ahorro de tiempo. Hay otros dos parámetros más que configuran el algoritmo: t_{max}, que es el tiempo máximo de ejecución, y k_{max}, que es el número máximo de vecindades a explorar. En cada iteración, la solución es perturbada aleatoriamente con el procedimiento `Perturbacion` con tamaño k obteniendo una nueva solución, S', a la que después se le aplica una búsqueda local obteniendo una solución mejorada, S''. Finalmente, la solución obtenida es comparada con la mejor solución obtenida hasta el momento, a través del procedimiento `CambioDeVecindad`. Este último procedimiento tiene una doble función. Por un lado determina si la nueva solución encontrada es mejor que la previa (y en caso afirmativo la actualiza) y, por otro, determina el valor de k (la vecindad a explorar) en la siguiente iteración del algoritmo. En concreto, si la nueva solución S'' es mejor que la solución previa S entonces el valor $k = 1$, en caso contrario, el valor de k se incrementa en una unidad. Este proceso es repetido hasta que k alcanza el valor k_{max}. Llegado a ese punto, se habría completado una iteración completa del método. Si no se ha alcanzado t_{max}, el algoritmo inicia una nueva iteración, empezando de nuevo por la primera vecindad a explorar y con la mejor solución de la que disponga hasta ese momento.

En este trabajo, la solución inicial que recibe el procedimiento BVNS viene dada por un algoritmo constructivo voraz iterativo. Dicho algoritmo parte de una solución vacía y va añadiendo elementos a la misma. En concreto, el procedimiento que sigue el algoritmo para añadir elementos a la solución, de manera ordenada, está definido de la siguiente manera: primero, el algoritmo obtiene una lista de nodos, $L1$, ordenados de mayor a menor por la expresión $M \cdot Pos_{lab} + Neg_{unlab}$, donde M es un número real positivo, Pos_{lab} es el número de aristas positivas entre el nodo evaluado y la solución parcial, y Neg_{unlab} es es el número de aristas negativas entre el nodo evaluado y los nodos que no están todavía en la solución. Si solo existe un nodo en $L1$, se añade a la solución. Si no, se crea otra lista de nodos, $L2$, ordenados de mayor a menor con respecto al

Algoritmo 1: Esquema general de BVNS

Entrada: S; t_{max}; k_{max}
Salida: Una solución mejor o igual que S

```
1 mientras (! Condición de parada) hacer
2 |   k ← 1;
3 |   mientras k ≤ kmax hacer
4 |   |   S' ← Perturbacion(S, k);
5 |   |   S'' ← BusquedaLocal(S');
6 |   |   k ← CambioDeVecindad(S, S'', k);
7 |   fin
8 fin
9 devolver S
```

número de aristas positivas entre los nodos de $L1$ y la solución parcial. De $L2$ se obtiene un nodo al azar, que será el que se añada finalmente a la solución. El algoritmo termina cuando todos los nodos han sido añadidos a la solución.

La perturbación aplicada por el método `Perturbación` se realiza mediante k intercambios aleatorios de posición entre dos vértices de la solución. Para el proceso de `BúsquedaLocal`, en cambio, se han explorado dos vecindades: la resultante de aplicar movimientos de desplazamiento y la resultante de aplicar movimientos de intercambio. En la primera, LS_{desp}, un nodo dado se mueve de la posición en la que está a otra diferente. Como efecto colateral, los nodos que se encuentren a su derecha (si se mueve hacia la derecha) o a su izquierda (si se mueve hacia la izquierda) ven también desplazadas sus posiciones iniciales en una unidad. En la segunda, LS_{int}, un nodo dado intercambia su posición con otro nodo, de manera que el resto de nodos de la solucón permanecen en sus posiciones iniciales. Ambas búsquedas locales exploran sus respectivas vecindades con una estrategia *first improvement*, es decir, se acepta el primer movimiento que produzca una mejora. Por último, el procedimiento `CambioDeVecindad` está definido como sigue: si S'' es mejor que S, S es actualizada con S'' como la mejor solución encontrada hasta el momento y k se inicializa a 1; en otro caso, k se incrementa.

4. Experimentos

En esta sección se presentan los experimentos realizados para estudiar empíricamente la influencia de las estrategias propuestas. Posteriormente, se comparará la mejor variante con los mejores algoritmos previos del estado del arte [12]. Los algoritmos se han implementado en Java 7 y la experimentación se ha realizado en un Intel QuadCore con 2.6 GHz y 6 GB de RAM, con un sistema operativo Debian 8 de 64 bits. En primer lugar se presenta, en la Sección 4.1, una descripción detallada de las instancias. A continuación, en la Sección 4.2 se introducen una serie de experimentos preliminares que ayudarán a configu-

rar la versión final del algoritmo. Por último, en la Sección 4.3, se realiza la comparación con el estado del arte.

4.1. Descripción de las instancias

Se ha considerado un conjunto de instancias usadas previamente en la literatura para este problema de optimización. En concreto, se han utilizado los tres subconjuntos presentados en [12]: el conjunto `Complete`, el conjunto `Random` y el conjunto `Interval`, que contienen grafos completos, aleatorios e intervalos, respectivamente.

Tanto el conjunto `Random` como el `Interval` contienen 117 instancias. En ambos casos, el número total de aristas varía entre un 20 %, 50 % y 80 % del total de aristas que se podrían obtener si los grafos fuesen completos. De las aristas generadas, el número de aristas negativas varía entre un 20 %, 50 % y 80 % del total (y el resto, positivas). El número de nodos varía de 10 a 250, aumentando 20 nodos en cada grupo de instancias. En cambio, el conjunto `Complete` contiene 108 instancias, todas ellas con grafos completos, en las que el número de aristas negativas varía también entre un 20 %, 50 % y 80 % del total de aristas. El número de vértices, esta vez, varía de 10 a 230, aumentando 20 nodos en cada grupo de instancias.

4.2. Experimentación preliminar

En este apartado se ajustan los parámetros del algoritmo (valor de M en el procedimiento constructivo, valor de k_{max} para el algoritmo BVNS y búsqueda local que se utiliza en el mismo). Los experimentos preliminares para ajustar estos parámetros se han realizado sobre un subconjunto del total de instancias. En concreto, se ha seleccionado únicamente un 11 % del total de instancias, para que no se produzca un excesivo aprendizaje de las mismas, (12 del conjunto `Complete`, 13 del conjunto `Random` y otras 13 del conjunto `Interval`).

Primero se configura el procedimiento constructivo con diferentes valores del parámetro M. En la Tabla 1 se muestra, para cada valor escogido de M, el valor de la función objetivo (Valor F.O.), el tiempo de ejecución del algoritmo expresado en segundos (Tiempo (s)), la desviación obtenida respecto al mejor del experimento (Desv. (%)) y el número de mejores soluciones que se obtienen con esa configuración (#Mejores).

Como se puede apreciar en la Tabla 1, los tiempos de ejecución para diferentes valores de M apenas varían. En cambio, la desviación sí es muy diferente. Esto se debe, principalmente, al conjunto de instancias `Interval`, en las cuales se puede obtener una solución con 0 errores en cualquiera de las instancias. Por ello, en este subconjunto, instancias con un valor diferente de 0 obtiene una desviación muy elevada. En el número de mejores soluciones hay una situación parecida: según disminuye la desviación, aumenta el número de mejores soluciones (salvo en el caso de $M = 3{,}0$). Dado que este problema se podría solventar con una búsqueda local adecuada y, teniendo en cuenta el valor de la función objetivo,

Tabla 1. Configuración del parámetro M para el procedimiento constructivo.

M	Valor F.O.	Tiempo (s)	Desv. (%)	#Mejores
0.5	77135.97	0.01	1225.49	3
1.0	73616.82	0.01	547.99	5
1.5	72824.03	0.01	333.11	7
2.0	71968.32	0.01	66.61	10
2.5	71811.11	0.01	64.78	15
3.0	71917.37	0.01	9.34	10
3.5	72161.29	0.02	3.20	17

elegimos $M = 2{,}5$ para el algoritmo constructivo. Además, es el segundo mejor valor en cuanto a número de mejores soluciones.

Tras haber determinado la mejor configuración para el algoritmo constructivo, hay que determinar cuál es la mejor búsqueda local. Para ello, con el constructivo ya configurado, se aplica a las soluciones obtenidas con este un algoritmo de búsqueda local basado en desplazamientos y otro basado en intercambios. En la Tabla 2 se muestra, para cada una de las búsquedas locales, los mismos parámetros que en la Tabla 1. En ella, se puede ver que la búsqueda local basada en desplazamientos (LS_{desp}) obtiene mejores resultados que la basada en intercambios (LS_{int}) en todos los aspectos. Por ello, será la que finalmente se configure en el procedimiento BVNS.

Tabla 2. Comparativa entre búsqueda local basada en desplazamientos (LS_{desp}) y búsqueda local basada en intercambios (LS_{int}).

Búsqueda local	Valor F.O.	Tiempo (s)	Desv. (%)	#Mejores
LS_{desp}	69855.24	9.20	3.48	31
LS_{int}	70094.08	9.75	7044.31	14

Para finalizar la experimentación preliminar se tiene que configurar el parámetro k_{max} del algoritmo BVNS. En este caso, el valor de dicho parámetro se ha variado entre 3 y 7. Para realizar la comparativa se ha dejado correr el algoritmo BVNS una iteración, es decir, hasta alcanzar al valor establecido de k_{max}, con un tiempo máximo igual al número de nodos de la instancia (en segundos). Para cada valor de k_{max} se reporta, en la Tabla 3, los mismos datos que en tablas anteriores.

Los resultados expuestos en la Tabla 3 muestran que el mejor valor para k_{max} es 6, puesto que, aunque no es el que menos tiempo de ejecución necesita, es aquel que mejores valores presenta, tanto para la función objetivo como para la desviación y el número de mejores soluciones obtenidas.

Tabla 3. Configuración del parámetro k_{max} para el algoritmo BVNS.

k_{max}	Valor F.O.	Tiempo (s)	Desv. (%)	#Mejores
3	69335.16	56.66	2124.88	14
4	69396.08	63.82	2111.75	17
5	69367.50	71.24	17.61	16
6	69279.84	71.94	2.40	20
7	69457.55	74.63	2037.39	19

4.3. Experimentación final

Una vez se han identificado los parámetros que configuran los algoritmos, así como la estrategia de búsqueda local más prometedora, la experimentación final está dedicada a comparar el rendimiento de la propuesta con los mejores algoritmos del estado del arte. En concreto, se ha seleccionado un valor de $M = 2{,}5$ para el método constructivo, un valor de $k_{max} = 6$ para el BVNS, y una búsqueda local basada en desplazamientos embebida dentro del esquema de BVNS. En este caso, el conjunto de instancias considerado ha sido el conjunto total anteriormente presentado.

Los algoritmos del estado del arte, detallados en [12], son un algoritmo GRASP con una búsqueda local y un algoritmo voraz con búsqueda local. En la Tabla 4 estos algoritmos han sido denotados como GRASP+LS y Greedy+LS, respectivamente. Como se puede observar por los resultados obtenidos, el BVNS propuesto mejora los métodos previos en los tres conjuntos de instancias, si bien es cierto que es el más lento de los algoritmos comparados.

Es importante destacar que los resultados presentados en esta tabla han sido separados por conjuntos de instancias, debido a que la diferencia de escala de los números dejaría la columna Valor F.O. prácticamente sin sentido.

5. Conclusiones

En este trabajo se ha presentado un algoritmo, basado en la metodología Busqueda de Vecindad Variable, para el problema de la ordenación lineal de los vértices de un grafo etiquetado con signos positivos y negativos en las aristas. La variante elegida, Búsqueda de Vencidad Variable Básica, ha sido configurada mediante el diseño de un método de búsqueda local y de mecanismo de agitación. Los parámetros del algoritmo han sido ajustados mediante la realización de experimentación preliminar y, la mejor configuración final del algoritmo, ha sido comparada satisfactoriamente con métodos previos en el estado del arte sobre un conjunto de instancias de referencia.

Tabla 4. Comparativa entre los algoritmos del estado del arte y el BVNS configurado.

Conjunto	Algoritmo	Valor F.O.	Tiempo (s)	Desv. (%)	#Mejores
`Random`	GRASP+LS	66369.56	44.40	21.57	12
	Greedy+LS	66464.91	47.89	17.95	8
	BVNS	65301.74	130.00	0.00	114
`Interval`	GRASP+LS	2209.43	71.94	2.40	42
	Greedy+LS	238.99	32.41	211638.98	78
	BVNS	97.85	23.48	13277.99	111
`Complete`	GRASP+LS	177418.33	107.47	2.15	11
	Greedy+LS	177697.22	106.59	2.71	11
	BVNS	175259.25	120.01	0.01	103
Promedio	GRASP+LS	79488.07	60.22	72410.87	65
	Greedy+LS	78934.80	58.08	4549.47	97
	BVNS	77718.63	126.85	2419.89	328

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad (Referencia TIN2015-65460-C2-2-P) y por la Comunidad de Madrid (Referencia S2013/ICE-2894).

Referencias

1. Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O.: Sitting closer to friends than enemies, revisited. In: 37th International symposium on mathematical foundations of computer science (2012)
2. Derrien, A., Menendez, B., Sevaux, M., Pardo, E.G., Duarte, A.: VNS pour le Minimum Sitting Arrangement Problem. In: ROADEF: Recherche Opérationnelle et d'Aide à la Décision. Compiègne, France (Feb 2016), https://hal.archives-ouvertes.fr/hal-01275941
3. Duarte, A., Pantrigo, J., Pardo, E.G., Mladenović, N.: Multi-objective variable neighborhood search: an application to combinatorial optimization problems. Journal of Global Optimization pp. 1–22 (2014)
4. Duarte, A., Pantrigo, J., Pardo, E., Sánchez-Oro, J.: Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA Journal of Management Mathematics 27(1), 55–73 (2016)
5. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3), 449–467 (2001)
6. Hansen, P., Mladenović, N.: Variable Neighborhood Search, pp. 313–337. Springer US, Boston, MA (2014)
7. Kermarrec, A.M., Thraves, C.: Can everybody sit closer to their friends than their enemies? In: MFCS. pp. 388–399 (2011)
8. Kermarrec, A.M., Thraves, C.: Signed graph embedding: when everybody can sit closer to friends than enemies. arXiv preprint arXiv:1405.5023 (2014)
9. Menéndez, B., Pardo, E., Sánchez-Oro, J., Duarte, A.: Parallel variable neighborhood search for the min-max order batching problem. International Transactions in Operational Research (2016), http://dx.doi.org/10.1111/itor.12309

10. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers & Operations Research 24(11), 1097–1100 (1997)
11. Pardo, E.G., Mladenović, N., Pantrigo, J.J., Duarte, A.: Variable formulation search for the cutwidth minimization problem. Applied Soft Computing 13(5), 2242 – 2252 (2013)
12. Pardo, E.G., Soto, M., Thraves, C.: Embedding signed graphs in the line. Journal of Combinatorial Optimization 29(2), 451–471 (2015)