Cálculo de separadores de grafos utilizando búsqueda de vecindad variable reducida

D. Fernández-López, J. Sánchez-Oro, y A. Duarte[1]

Dept. Ciencias de la Computación, Universidad Rey Juan Carlos (Spain)
{ david.fernandez, jesus.sanchezoro, abraham.duarte }@urjc.es

Resumen Dado un grafo conexo y no dirigido $G = (V, E)$ y un límite entero positivo b, el problema del *Vertex Separator* (VS) consiste en encontrar el conjunto de vértices C, que eliminado divide G en componentes aisladas A, B, donde $\max(|A|, |B|) \leq b$. Este es un problema de optimización NP-completo y puede ser utilizado en un amplio rango de aplicaciones. Por ejemplo, en redes de telecomunicaciones un separador determina la capacidad y fragilidad de la red. En el campo de los algoritmos para grafos, el cálculo de pequeños separadores balanceados es muy útil, especialmente en los algoritmos del tipo divide y vencerás. Este trabajo presenta un nuevo algoritmo heurístico basado en el método *Variable Neighborhood Search* (VNS) para el cálculo de separadores. El método es comparado con el estado del arte (dos procedimientos de ramificación y poda recientes). Los resultados muestran que el método obtiene la solución óptima en todas las instancias pequeñas y medianas, y obtiene buenos resultados en instancias grandes. Aunque los métodos anteriores aseguran encontrar la solución óptima, tienen un tiempo de ejecución mucho mayor que el método presentado (aunque éste no asegura que la solución obtenida sea la óptima).

Keywords: Optimización combinatoria, Metaheurísticas, VNS, Separadores de grafos

1. Introducción

Se define un grafo $G = (V, E)$ mediante su conjunto de vértices V y de aristas E. Sea c_j el coste asociado con cada vértice $j \in V$. Un separador en G se define como un subconjunto de vértices o aristas que al ser eliminados convierten el grafo en inconexo. Los separadores también son conocidos por otros nombres, incluyendo bisectores, bifurcadores, cortes balanceados y particiones.

Este trabajo se centra en el problema del *Vertex Separator* (VS), que consiste en encontrar una partición de V en tres conjuntos no vacíos A, B y C, tal que no existe ninguna arista entre A y B, el tamaño de ambos conjuntos esta limitado por un entero positivo b (normalmente representado como una función de $|V|$), y que minimiza la suma de los pesos de los vértices en C.

Dado un grafo $G = (V, E)$, una solución x para el problema del VS puede representarse como tres conjuntos A, B, y C tales que $A \cup B \cup C = V$ y $A \cap B =$

$A \cap C = B \cap C = \emptyset$. Por lo tanto, el valor de la función objetivo, Sep, de una solución, $x = \{A, B, C\}$, se define como $Sep(x, G) = \sum_{j \in C} c_j$. Así, el problema de optimización puede ser formulado del siguiente modo:

$$\begin{aligned} &\min \sum_{j \in C} c_j \\ &s.t. \\ &\qquad \max\{|A|, |B|\} \leq b \\ &\qquad \forall i \in A \wedge \forall j \in B, (i, j) \notin E \end{aligned}$$

Nótese que el problema del VS es completamente equivalente a maximizar la suma de los pesos de los vértices en A y B (ver por ejemplo [2]).

La Figura 1.(a) muestra un ejemplo de un grafo G con cinco vértices y seis aristas. La Figura 1.(b) representa una posible solución, donde los conjuntos A, B, y C están delimitados mediante una línea discontinua. Si se asume, por simplicidad, que cada vértice tiene coste 1, el valor de esta solución es 1 ya que sólo hay un vértice en el conjunto C.

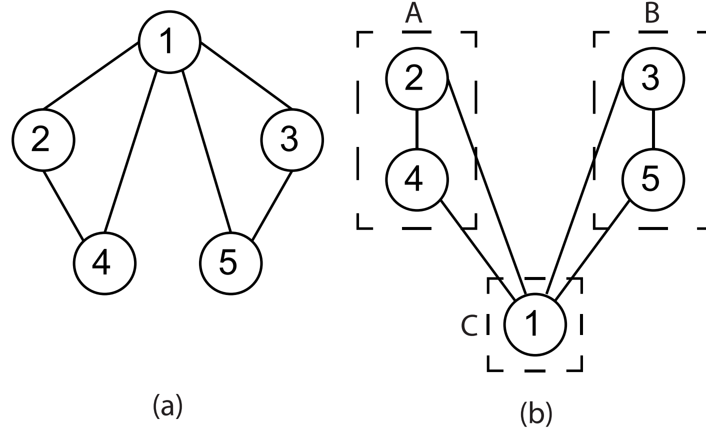

Figura 1. Ejemplo de un grafo (a) y una posible solución para el *Vertex Separator* (b)

Este problema de optimización fue originalmente presentado en [4] para el diseño de VLSI. Sin embargo, encontrar separadores balanceados y de pequeño tamaño es importante en muchos contextos. Por ejemplo, en redes de telecomunicaciones ([15], [13]). En el campo de la algoritmia para grafos, el calculo de separadores balanceados de pequeño tamaño es muy útil, especialmente en los algoritmos del tipo divide y vencerás ([17]). En bioinformática y biología computacional, se utilizan para simplificar la representación de las proteínas.

Encontrar el separador mínimo de un grafo general es un problema NP-completo [5]. Por lo tanto, los métodos exactos solo resuelven instancias de tamaño moderado. En concreto, Souza and Balas [2] propusieron una formulación de programación entera mixta, investigaron el politopo del VS y la envolvente

convexa de los vectores incidentes de los separadores. Este estudio teórico fue posteriormente incluido en un procedimiento de ramificación y poda. Los resultados computacionales mostraron que el método exacto encontraba el óptimo para instancias con tamaños comprendidos entre 50 y 150 en un tiempo de ejecución moderado. Biha and Meurs [7] introdujeron nuevas restricciones válidas y un límite inferior simple. Los experimentos mostraron que el nuevo método exacto resolvía de manera óptima todas las instancias propuestas por Balas and Souza [23] en menos tiempo. Por lo tanto, este método aparece como el estado del arte con respecto a los métodos exactos.

Muchos de los trabajos previos en el problema del VS están basados en el diseño de algoritmos de aproximación [16], mientras que otros aproximan la solución al VS mediante la solución del problema del *Edge Separator* [14,1,9,10]. En la literatura, hasta donde sabemos, solo existe un método heurístico [23], con el objetivo de encontrar soluciones factibles para el procedimiento de ramificación y poda, utilizando una relajación lineal del problema.

En este trabajo se propone un método basado en *Variable Neighborhood Search* (VNS) [11] para el problema del *Vertex Separator*. Esta metaheurística ha sido aplicada satisfactoriamente en un gran número de problemas de optimización. Por ejemplo en [21,8,18,22,20]. Concretamente se utiliza la variante *Reduced Variable Neighborhood Search* (RVNS) para diseñar un procedimiento de resolución eficiente y efectivo para el problema del VS. La Sección 2 presenta el procedimiento constructivo diseñado. La Sección 3 describe un nuevo procedimiento *shake*, diseñado para balancear la intensificación y la diversificación. La Sección 4 describe el procedimiento VNS en sí mismo, y cómo utiliza el constructivo y el método de *shake*. La Sección 5 muestra una extensa experimentación para validar el algoritmo propuesto comparando su rendimiento y tiempo de ejecución con los métodos del estado del arte. Por último, la Sección 6 resume las principales conclusiones de la investigación.

2. Procedimiento constructivo

Los árboles de nivel (ver [6]) son procedimientos constructivos basados en la partición de los vértices de un grafo en diferentes niveles, $L_1, \ldots, L_s$, de tal manera que los extremos de cada arista en el grafo estén en el mismo nivel L_i o en dos niveles consecutivos, L_i y L_{i+1}. Esta estructura de nivel garantiza que los vértices en niveles no consecutivos no son adyacentes. El procedimiento constructivo genera la estructura de nivel usando un método de búsqueda en anchura. Este algoritmo recibe como parámetro de entrada un vértice v y se construye un árbol de expansión con raíz en v. Por lo tanto, si un grafo tiene $|V|$ vértices, se construirán $|V|$ árboles de expansión diferentes.

El Algoritmo 1 muestra el pseudocódigo del procedimiento constructivo. En concreto, recibe como parámetro de entrada el árbol de expansión T y devuelve una solución factible para el problema del VS. El algoritmo comienza identificando el conjunto de niveles del árbol (paso 2). Después se inicializa un conjunto de variables auxiliares (pasos 3 al 7). El algoritmo intenta alternativamente asignar

un nivel del árbol a cada conjunto. Concretamente, el nivel L_1 es asignado a A y el nivel L_s es asignado a B. Si ningún conjunto está lleno, se realiza otra iteración asignando L_2 y L_{s-1} a A y B, respectivamente (ver pasos 9 al 24). Está lógica se mantiene hasta que se alcanza el nivel medio del árbol ($\lfloor s/2 \rfloor$) o uno de los subconjuntos está lleno (comprobar pasos 9 y 16). Si ni A ni B alcanza el máximo de su capacidad, el nivel crítico será $\lfloor s/2 \rfloor$. En otro caso, el nivel crítico será aquel en el que se llenó uno de los niveles (ver pasos 13 y 20). En el momento que un conjunto está lleno, los niveles restantes, menos el crítico, se introducen en el otro conjunto.

Algoritmo 1 Pseudocódigo del procedimiento constructivo

1: **procedure** `Constructivo`(T)
2: $T = \{L_1, L_2, \ldots, L_s\}$
3: $limite \leftarrow \lfloor s/2 \rfloor$
4: $i \leftarrow 1$
5: $j \leftarrow s$
6: $llenoA \leftarrow$ `false`
7: $llenoB \leftarrow$ `false`
8: **while** $(i < limite)$ **or** $(j > limite)$ **do**
9:     **if** $(|A| + |L_i| < b)$ **and not** $llenoA$ **then**
10:         $A \leftarrow A \cup L_i$
11:     **else**
12:         $llenoA \leftarrow$ `true`
13:         $critico \leftarrow i$
14:         $limite \leftarrow i + 1$
15:     **end if**
16:     **if** $(|B| + |L_j| < b)$ **and not** $llenoB$ **then**
17:         $B \leftarrow B \cup L_j$
18:     **else**
19:         $llenoB \leftarrow$ `true`
20:         $critico \leftarrow j$
21:         $limite \leftarrow j - 1$
22:     **end if**
23:     $i \leftarrow i + 1$
24:     $j \leftarrow j - 1$
25: **end while**
26: **for all** $v \in L_{critico}$ **do**
27:     $N(v) \rightarrow \{u \in V : (u, v) \in E\}$
28:     $N_A(v) \rightarrow N(v) \cap A$
29:     $N_B(v) \rightarrow N(v) \cap B$
30:     **if** $|N_A(v)| = \emptyset$ **then**
31:         $B \rightarrow B \cup \{v\}$
32:     **else if** $|N_B(v) = \emptyset$ **then**
33:         $A \rightarrow A \cup \{v\}$
34:     **else**
35:         $C \rightarrow C \cup \{v\}$
36:     **end if**
37: **end for**

Los elementos pertenecientes al nivel crítico son tratados en los pasos del 26 al 37. En concreto, para cada vértice, se comprueba si tiene adyacentes en A o B (pasos 28 y 29). Si el vértice correspondiente no tiene adyacentes en A, será asignado a B o viceversa (pasos 30 al 33). Finalmente, si el vértice tiene adyacentes en A y B será asignado a C.

3. Método *Shake*

Hay varios problemas de optimización donde la estructura de la solución apenas permite el diseño de una búsqueda local eficiente. Este es el caso del problema del VS. En concreto, una solución x está definida por tres subconjuntos disjuntos A, B, y C. Los movimientos en esa solución pueden ser muy complejos. Por ejemplo, eliminando un vértice en A (simétricamente en B) e insertándolo en B (simétricamente en A) la función objetivo no varía. En la misma línea, eliminando vértices de A o B e insertándolos en C la función objetivo empeora. Finalmente, casi nunca es posible eliminar un vértice de C e insertarlo en A o B porque la solución obtenida es infactible. La mayoría de algoritmos VNS utilizan el método *shake* para salir de un óptimo local mediante perturbaciones aleatorias de la solución. Sin embargo, en este caso es necesario incluir una componente intensificadora para reducir la aleatoriedad del método.

La función *shake* propuesta en este trabajo, $Shake(x, k)$ se muestra en el Algoritmo 2. En primer lugar identifica los tres conjuntos disjuntos A, B, y C (paso 2). Después, se identifica el conjunto de k vértices (llamado $Cand$) el cual será perturbado. Para diversificar la búsqueda, estos vértices son seleccionados de $A \cup B$ de manera aleatoria (paso 3). A continuación, el método genera una solución parcial eliminando los vértices k seleccionados (pasos 4 y 5). En esa solución parcial, se intenta reducir la función objetivo eliminando elementos de C e insertándolos en A o B (fase de intensificación). Primero se prueba si eliminando un elemento de C e insertándolo en A (paso 7) o B (paso 8) la solución es factible. Se utiliza la función `Factible` $(A \cup \{c\})$,la cual devuelve `true` si al insertar el elemento c no se sobrepasa el límite b (es decir, $A \cup \{c\} \leq b$) y $N_B(c) = \emptyset$. La función `Factible` $(B \cup \{c\})$ es completamente simétrica.

La inserción de un elemento en A o B se puede separar en tres casos diferentes (pasos 9 al 19). Si es posible insertar el elemento en ambos conjuntos, se insertará en el conjunto con menor cardinalidad para asegurar la similitud de tamaños entre conjuntos (pasos 10 al 14). En otro caso, el elemento se insertará en el único conjunto que genera una solución factible (pasos 15 al 19).

Por último, el procedimiento *shake* termina insertando los k elementos eliminados, primero, en A o B (de nuevo, manteniendo el tamaño de los conjuntos lo más similar posible). Cuando esto no sea posible, el elemento finalmente será insertado en C (ver pasos 21 al 37).

Algoritmo 2 Pseudocodigo del método *shake*

```
1: procedure Shake(x, k)
2: x = {A, B, C}
3: Cand ← SeleccionAleatoria(A ∪ B, k)
4: A ← A \ Cand
5: B ← B \ Cand
6: for all c ∈ C do
7:    feasA ← Factible(A ∪ {c})
8:    feasB ← Factible(B ∪ {c})
9:    if (feasA and feasB) then
10:      if ((|A| < |B|)) then
11:         A ← A ∪ {c}
12:      else
13:         B ← B ∪ {c}
14:      end if
15:   else if feasA then
16:      A ← A ∪ {c}
17:   else if feasB then
18:      B ← B ∪ {c}
19:   end if
20: end for
21: for all c ∈ Cand do
22:   feasA ← Factible(A ∪ {c})
23:   feasB ← Factible(B ∪ {c})
24:   if (feasA and feasB) then
25:      if ((|A| < |B|)) then
26:         A ← A ∪ {c}
27:      else
28:         B ← B ∪ {c}
29:      end if
30:   else if feasA then
31:      A ← A ∪ {c}
32:   else if feasB then
33:      B ← B ∪ {c}
34:   else
35:      C ← C ∪ {c}
36:   end if
37: end for
```

4. Búsqueda de vecindad variable reducida

La búsqueda de vecindad variable reducida (*Variable Neighborhood Search*, (VNS)) es una metodología para resolver problemas de optimización basada en el cambio de estructuras de vecindad. En los últimos años, se han propuesto una gran cantidad de variantes para el método del VNS. En [11,12] se presentan dos completas revisiones de la metodología.

Sea N_k con $1 \leq k \leq k_{max}$ un conjunto finito de estructuras de vecindades preseleccionadas, donde $N_k(x)$ es el conjunto de soluciones vecinas de x en la

k-esima vecindad. Cuando se soluciona un problema de optimización mediante el uso de estructuras de vecindad, el método VNS propone tres métodos diferentes para su exploración: (i) aleatorio, (ii) determinista, o (iii) mixto, el cual mezcla (i) y (ii). Este trabajo se centra en la variante RVNS, que consiste en la exploración (generación) de soluciones aleatorias en cada vecindad N_k. Esta variante no considera la aplicación de un procedimiento de búsqueda local. De hecho, los valores de las soluciones generadas son directamente comparados con el valor de la solución correspondiente, actualizando la mejor solución en caso de mejora.

Algoritmo 3 Pseudocódigo del RVNS

```
 1: procedure RVNS(k_max, t_max)
 2: repeat
 3:    x ← Constructivo()
 4:    k ← 1
 5:    repeat
 6:       x' ← Shake(x, k)
 7:       if Sep(x', G) < Sep(x, G) then
 8:          x ← x'
 9:          k ← 1
10:       else
11:          k ← k + 1
12:       end if
13:    until k = k_max
14:    t ← TiempoCPU()
15: until t > t_max
16: end RVNS
```

El pseudocódigo del RVNS se muestra en el Algoritmo 3. Tiene dos argumentos de entrada: la vecindad predefinida más grande (k_{max}) y el máximo tiempo de ejecución (t_{max}). El procedimiento comienza construyendo una solución factible (paso 3), utilizando el método descrito en la Sección 2. El RVNS comienza la búsqueda en la primera estructura de vecindad (paso 4). Después, la solución es perturbada utilizando la función `Shake` en el paso 6, obteniendo una nueva solución x' (ver Sección 3). En el paso 7, se decide si el RVNS realiza un movimiento (lo que implica que x' es mejor que x) o no. Si es así, la solución es actualizada (paso 8) y el método vuelve a la primera vecindad (paso 9). En otro caso, (es decir, x' es peor que x) el RVNS explora una vecindad mayor incrementando k (paso 11). Los pasos 6 y 12 son repetidos hasta que se alcanza k_{max}. Este parámetro determina el número máximo de vecindades diferentes a explorar en la iteración actual cuando no hay mejora en la solución. Por último, se repiten los pasos del 3 al 14 hasta que se alcanza t_{max}, comenzando cada iteración desde la construcción de una nueva solución.

Para diversificar la búsqueda el RVNS se ejecuta $|V|$ iteraciones independientes. Específicamente, se considera en cada iteración un árbol de expansión construido con un algoritmo de búsqueda en profundidad. El método completo devuelve la mejor solución encontrada.

5. Experimentos computacionales

Está sección muestra los experimentos realizados para evaluar la eficiencia del RVNS propuesto para solucionar el problema del VS. El algoritmo ha sido implementado en Java SE 6. Se han considerado cuatro conjuntos de instancias utilizadas previamente en este problema.

Las instancias DIMACS provienen del reto DIMACS en coloreado de grafos. Los grafos incluidos en este conjunto no tienen más de 150 vértices, con excepción de la instancia myciel7 que tienen 191 vértices. La segunda clase de instancias deriva de los grafos del *Matrix Market* (grafos MM para abreviar). Esta colección está compuesta por un conjunto de matrices de prueba estándar $M = M_{uv}$ derivadas de problemas en sistemas lineales, mínimos cuadrados y calculo de autovalores de una amplia variedad de disciplinas científicas e ingenieriles. Los grafos derivan de estas matrices considerando una arista (u, v) por cada elemento $M_{uv} = 0$. Se utilizan tres categorías de grafos MM. La primera, llamada MM-I, corresponde a todas las matrices con entre 20 y 100 columnas. La segunda, llamada MM-II, fue obtenida de las matrices cuyo número de columnas está entre 100 y 200. La tercera categoría de instancias, denotada MM-HD, solo contiene grafos con al menos una densidad del 35 %.

Los resultados del RVNS son comparados con los obtenidos por De Souza y Balas [23] (Pentium 4 a 2.5 GHz y 2 GB de RAM) y Biha y Meurs [7] (Pentium M740 a 1.73 Ghz y 1 GB de RAM). En nuestra opinión, el rendimiento de los ordenadores donde se ejecutaron los algoritmos no difiere demasiado. El algoritmo propuesto se ha ejecutado en un Intel Core 2 Duo T6400 a 2GHz y 3 GB de memoria RAM.

La Tabla 1 muestra los resultados experimentales. Cada línea se corresponde a uno de los conjuntos de instancias. La primera columna contiene el nombre del conjunto de instancias (Instancias) seguido, entre paréntesis, por el número de instancias que lo forman. La segunda columna contiene el valor promedio de la solución óptima (Opt.). La tercera y cuarta columna muestran, respectivamente, el promedio de tiempo de ejecución de los dos métodos exactos (Tiempo SB y Tiempo BM). La quinta y sexta columna contienen el promedio del valor de la función objetivo (RVNS) y su tiempo de ejecución en segundos (Tiempo). Finalmente, la séptima columna reporta el número de óptimos encontrados en cada conjunto de instancias.

Instancias	**Óptimo**	**Tiempo SB**	**Tiempo BM**	**RVNS**	**Tiempo**	**#Opt**
MM-I (24)	49.63	48.46	7.87	49.63	0.13	24
MM-II (20)	107.8	147.92	53.98	107.75	1.43	19
MM-HD (39)	72.49	13.74	13.67	72.21	0.46	33
DIMACS (21)	69.80	634.97	609.37	69.76	1.07	20

Tabla 1. Resultados obtenidos en los diferentes conjuntos de instancias.

En el conjunto más sencillo, MM-I, nuestro procedimiento obtiene todos los óptimos (24) en 0.13 segundos, frente a los 7.87 y 48.46 segundos que necesi-

tan BM y SB, respectivamente. En el segundo conjunto, MM-II, el algoritmo propuesto obtiene todos los óptimos menos uno (19 de 20) en 1.43 segundos, mientras que BM y SB necesitan 147.92 y 53.98 segundos. Además, en la instancia donde RVNS no encuentra el óptimo, se queda a una sola unidad. En el conjunto MM-HD el método RVNS no encuentra el óptimo en tan solo 6 instancias, quedándose como mucho a 2 unidades, y utilizando 0.46 segundos en promedio. En este caso, tanto SB como BM necesitan aproximadamente 13 segundos para finalizar. Finalmente, en el conjunto DIMACS se obtienen todos los óptimos salvo uno, en los que RVNS se queda a una unidad del óptimo. El tiempo promedio es de 1.07 segundos, frente a los 600 segundos que necesitan tanto SB como BM para encontrar el valor óptimo.

6. Conclusiones

En este trabajo se propone el uso del método *Reduced Variable Neighborhood Search* (RVNS) para la solución del problema del *Vertex Separator* (VS). Se presenta un procedimiento constructivo muy efectivo que permite encontrar regiones muy prometedoras en el espacio de búsqueda. Con el objetivo de diversificar la búsqueda, el RVNS comienza con diferentes soluciones iniciales. Además, también se propone un procedimiento *shake* que consigue un compromiso entre intensificación y diversificación. Se proporciona una extensiva comparación experimental con los mejores métodos anteriores en el estado del arte sobre un conjunto de 104 instancias. Los resultados experimentales muestran que el algoritmo propuesto encuentra la solución óptima en la mayoría de los casos con tiempos de ejecución realmente cortos.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Educación y Ciencia a través de los proyectos con referencias TIN2009-07516, TIN2011-28151, y TIN2012-35632, y por la Comunidad de Madrid a través del proyecto con referencia S2009/TIC-1542.

Referencias

1. Arora, S., Rao, S. & Vazirani,U. (2004).Expander flows, geometric embeddings, and graph partitionings. 36th Annual Symposium on the Theory of Computing, 222—231.
2. Balas, E. & de Souza, C. (2005). The vertex separator problem: a polyhedral investigation. Mathematical Programming, 103:583–608.
3. Basu, A., Bonami, P., Cornuejols, G, & Margot, F. (2009). On the Relative Strength of Split, Triangle and Quadrilateral Cuts. Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), New York, USA, 1220–1229.
4. Bhatt,S.N. & Leighton,F.T.(1984). A framework for solving VLSI graph layout problems. J. Computer System Sci., 28:300—343.

5. Bui,T.N. & Jones,C.(1992). Finding good approximate vertex and edge partitions is NP-hard. Inf. Process. Lett., 42:153—159.
6. Díaz, J., Petit, J. & Serna,M. (2002) A survey of graph layout problems. Journal ACM Computing Surveys, 34(3):313–356.
7. Didi Biha, M. & Meurs, MJ. (2011). An exact algorithm for solving the vertex separator problem. Journal of Global Optimization, 49:425–434.
8. Duarte, A., Escudero L.F., Martí R., Mladenovic N., Pantrigo J.J., Sánchez-Oro, J. (2012). Variable Neighborhood Search for the Vertex Separation Problem. Computers and Operations Research, 39(12):3247—3255.
9. Feige, U. & Kogan,S. (2004) Hardness of approximation of the balanced complete bipartite subgraph problem. Technical report MCS04-04, Department of Computer Science and Applied Math., The Weizmann Institute of Science.
10. Feige, U., Hajiaghayi, M. & Lee, J.R. (2005). Improved approximation algorithms for minimum-weight vertex separators. Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, 563–572.
11. Hansen, P. & Mladenović, N. (2003). Variable Neighborhood Search. International Series in Operations Research & Management Science, 57:145–184.
12. Hansen, P., Mladenovic, N., Perez, J.A.M. (2010). Variable neighbourhood search: methods and applications. Annals of Operations Research 175:367-–407.
13. Leighton F.T. (1983). Complexity Issues in VLSI: Optimal Layout for the Shuffle-Exchange Graph and Other Networks. MIT Press, Cambridge, MA.
14. Leighton, T. & Rao, S. (1999) Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM. 46:787-–832.
15. Leiserson, C. (1980). Area-efficient graph layouts (for VLSI). 21th Annual Symposium on Foundations of Computer Science. IEEE Computer Soc., Los Alamitos, CA, 270—280.
16. Lipton, R.J. & Tarjan, R.J. (1979). A Separator Theorem for Planar Graphs. SIAM Journal on Applied Math, 36:177–189.
17. Lipton, R.J. & Tarjan,R.E. (1980). Applications of a planar separator theorem. SIAM J. Comput., 9:615-–627.
18. Lozano, M., Duarte, A., Gortázar, F. & Martí R. (2012). Variable Neighborhood Search with Ejection Chains for the Antibandwidth Problem. Journal of Heuristics, 18:919–938.
19. Mladenovic, N., Hansen P.,(1997). Variable neighborhood search. Computers and Operations Research 24 (11): 1097-–1100.
20. Montemayor. A.S., Duarte A., Pantrigo J.J., Cabido R. (2005). High-Performance VNS for the Max-Cut Problem Using Commodity Graphics Hardware. Mini-Euro Conference on VNS (MECVNS 05), Tenerife (Spain), 1–11.
21. Pardo, E.G., Mladenovic, N., Pantrigo, J.J. & Duarte, A. (2013). Variable Formulation Search for the Cutwidth Minimization Problem. Applied Soft Computing, 13:2242–2252.
22. Sánchez-Oro, J., Pantrigo J.J., Duarte A. (2013). Balancing intensification and diversification strategies in VNS. An application to the Vertex Separation Problem. Technical Report. Dept. Ciencias de la Computación. Universidad Rey Juan Carlos.
23. de Souza, C. & Balas, E. (2005) The vertex separator problem: algorithms and computations. Mathematical Programming, 103:609-–631.
24. Strunk, W. Jr., & White, E.B. (2000). The elements of style. (4th ed.). New York: Longman.
25. Van der Geer, J., Hanraads, J.A.J., & Lupton, R.A. (2010). The art of writing a scientific article. Journal of Scientific Communications, 163:51-–59.