ISSN 1870-4069

Un nuevo algoritmo de ramificación y acotamiento para el problema de la bisección de vértices

Eduardo Del Ángel-Martínez[1], Héctor Fraire-Huacuja[1], Carlos Soto[1,2], Nelson Rangel[1], Laura Cruz-Reyes[1], Claudia Gómez-Santillán[1]

[1] Tecnológico Nacional de México / Instituto Tecnológico de Ciudad Madero, Tamaulipas, México
[2] Universidad de Cádiz, Escuela Superior de Ingeniería, Cádiz, España
edelangel.mz@gmail.com,automatas2002@yahoo.com.mx,
{nelson.rangel,lauracruzreyes,claudia.gomez}@itcm.edu.mx,
carlos.sotomonterrubio@alum.uca.es

Resumen. En este artículo se aborda el problema de la solución exacta del problema de la bisección de los vértices de un grafo. Este es un problema NP-duro que surge en el contexto de las redes de comunicación. Actualmente, el mejor algoritmo del estado del arte es el denominado BBVBMP, el cual utiliza la metodología de Ramificación y Acotamiento. En este artículo se propone un nuevo algoritmo basado en la misma metodología. El algoritmo propuesto toma ventaja de una heurística constructiva para inicializar la cota superior. Esta heurística considera el grado de los nodos a ser seleccionados como candidatos. Además, el algoritmo propuesto explora el árbol de búsqueda en el espacio combinatorio con una altura máxima de $\lfloor n/2 \rfloor$ nodos, reduciendo de esta forma el tamaño del árbol. Para evaluar el desempeño de las nuevas heurísticas se utilizó un conjunto de instancias estándar con óptimo conocido por construcción. Los resultados experimentales muestran que la heurística incorporada en el algoritmo propuesto supera ligeramente en eficiencia a la utilizada en el BBVBMP. Finalmente se compara el desempeño del algoritmo propuesto contra el BBVBMP, cuando resuelven el conjunto completo de instancias estándar para el problema de la bisección de vértices. Los resultados muestran que el algoritmo propuesto alcanza la misma calidad en las soluciones que BBVBMP, con una reducción del tiempo de ejecución de un 15 %.

Palabras clave: problema de la bisección de vértices, algoritmo de ramificación y poda, heurística constructiva.

A New Branch-And-Bound Algorithm for the Vertex Bisection Problem

Abstract. In this paper we approach the exact solution of the vertex bisection problem. It is a NP-hard problem which ocurrs in the context of

communication networks. Currently, the best state-of-the-arte algorithm is the BBVBMP, it use the branch-and-bound methodology. In this paper, we propose a new algorithm based in the same methodology. Our proposal takes advantage of a constructive heuristic to initiliaize the upper bound. This heuristic considers the degree of the vertex candidates. Also, the proposed algorithm explores the search tree in the combinatorial space with a maximum height of $\lfloor n/2 \rfloor$ nodes, decreasing the size of the tree. The performance of the new heuristics was measure with a benchmark set with known optimals by construction. The experimental results shows that the heuristic in the porposed algorithm outperforms in eficiency the heuristic incorporated in BBVBMP. Finally, the performace of the proposed algorithm is compared against the BBVBMP when solving the vertex bisection problem benchmark. The results show that the proposed algorithm reach the same quality solution as BBVBMP, with a reduction in the execution time of 15 %.

Keywords: vertex bisection problem, branch and bound algorithm, constructive heuristic.

1. Introducción

El problema de la bisección de vértices (VBP) pertenece a la familia de problemas de partición de grafos. Estos problemas consisten en encontrar una partición de sus vértices tal que una función objetivo en particular sea optimizada. El VBP consiste en dividir un grafo en dos subconjuntos B y B' de igual tamaño, excepto cuando el número de vértices es impar ya que el subconjunto B' tendrá un vértice más. Su función objetivo es minimizar el ancho del vértice, definida como el número de vértices en B con al menos un vértice adyacente en B'.

Su formulación formal es la siguiente: Dado un grafo $G(V, E)$ finito, acíclico y no dirigido, el VBP consiste en determinar una partición de V en dos subconjuntos B y B' donde $|B| = \lfloor n/2 \rfloor$ y $|B'| = |V \setminus B|$ tal que el número de vértices en B adyacentes a vértices en B' sea mínimo [4]. Un ordenamiento φ de un grafo no dirigido $G(V, E)$ con $n = |V|$ nodos es una función biyectiva $\varphi : V \rightarrow \{1, 2, ..., n\}$, mientras que $\Phi(G)$ es el conjunto de todos los ordenamientos del grafo G. Un ordenamiento es también llamado un ordenamiento lineal de todos los nodos de G. Dado un ordenamiento φ de un grafo $G = (V, E)$ y un entero i. La separación de vértices en la posición i de φ se define como $\delta(i, \varphi, G) = \left|\{ u \in L(i, \varphi, G) : \exists v \in R(i, \varphi, G), (u, v) \in E \}\right|$, donde $L(i, \varphi, G) = \{ u \in V : \varphi(u) \leq i \}$ y $R(i, \varphi, G) = \{ u \in V : \varphi(u) > i \}$. El costo del ordenamiento es una función F que asocia a cada ordenamiento φ de un grafo G un entero $F(\varphi, G)$. Sea F el costo de ordenamiento; el problema de optimización del ordenamiento asociado con F consiste en determinar algún ordenamiento $\varphi^* \in \Phi(G)$ de un grafo G tal que $F(\varphi^*, G) = \text{mín}_{\varphi \in \Phi(G)} F(\varphi, G)$. Para el problema de la bisección de vértices, $B = L(\lfloor n/2 \rfloor, \varphi, G)$ y $B' = V \setminus B$.

Formalmente, el VBP consiste en determinar el ordenamiento de vértices óptimo $\varphi^* \in \Phi(G)$ tal que minimice la siguiente función objetivo:

$$VB(\varphi^*, G) = \min_{\varphi \in \Phi(G)} VB(\varphi, G). \quad (1)$$

El VBP es un problema combinatorio perteneciente a la clase *NP-Duro* [2]. Este problema se presenta en las redes de telecomunicaciones cuando un mensaje debe ser transmitido a todos los nodos de la red. En esta tarea, en lugar de distribuir el mensaje desde un nodo a todos los nodos; la red de nodos es dividida en dos subconjuntos del mismo tamaño (B y B'). Estos subconjuntos son construidos con la intención de minimizar el número de nodos que tienen una conexión desde B hasta B'. El mensaje es almacenado en B y distribuido hasta B'. En este contexto, el grafo $G = (V, E)$ representa la red, V es el conjunto de nodos de la red y E es el conjunto de canales de comunicación. En este contexto, al VBP se le denomina como el problema de *gossip* [1].

El VBP ha sido resuelto en la literatura por métodos estocásticos y exactos. Los más recientes y relevantes métodos exactos para resolverlo son dos algoritmos de ramificación y acotamiento en [6] y en [8]. En [6], proponen un algoritmo exhaustivo. En su experimentación, usan un benchmark compuesto de 108 instancias: 5 Grids, 84 Small, 15 Trees y 4 Harwell-Boeing. En [8], proponen un algoritmo de ramificación y acotamiento llamado BBVBMP, el cual utiliza una heurística constructiva para generar una cota superior inicial. Además, proponen un nuevo cálculo de la cota inferior. Ya que el BBVBMP explora en el espacio combinatorio, implementa un nuevo procedimiento de ramificación que evita nodos duplicados. Utiliza la estrategia de búsqueda en profundidad para la exploración del árbol de soluciones. Actualmente, el BBVBMP es considerado el algoritmo de ramificación y acotamiento de mejor desempeño en el estado del arte para la solución exacta del VBP.

En este trabajo, se propone un nuevo algoritmo de ramificación y acotamiento que supera al BBVBMP. Este algoritmo es llamado BBVBP y sus principales características son:

1. Una nueva inicialización de la cota superior mediante una heurística constructiva, la cual considera como candidatos a los vecinos de menor grado del nodo actual.
2. Incorpora una estrategia de memorización de cada nivel del árbol de búsqueda. Esta memoria toma ventaja del cálculo previo del grado de los vértices para decidir si un nodo es podado o ramificado.
3. La exploración del árbol de búsqueda se realiza en el espacio combinatorio sin repetición dentro de $C^n_{\lfloor n/2 \rfloor}$.
4. Para optimizar la navegación la implementación de la representación del árbol se basa en estructuras de datos de tipo pila.

Este trabajo está estructurado como sigue: Las dos heurísticas constructivas para inicializar la cota superior se describen en la Sección 2. La estrategia de memorización utilizada para el cálculo de la cota inferior se describe en la

Sección 3. El enfoque de exploración del árbol de búsqueda y el algoritmo de ramificación y acotamiento se presenta en la Sección 4. En la Sección 5 se describen los experimentos computacionales y los resultados obtenidos. Finalmente, las conclusiones y trabajos futuros son presentados en la Sección 6.

2. Definición de la cota superior

En el algoritmo de ramificación y acotamiento, es necesario almacenar y actualizar la cota superior (ub) durante el proceso de exploración del árbol de búsqueda. El valor ub ayuda a decidir si un nodo es ramificado o podado. Si este valor es cercano al óptimo entonces permitirá reducir el tamaño del árbol de búsqueda mediante la poda de más nodos. Este puede ser calculado mediante la generación de una solución inicial aleatoria o por un algoritmo heurístico (como en nuestro enfoque propuesto). Nuestra propuesta es una heurística constructiva voraz para generar un límite superior inicial para el VBP. En esta sección se describen dos heurísticas: la primera heurística (H1) es de [8]; la segunda heurística (H2) genera un conjunto se soluciones que le permiten seleccionar la que presenta un mejor valor objetivo.

2.1. Cota superior con H1

H1 comienza seleccionando el vértice de grado mínimo en G, agregándolo a B junto con el conjunto de sus vecinos, si no es posible agregarlos en su totalidad entonces se agrega un subconjunto del mismo. Mientras el número de elementos en B sea inferior a $\lfloor n/2 \rfloor$, se selecciona un elemento de B en el cual su número de vecinos que no forman parte de B y que no sea 0, es mínimo, agregándolo a B junto con el conjunto de sus vecinos que no forman parte de B, o un subconjunto del mismo si no es posible agregarlos en su totalidad. (Para más detalles vea [8]).

2.2. Cota superior con H2

Sea $N(v) = \{ u \in V \text{ tal que } (v, u) \in E \}$ el conjunto de vecinos del vértice v y φ un conjunto vacío que representa a la mejor solución encontrada, nuestro enfoque propuesto en el Algoritmo 1 tiene el siguiente funcionamiento: inicialmente se crea un conjunto de vértices (A) de grado mínimo de G (línea 1-2). Para cada vértice v de A, se genera una nuevo conjunto B. Se crea un subconjunto C de vértices candidatos que inicialmente solo se encuentra conformado por v (línea 5). En las siguientes $\lfloor n/2 \rfloor$ iteraciones, un vértice v de grado mínimo del conjunto de candidatos que no forme parte del conjunto B, se extrae. De existir más vértices del mismo grado, es elegido el primer vértice de grado mínimo lexicográficamente. Dicho vértice v formará parte del conjunto B (linea 8) y todos sus vecinos que no formen parte de B serán agregados al conjunto de candidatos C (linea 9). Al completar una solución, su valor objetivo es calculado y se actualiza la mejor solución actual φ. Finalmente se regresa la solución de valor mínimo encontrada durante la exploración (φ).

Algoritmo 1: Heurística Constructiva Voraz (H2).

Entrada: Un grafo $G = (V, E)$, donde V es el conjunto de nodos y E es el conjunto de aristas; el número de nodos es $n = |V|$.

Salida: Un ordenamiento φ.

$\varphi \leftarrow \emptyset$
$A \leftarrow \left\{ v \in V \;\middle|\; |N(v)| \leq |N(u)|, \forall u \in V \right\}$
for $a \in A$ **do**
  $B \leftarrow \emptyset$
  $C \leftarrow \{ a \}$
  for 1 to $\lfloor n/2 \rfloor$ **do**
    Seleccionar $v \in C \setminus B$ tal que $|N(v) \setminus B| \leq |N(u) \setminus B|, \forall u \in C$
    $B \leftarrow B \cup \{ v \}$
    $C \leftarrow C \cup N(v) \setminus B$
  end for
end for
if $\varphi = \emptyset$ o $VB(B, G) < VB(\varphi, G)$ **then**
  $\varphi \leftarrow B$
end if
return φ

La figura 1 muestra la construcción de tres soluciones para un problema con 6 vértices. El número a la izquierda de cada vértice indica su grado. Los vértices con un doble círculo son los vértices candidatos. Los vértices en gris forman parte de la solución. Las aristas descartadas en el cálculo de los grados se encuentran marcadas con líneas punteadas.

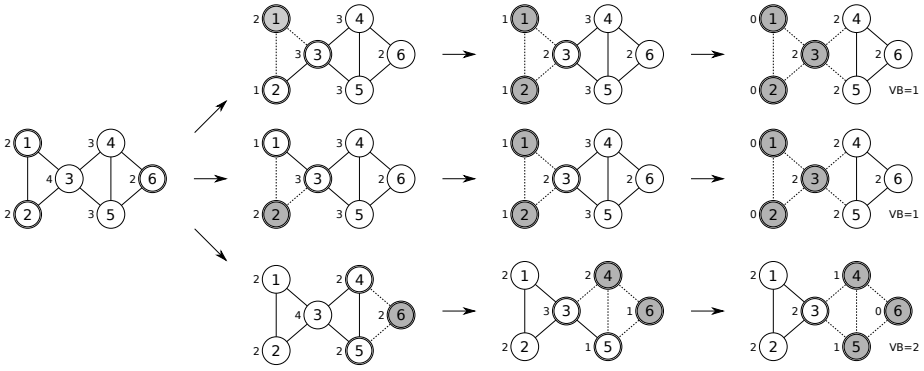

Fig. 1. Heurística Constructiva Voraz (H2).

La heurística comienza identificando el conjunto de los vértices de menor grado (vértices con doble círculo), en este caso, los vértices son $\{ 1, 2, 6 \}$. En los

siguientes pasos, se construye una solución con cada uno de ellos. La primera solución agrega el vértice 1 al conjunto B (marcado en color gris) y los vértices vecinos $\{2, 3\}$ se colocan en el conjunto de candidatos, removiendo las aristas para disminuir el grado. En la siguiente iteración se selecciona el nodo de menor grado disponible en el conjunto de candidatos; siendo el grado el número de vecinos del nodo actual que no forman parte de B, de existir varios de menor grado entonces se selecciona el primero lexicográficamente. En este caso se selecciona al vértice 2 (con grado 1) y son agregados como candidatos aquellos vecinos de 2 que no han sido agregados y que no forman parte del conjunto B. En la siguiente iteración, el vértice de menor grado y único disponible en la posición actual es el vértice 3, el cual es agregado al conjunto B. Al haber alcanzado los elementos requeridos para el conjunto B, los cuales son $\{1, 2, 3\}$, se evalúa la función objetivo con la solución actual, la cual da un valor de $VB = 1$. De igual forma se construyen las dos soluciones restantes. Al terminar las iteraciones, el algoritmo devuelve aquella solución con una función objetivo mínima. En caso de existir varias con el mismo valor objetivo, se devuelve la primera solución de valor mínimo encontrada.

3. Definición de la cota inferior

El cálculo de la cota inferior permite mejorar el desempeño del algoritmo de ramificación y acotamiento. Este enfoque sigue el principio de memorización de la contribución generada por las soluciones parciales en cada nivel del árbol de búsqueda.

La cota inferior (lb) es la contribución de la solución parcial en el nivel actual (n) que es dada por un vector Cbn de tamaño $\lfloor n/2 \rfloor$ que almacena cada una de las contribuciones por nivel del árbol siguiendo:

$$lb = Cbn_i = \begin{cases} l@ \quad l@@l@ \quad l@ \begin{matrix} Cbn_{i-1} + 1, \\ Cbn_{i-1} \end{matrix} & \begin{matrix} |N(nodo) \setminus B| > (\lfloor n/2 \rfloor - |B|), \\ \text{en otro caso.} \end{matrix} \end{cases} \tag{2}$$

Si el $lb \geq ub$, entonces el nodo actual es podado ya que este garantiza que no existe un nodo que al ser ramificado de él mejore el límite superior.

4. Algoritmo de ramificación y acotamiento

Un Algoritmo de Ramificación y Acotamiento es una estrategia para encontrar la solución óptima de problemas de optimización, explorando exhaustivamente todo el espacio de soluciones. Este algoritmo estructura el espacio de soluciones como un árbol de búsqueda y su objetivo es explorar todo el árbol completo para garantizar que encuentra la solución óptima. En el árbol de búsqueda, cada nodo interior representa una solución parcial del problema, mientras que los nodos en las hojas representan soluciones completas. El algoritmo

requiere el cálculo de una cota superior (ub) y una inferior (lb) para decidir si una solución parcial es ramificada o podada. Usualmente, una heurística constructiva voraz o una estrategia aleatoria inicializan el ub y este es actualizado cuando se encuentra un mejor valor en el cálculo de la función objetivo de las hojas del árbol de búsqueda. Mientras se exploran las soluciones parciales, lb es calculado y comparado contra ub para decidir si se poda o ramifica la solución parcial actual. En los problemas de minimización, si lb es inferior a ub; entonces el nodo actual es ramificado; en otro caso, el nodo actual es podado. Este procedimiento permite reducir el tamaño del árbol de búsqueda. Cuando una hoja es alcanzada, el valor objetivo de la solución completa es calculado. Si este es mejor que ub; entonces ub es actualizado. Cada vez que esto ocurre, la nueva solución completa actualiza a la mejor solución actual. Ya que se explora el árbol de búsqueda completo, se puede garantizar que ub es el valor objetivo óptimo y la última solución almacenada es la solución óptima.

En la literatura, existen diversas estrategias para la exploración del árbol de búsqueda, por ejemplo, mediante el uso de listas enlazadas [9,10], pilas [3,7] y una nueva estructura de datos como lo es la IVM [11]. En esta sección, proponemos un nuevo Algoritmo de Ramificación y Acotamiento basado en la representación con pilas. Las pilas controlan el nivel correspondiente de cada nodo desde el árbol de búsqueda mientras se realiza una exploración en profundidad. En el algoritmo 2, presentamos el Algoritmo de Ramificación y Acotamiento propuesto basado en pilas como una estrategia de navegación. El número de niveles o profundidad del árbol de búsqueda para este problema es de $\lfloor n/2 \rfloor$ como puede ser observado en la figura 2. Los niveles internos representan a las soluciones parciales en las cuales el número de nivel corresponde al número de nodos en B.

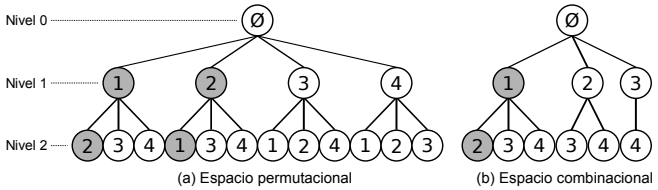

Fig. 2. Árbol de búsqueda en las permutaciones y en las combinaciones.

En la figura 2 se puede observar el árbol de búsqueda en el espacio de las permutaciones (figura 2a) y en el de las combinaciones (figura 2b) para un problema con 4 vértices. Este árbol de búsqueda tiene una profundidad de 2 niveles. En la figura 2a, se puede observar que la solución $\{1, 2\}$ tiene el mismo valor objetivo que $\{2, 1\}$. En la figura 2b, esta solución sólo aparece una vez. También, esto reduce el número de hijos en los nodos que se encuentran más a la derecha, teniendo con esto una reducción del árbol de búsqueda. La última rama en este árbol es $\{3, 4\}$.

Cuando se explora el árbol de búsqueda en el espacio de las permutaciones, algunas soluciones pueden contener nodos duplicados. En el VBP, las soluciones con nodos repetidos son soluciones infactibles. Así, soluciones con los mismos nodos en diferentes posiciones tendrán el mismo valor objetivo ya que esto garantiza tener el mismo número de vértices adyacentes entre B y B'. Estas soluciones se evitan mediante la exploración del espacio de búsqueda combinatorio sin repeticiones, el cual reduce el tamaño del árbol de búsqueda significativamente. En nuestro enfoque, exploramos el espacio de búsqueda combinatorio sin repeticiones con únicamente soluciones factibles. Para lograrlo, en cada nivel se usa la ecuación 3 para realizar el proceso de ramificación y agregar los nodos al árbol en orden lexicográfico. Esta estrategia permite reducir la exploración del árbol de búsqueda como se ve en la figura 2b.

$$hijos(v, nivel) \leftarrow \{\, u \in V : v < u \leq (n - \lfloor n/2 \rfloor + nivel + 1) \,\}. \tag{3}$$

4.1. Algoritmo de ramificación y acotamiento basado en pilas

En el algoritmo 2, B es un vector que almacena los nodos de la solución actual. Cbn es un vector el cual trabaja como memoria de la contribución (lb) de cada elemento en su nivel correspondiente. Su tamaño es equivalente al número de nodos en B. Este vector ayuda a acelerar la decisión de ramificar o podar un nodo evitando calcular la contribución de todos los nodos de la solución parcial. La pila *Elementos* almacena los nodos pendientes a ser explorados en cada nivel, mientras que la pila $Niveles$ almacena el nivel correspondiente de cada elemento almacenado. El conjunto de los nodos adyacentes o vecinos de un nodo v es denotado por $N(v) = \{\, u \in V : (v, u) \in E \,\}$.

El algoritmo 2 recibe el grafo G y el número de nodos n en G. De la línea 1 a la línea 5, se inicializan las variables necesarias. De la línea 6 a la línea 9 se inicializan las pilas *elementos* y *niveles* con todos los nodos posibles. En la línea 10, comienza el proceso principal; se realizan estos pasos mientras la pila *elementos* tenga elementos pendientes. En la línea 11 y 12 se recibe un nodo de la pila *elementos* y el nivel correspondiente de la pila *niveles*. La contribución de la solución parcial es actualizada de la línea 13 a la línea 16. En la línea 17 se decide si la solución actual será podada o será evaluada para ramificar. Si se cumple la condición de la línea 17, se agrega v al vector B en la posición *nivel* en la línea 18. La línea 19 verifica si una hoja es alcanzada. De la línea 20 a la línea 23, si una hoja es alcanzada se evalúa la función objetivo de la solución actual y si la nueva solución mejora al ub, la mejor solución global es actualizada. De la línea 25 a la línea 28, si una hoja no es alcanzada, se calcula el valor del *limite* y son agregados los nuevos elementos a la pila con su respectivo nivel. Finalmente, se regresa la solución óptima en la línea 29.

La figura 3 muestra el desarrollo del algoritmo de ramificación y acotamiento con la exploración del árbol de soluciones mediante pilas para una instancia de 6 nodos. El grafo representativo del problema es el observado en la figura 1. En el paso (a), la pila de elementos E es rellenada con los primeros nodos que conforman el problema desde 1 hasta el *limite* calculado desde el nivel 0 y la pila

Algoritmo 2: Algoritmo de Ramificación y Acotamiento exploración del árbol de búsqueda con pilas.

Entrada: Un grafo $G = (V, E)$, el número de nodos $n = |V|$.
Salida: El ordenamiento óptimo φ^* para G.

$\varphi^* \leftarrow \emptyset$
$ub \leftarrow$ inicializacion-ub(G, n)
$B \leftarrow \emptyset$
$nivel \leftarrow 0$
$Cbn_0 \leftarrow 0$
$limite \leftarrow (n - \lfloor n/2 \rfloor + nivel + 1)$
for $i \leftarrow 1$ **to** $limite$ **do**
  $Elementos.push(i)$
  $Niveles.push(0)$
end for
while $Elementos \neq \emptyset$ **do**
  $v \leftarrow Elemento.pop()$
  $nivel \leftarrow Niveles.pop()$
  if $\left|N(v) \setminus B\right| > (\lfloor n/2 \rfloor - |B|)$ **then**
    $Cbn_{nivel} \leftarrow Cbn_{nivel-1} + 1$
  else
    $Cbn_{nivel} \leftarrow Cbn_{nivel-1}$
  end if
  if $Cbn_{nivel} < ub$ **then**
    $B_{nivel} \leftarrow v$
    if $|B| = \lfloor n/2 \rfloor$ **then**
      $lb \leftarrow BV(B)$
      if $lb < ub$ **then**
        $\varphi^* = B$
        $ub \leftarrow lb$
      end if
    else
      $limite \leftarrow (n - \lfloor n/2 \rfloor + nivel + 1)$
      for $i \leftarrow (v + 1)$ **to** $limite$ **do**
        $Elementos.push(i)$
        $Niveles.push(nivel + 1)$
      end for
    end if
  end if
end while
return φ^*

N se rellena con el nivel correspondiente a cada uno de los nodos. El valor del ub se establece con el valor de 6 equivalente al número de nodos pertenecientes al problema al no provenir de una solución inicial. El vector B inicialmente se encuentra vacío. En el paso (b), se extrae el elemento 1 de la pila y se coloca en la variable e así como el nivel 1 y se coloca en la variable n. La actualización de B

se realiza ingresando en la posición n del vector B el elemento e, representando la solución parcial actual. El cálculo de la contribución se realiza mediante la ecuación 2 y se coloca en el vector Cbn en el nivel actual (Cbn_n). En el paso (c), ya que el nivel de la contribución actual es inferior al ub y no se ha alcanzado una hoja, se procede a ramificar, por lo cual, se agrega el conjunto de hijos a la pila E obtenido mediante la ecuación 3 y su nivel correspondiente ($nivel + 1$) en la pila N. En el paso (d), se realiza la extracción de los elementos siguientes de las pilas y se realiza la actualización tanto del vector B como de la contribución. Como la contribución es inferior al ub y aún no se alcanza una hoja, se realiza una nueva ramificación. En el paso (e), se realiza una nueva extracción, actualización de B y cálculo de la contribución. En este paso la contribución es inferior que el ub y es alcanzada una hoja. En el paso (f) se realiza el cálculo de la función objetivo y se compara con el ub. Ya que el valor de la función objetivo resultó tener un valor inferior al ub, se realiza la actualización del ub con el nuevo valor obtenido y se procede a guardar la solución actual. En el paso (g), se realiza la extracción de los siguientes elementos de las pilas, la actualización del vector B y el cálculo de la contribución del nivel actual. En este paso, ya que la contribución tiene el mismo valor que el ub, se procede a podar el nodo actual, omitiendo la ramificación. En los pasos siguientes, se puede observar que los cálculos de las contribuciones siempre igualan (o superan) el valor del ub, por lo que se procede a podar cada nodo siguiente. Este proceso se realiza hasta que la pila E se queda vacía.

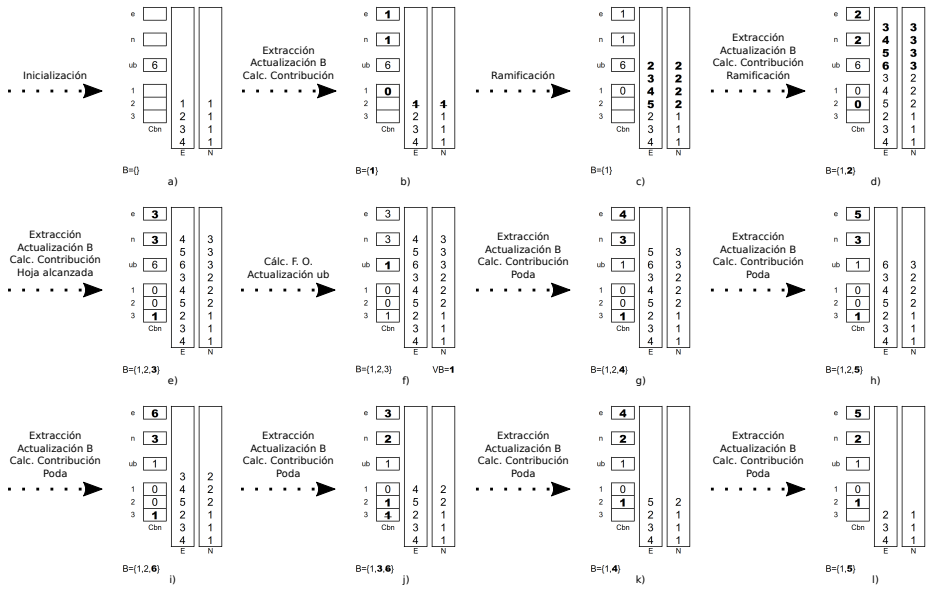

Fig. 3. Algoritmo de Ramificación y Acotamiento (BBVBP).

5. Experimentos computacionales

Los experimentos computacionales consisten en dos pruebas. En la primera, la calidad de las heurísticas (H1, H2) es medida en base al porcentaje del número de óptimos logrados y el porcentaje de proximidad a la optimalidad utilizando instancias con óptimo conocido por construcción. En la segunda prueba, se compara la eficiencia del algoritmo propuesto (BBVBP) contra el algoritmo de ramificación y acotamiento perteneciente al estado del arte (BBVBMP) mediante el tiempo de procesamiento acumulado, utilizando el benchmark utilizado en el estado del arte. Nuestra propuesta utiliza la mejor heurística de la experimentación anterior. La segunda prueba es una réplica de la experimentación reportada en [8].

La experimentación se realizó sobre un equipo con Procesador Intel Core i5-7200U a 2.50GHz Dual Core y 7.7GB de RAM. Los algoritmos fueron implementados en C++ y compilados con G++ 7.3.0. Se utilizó un total de 108 instancias del benchmark VSPBLIB [5]. De este benchmark se seleccionaron algunas instancias de cada conjunto: 1) Small con 84 instancias; 2) Grid con 5 instancias; 3) Tree con 15 instancias; 4) Harwell-Boeing (HB) con 4 instancias.

5.1. Calidad de las heurísticas

La calidad de las heurísticas (H1, H2) es medida en base al porcentaje del número de óptimos logrados y el porcentaje de proximidad al óptimo conocido por construcción. La tabla 1 muestra el porcentaje de valores óptimos logrados (columnas 2 y 3) y el porcentaje de proximidad al óptimo o porcentaje de *gap* (columnas 4 y 5) de cada heurística por cada tipo de instancia. La primera columna indica el tipo de instancia; entre paréntesis se indica el número de instancias. La última fila contiene los porcentajes resultantes. Los mejores resultados se encuentran con fuente negrita.

De la tabla 1 podemos observar que H2 logra un porcentaje de 80.95 % de valores óptimos encontrados y un menor porcentaje de proximidad al óptimo o porcentaje de *gap* de 10.78 %. Podemos observar que las dos heurísticas logran todos los valores óptimos en las instancias de tipo Grid y Tree. Como H2 logró mejores resultados, es seleccionada para inicializar el *ub* del BBVBP.

5.2. Eficiencia de los algoritmos de ramificación y acotamiento

La experimentación reportada en [8] establece un tiempo límite de 5 minutos por instancia de tipo Small, Grid y Tree, así como un tiempo de 1 hora por instancia de tipo Harwell-Boeing (HB). La tabla 2 muestra los resultados de la experimentación. Se midió la calidad de los algoritmos de ramificación y acotamiento en términos del porcentaje del número de los valores óptimos logrados y la eficiencia mediante el tiempo requerido para alcanzarlos.

Los resultados de la tabla 2 muestra la eficiencia en la búsqueda de las soluciones óptimas en el tiempo límite dado. Ambos algoritmos encontraron el mismo número de soluciones óptimas, pero el BBVBP logró una reducción del tiempo de cálculo del 84.35 % con respecto al BBVBMP.

Tabla 1. Porcentaje promedio de valores óptimos alcanzados y distancia hacia el óptimo (*gap*).

	% Optimal		% Gap	
Tipo de instancia	H1	H2	H1	H2
Grid (5)	100	100	0	0
Small (84)	25	42.86	49.23	32.34
Tree (15)	100	100	0	0
Promedio	75.00	**80.95**	16.41	**10.78**

Tabla 2. Porcentaje de valores óptimos y tiempo de los algoritmos de ramificación y acotamiento.

	% Optimal		Tiempo (sec)	
Tipo de instancia	BBVBMP	BBVBP	BBVBMP	BBVBP
Grid (5)	60	60	737.9	602.1
Small (84)	100	100	1680.5	16.2
Tree (15)	100	100	313.2	3.72
Harwell-Boeing (4)	25	25	10810.4	10801.0
Promedio	71.2	71.2	3385.5	**2855.7**

6. Conclusiones y trabajo futuro

En este trabajo se abordó el Problema de la Bisección de Vértices. Se propone un nuevo algoritmo de ramificación y acotamiento (BBVBP) y una nueva heurística constructiva voraz (H2) para la inicialización del límite superior (ub). Los resultados muestran que el algoritmo propuesto supera en eficiencia al algoritmo de ramificación y acotamiento del estado del arte (BBVBMP), reduciendo el tiempo de ejecución en un 18 %, 99 %, 98 %, 0.08 % sobre los conjuntos de instancias Grid, Small, Tree y Harwell-Boeing respectivamente. La reducción global que se logra sobre todos los conjuntos de instancias es de un 15 %. Como ambos algoritmos utilizan un proceso de exploración de complejidad similar, la diferencia en eficiencia se deriva de la complejidad del cálculo de la cota inferior. Para el algoritmo del estado del arte, dicho cálculo es de complejidad temporal $O(n^2)$ y para el algoritmo propuesto es $O(n)$. La heurística propuesta supera ligeramente en calidad a la heurística utilizada en el estado del arte mientras que la eficiencia es similar al tener ambas estrategias una complejidad $O(n^2)$. A fin de resolver el VBP, se propone una nueva estrategia de exploración del árbol de búsqueda. Esta estrategia únicamente explora soluciones factibles en el espacio combinatorio sin repeticiones, reduciendo así el número de nodos explorados significativamente. El algoritmo propuesto incluye un nuevo cálculo de la cota inferior para las soluciones parciales.

Como trabajo futuro se considera diseñar un nuevo mecanismo de cálculo de la cota inferior para mejorar la eficiencia del algoritmo de ramificación y

acotamiento mediante la reducción de los nodos ramificados durante el proceso de exploración del árbol de soluciones. Adicionalmente se implementará la exploración del árbol de soluciones utilizando una estructura de datos diferente a la utilizada en este trabajo.

Agradecimientos. Los autores manifiestan su agradecimiento a Jain Pallavi por haber apoyado con el código fuente de su algoritmo. Al Consejo Nacional de Ciencia y Tecnología (CONACYT) y al Tecnológico Nacional de México por el apoyo financiero brindado para el desarrollo de este proyecto. Al Laboratorio Nacional de Tecnologías de la Información (LANTI) por el acceso a los recursos de cómputo de alto desempeño bajo su resguardo.

Referencias

1. Böckenhauer, H.J.: Two open problems in communication in edge-disjoint paths modes. Acta Mathematica et Informatica Universitatis Ostraviensis 7(1), 109–117 (1999)
2. Brandes, U., Fleischer, D.: Vertex Bisection is Hard, too. Journal of Graph Algorithms and Applications 13(2), 119–131 (2009)
3. Delling, D., Fleischman, D., Goldberg, A.V., Razenshteyn, I., Werneck, R.F.: An exact combinatorial algorithm for minimum graph bisection. Mathematical Programming 153(2), 417–458 (nov 2015)
4. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Computing Surveys (CSUR) 34(3), 313–356 (2002)
5. Duarte, A., Escudero, F., Martí, R., Mladenović, N., Pantrigo, J.J., Sánchez-Oro, J.: Vertex separation problem (2012), `http://www.optsicom.es/vsp/#instances`
6. Fraire, H., Terán-Villanueva, J.D., García, N.C., Barbosa, J.J.G., del Angel, E.R., Rojas, Y.G.: Exact methods for the vertex bisection problem. In: Recent Advances on Hybrid Approaches for Designing Intelligent Systems, pp. 567–577. Springer (2014)
7. Huacuja, H.F., Castillo-García, N., Rangel, R.A.P., Flores, J.A.M., Barbosa, J.J.G., Valadez, J.M.C.: Two New Exact Methods for the Vertex Separation Problem. International Journal of Combinatorial Optimization Problems and Informatics 6(1), 31 (2015)
8. Jain, P., Saran, G., Srivastava, K.: Branch and bound algorithm for vertex bisection minimization problem. In: Advanced Computing and Communication Technologies, pp. 17–23. Springer (2016)
9. Kleniati, P.M., Adjiman, C.S.: A generalization of the Branch-and-Sandwich algorithm: From continuous to mixed-integer nonlinear bilevel problems. Computers and Chemical Engineering 72, 373–386 (2015)
10. Lalami, M.E., El-Baz, D.: GPU Implementation of the Branch and Bound Method for Knapsack Problems. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. pp. 1769–1777. IEEE (may 2012)
11. Leroy, R.: Parallel Branch-and-Bound revisited for solving permutation combinatorial optimization problems on multi-core processors and coprocessors. Ph.D. thesis, Université Lille 1 (2015)