UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

TÍTULO DE TESIS

"UN ALGORITMO PARA APAREAMIENTO EN GRÁFICAS BASADO EN DFS"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

CARLOS ZERÓN MARTÍNEZ

DIRECTORA DE TESIS:
M. EN I. MARÍA DE LUZ GASCA SOTO

2008

Hoja de Datos del Jurado

1. Datos del alumno Apellido paterno Apellido materno Nombre(s) Teléfono Universidad Nacional Autónoma de México Facultad de Ciencias Carrera Número de cuenta	1. Datos del alumno Zerón Martínez Carlos 56 03 94 90 Universidad Nacional Autónoma de México Facultad de Ciencias Ciencias de la Computación 099346225
2. Datos del tutor Grado Nombre(s) Apellido paterno Apellido materno	2. Datos del tutor M. en I. María de Luz Gasca Soto
3. Datos del sinodal 1 Grado Nombre(s) Apellido paterno Apellido materno	3. Datos del sinodal 1 Dra. Elisa Viso Gurovich
4. Datos del sinodal 2 Grado Nombre(s) Apellido paterno Apellido materno	4. Datos del sinodal 2 Dr. José de Jesús Galaviz Casas
5. Datos del sinodal 3 Grado Nombre(s) Apellido paterno Apellido materno	5. Datos del sinodal 3 Act. Leonardo López Monroy
6. Datos del sinodal 4 Grado Nombre(s) Apellido paterno Apellido materno	6. Datos del sinodal 4 M. en C. Federico Juárez Almaraz
7. Datos del trabajo escrito Título Número de páginas Año	7. Datos del trabajo escrito Un Algoritmo para Apareamiento en Gráficas basado en DFS 126 p 2008

Agradecimientos

A mis padres, Carlos y Violeta, que me han apoyado incondicionalmente a lo largo de mi vida y me han motivado para seguir siempre adelante y alcanzar mis objetivos.

A Lucy, por sus múltiples enseñanzas y por sus consejos para la elaboración del presente trabajo. A Elisa, Leonardo, José y Federico, por el tiempo dedicado a la revisión de este trabajo y por las observaciones y comentarios sobre el mismo. A todos los profesores de la Facultad de Ciencias con quienes tuve el privilegio de tomar clase, especialmente a Lucy, Corina, Elisa, Hortensia y José, por la forma en como imparten sus cursos.

A toda mi familia, por sus buenos deseos y ánimos durante el desarrollo de esta tesis y por todos los momentos compartidos.

Índice General

Introducción

Diversos problemas en la vida cotidiana pueden ser formulados como problemas de apareamiento máximo en gráficas. Algunos de esos problemas se pueden resolver en términos de apareamientos en gráficas bipartitas, ya que es posible dividir los objetos de estudio en dos grupos disjuntos y establecer un apareamiento de cada miembro de un grupo con un miembro del otro grupo de la mejor forma posible. Sin embargo, resulta interesante también observar qué ocurre cuando los objetos no necesariamente pueden dividirse en dos grupos, lo cual corresponde al caso general del problema de apareamientos.

En 1957 se demostró que una estrategia para encontrar un apareamiento máximo en una gráfica consiste en encontrar una trayectoria aumentante con respecto a un apareamiento dado, incrementar su tamaño y repetir este proceso hasta que ya no existan trayectorias aumentantes con respecto al apareamiento actual. No obstante, hasta 1965 se conocían únicamente algoritmos de complejidad exponencial para resolver el problema en gráficas no bipartitas. La razón por la cual no se había podido disminuir la complejidad de los algoritmos era que no se sabía cómo tratar con cierto tipo de ciclos de longitud impar en la búsqueda de trayectorias aumentantes. Jack Edmonds [9] describió un algoritmo polinomial que comprime esos ciclos, a los cuales denominó *blossoms*. El análisis explícito de este enfoque implica que cada vez que se tenga que contraer un ciclo, debe definirse una nueva gráfica en la cual se continúe buscando una trayectoria aumentante, y en cuanto ésta sea hallada, deben expandirse nuevamente los ciclos comprimidos que están involucrados en ella. En [11, 14, 19] se muestra cómo llevar a cabo todo este proceso sin construir explícitamente tal gráfica.

El presente trabajo es un desarrollo del artículo *Maximum matching in non-bipartite graphs without explicit consideration of blossoms* de Norbert Blum [5]; en el cual se describe un enfoque para resolver el problema de apareamientos que elimina la terminología tradicional de *blossoms* establecida por Edmonds. Para ello, se reduce el problema de encontrar una trayectoria aumentante con respecto a un apareamiento dado, al problema de determinar si un vértice es alcanzable desde otro en una gráfica bipartita dirigida. Dicho problema está descrito como *Reachability Problem* [5] y aquí haremos referencia a él como Problema RP. Además, se muestra que este problema, en general, puede ser

resuelto con una modificación del algoritmo de Búsqueda en Profundidad DFS.

Revisamos dos algoritmos polinomiales para la solución del problema de apareamiento máximo que siguen el enfoque de Blum y proporcionamos una implementación completa de ambos: el primero resuelve el problema para el caso particular de gráficas bipartitas, mientras que el segundo lo hace en general, utilizando conjuntos ajenos. Analizamos las complejidades y hacemos comparaciones desde el punto de vista práctico con una implementación del algoritmo de Edmonds [16, 19], la cual también emplea conjuntos ajenos.

En el Capítulo 1 revisamos los conceptos básicos y resultados referentes a los apareamientos, así como el método general para encontrar apareamientos máximos derivado de la teoría.

En el Capítulo 2 establecemos la forma en la cual se reduce el problema de encontrar trayectorias aumentantes en una gráfica dada, al problema RP en una gráfica bipartita dirigida, separando nuevamente el caso específico de gráficas bipartitas del caso general; con el fin de analizar con detalle la correspondencia entre las soluciones a ambos problemas para cada caso.

En el Capítulo 3, proponemos la solución del problema RP, separando nuevamente los casos mencionados y hacemos énfasis en la modificación al algoritmo de Búsqueda en Profundidad para adaptarlo al caso general y que denominamos Algoritmo Modificado de Búsqueda en Profundidad MDFS.

En el Capítulo 4, proporcionamos la justificación del algoritmo MDFS, con la cual también surgen herramientas para su implementación, la cual detallamos en el Capítulo 5, con las estructuras de datos auxiliares utilizadas.

En el Capítulo 6 describimos los algoritmos de Blum para la solución del problema de apareamientos máximos en el caso particular de gráficas bipartitas y en el caso general. También analizamos las complejidades de ambos algoritmos.

En el Capítulo 7 hacemos una comparación experimental entre los algoritmos descritos en el Capítulo 6 y el algoritmo de Edmonds, en el caso específico y en general. Todos los algoritmos fueron implementados en el lenguaje de programación Java.

Finalmente, anexamos tres apéndices: en el Apéndice A revisamos conceptos básicos de teoría de gráficas; en el B presentamos la base teórica y aspectos de la implementación del algoritmo de Edmonds; y en el Apéndice C incluimos los fundamentos para las estructuras de datos de conjuntos ajenos.

Capítulo 1

Apareamientos

En este capítulo se establece la teoría general de apareamientos en gráficas, así como un método general para encontrar apareamientos de máxima cardinalidad.

1.1 Conceptos Básicos

Definición 1.1. Sea G una gráfica y sea $M \subseteq E(G)$. Decimos que M es un **apareamiento** de G si cualesquiera dos aristas en M no tienen vértices extremos en común.

Definición 1.2. Un apareamiento M es **maximal** si $\forall\, e \in E(G) \setminus M$, $M \cup \{e\}$ no es un apareamiento.

En otras palabras, M es **maximal** si no es posible hacerlo más grande.

Definición 1.3. Un apareamiento M es **máximo** si no existe un apareamiento $M' \subseteq E(G)$ tal que $|M'| > |M|$.

Si además, $|M| = \lfloor |V(G)|/2 \rfloor$, decimos que M es **perfecto** o **completo**.

Un apareamiento máximo es también maximal, pero un apareamiento que es maximal no necesariamente es máximo.

Problema del apareamiento máximo.

Dada una gráfica G, encontrar un apareamiento máximo M de G.

Ejemplos:

1. En la Figura 1.1(a) se muestra el apareamiento

$$M_1 = \{v_2v_3, v_4v_5, v_6v_8, v_7v_{10}\}.$$

2. En la Figura 1.1(b) se muestra el apareamiento máximo

$$M_2=\{v_1v_2, v_3v_5, v_4v_7, v_6v_8, v_9v_{10}\}.$$

3. En la Figura 1.1(c) se muestra un apareamiento máximal que no es máximo:

$$M_3=\{v_2v_3, v_4v_5, v_7v_{10}, v_8v_9\}.$$

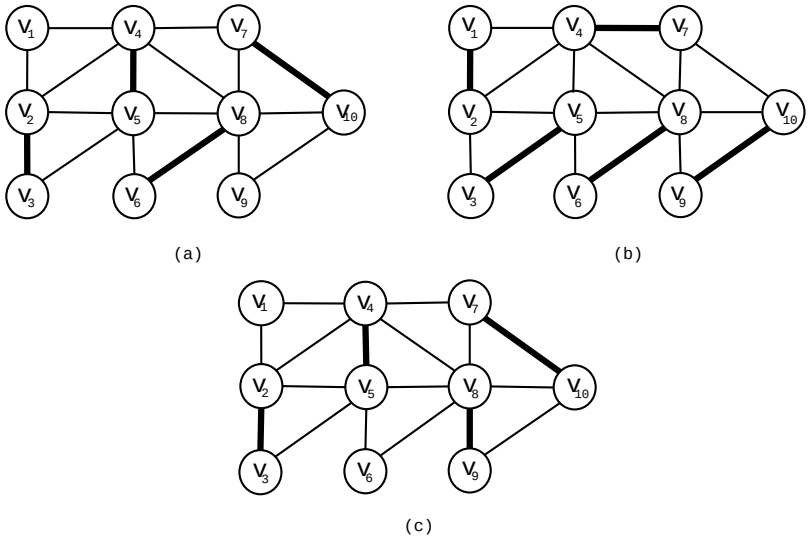

Figura 1.1: Apareamientos en gráficas

Definición 1.4. Sea G una gráfica y M un apareamiento de G. Una trayectoria $P : v_0, v_1, \ldots, v_k$ es **alternante** con respecto a M si las aristas en P de la forma $v_{2i}v_{2i+1}$ están en M y las aristas de la forma $v_{2i-1}v_{2i}$ están en $E(G) \setminus M$ o viceversa. A tal trayectoria también se le conoce como M-alternante.

En la Figura 1.1(a) tenemos que $P : v_2, v_3, v_5, v_4, v_7, v_{10}, v_8, v_6$ es una trayectoria M_1-alternante, pues las aristas $v_2v_3, v_5v_4, v_7v_{10}, v_8v_6$ están en M_1 mientras que $v_3v_5, v_4v_7, v_{10}v_8$ no.

Definición 1.5. Sea G una gráfica y sea $v \in V(G)$. Decimos que v es **apareado** con respecto a un apareamiento M si existe una arista $e \in M$ tal que v es extremo de e. A tal vértice se le conoce también como M-apareado. En caso contrario, decimos que v no es M-apareado.

Definición 1.6. Sea G una gráfica y M un apareamiento de G. Una trayectoria M-alternante $P : v_0, v_1, \ldots, v_k$ se dice que es **aumentante** con respecto a M si v_0 y v_k no son M-apareados. A tal trayectoria también se le conoce como M-aumentante.

En la Figura 1.1(c) tenemos que $P : v_1, v_2, v_3, v_5, v_4, v_7, v_{10}, v_9, v_8, v_6$ es una trayectoria M_3-aumentante, ya que v_1 y v_6 no son M_3-apareados.

1.2 Resultados

Lema 1.1. Sea G una gráfica y M un apareamiento de G. Entonces toda trayectoria M-aumentante tiene longitud impar.

Demostración. Sea $P : v_0, v_1, \ldots, v_k$ una trayectoria M-aumentante en G. Si P tuviera longitud par, como es M-alternante debe tener tantas aristas en M como en $E(G) \setminus M$, es decir, $P : v_0, v_1, \ldots, v_{2r-1}, v_{2r}$ donde $r \in \mathbb{N} \cup \{0\}$ y tenemos los siguientes dos casos:

Caso 1. Las aristas de la forma $v_{2i}v_{2i+1}$ están en M, mientras que aquellas que son de la forma $v_{2i-1}v_{2i}$ no están en M. En particular, la arista v_0v_1 está en M y por lo tanto v_0 es M-apareado, lo cual contradice el supuesto de que P es M-aumentante.

Caso 2. Las aristas de la forma $v_{2i}v_{2i+1}$ no están en M, mientras que aquellas que son de la forma $v_{2i-1}v_{2i}$ están en M. En particular, la arista $v_{2r-1}v_{2r}$ está en M y por lo tanto v_{2r} es M-apareado, lo cual contradice el supuesto de que P es M-aumentante.

$\therefore$ P tiene longitud impar. □

Consideremos cualquier apareamiento M en una gráfica G. Por el Lema 1.1, toda trayectoria M-aumentante puede escribirse de la siguiente manera:

$$P : v_0, v_1, \ldots, v_{2r-1}, v_{2r}, v_{2r+1} \text{ donde } r \in \mathbb{N} \cup \{0\}.$$

La trayectoria $P' : v_0, v_1, \ldots, v_{2r-1}, v_{2r}$ tiene r aristas en M y las otras r aristas no están en M, por lo cual P tiene $r + 1$ aristas que no están en M, ya que v_{2r+1} no es M-apareado.

Definición 1.7. Sea P una trayectoria M-aumentante en una gráfica G no dirigida. Denotamos por $M \oplus P$ a la diferencia simétrica de las aristas de M y P, es decir, aquellas que están en M pero no en $E(P)$ y en $E(P)$ pero no en M.

$$M \oplus P = (M \setminus E(P)) \cup (E(P) \setminus M)$$

Teorema 1.2. Sea G una gráfica, M un apareamiento de G y P una trayectoria M-aumentante. Entonces $M \oplus P$ es un apareamiento de G con cardinalidad $|M| + 1$.

Demostración. Veamos primero que $M \oplus P$ es un apareamiento de G. Supongamos que existen dos aristas $e, e' \in M \oplus P$ que tienen un extremo en común. Dada la definición de $M \oplus P$, tenemos tres casos:

1. $e, e' \in M \setminus E(P)$;
2. $e, e' \in E(P) \setminus M$;
3. $e \in M \setminus E(P)$ y $e' \in E(P) \setminus M$.

Caso 1. Tenemos que $e, e' \in M$ y además, tienen un vértice en común, lo cual contradice el hecho de que M es apareamiento.

Caso 2. Como P es una trayectoria M-aumentante, las aristas e y e' son de la forma $v_{2i}v_{2i+1}$ y por tanto no pueden tener vértices en común.

Caso 3. Supongamos que u es el vértice que tienen en común e y e'. Como $e' \in E(P) \setminus M$ entonces $e' = v_{2i}v_{2i+1}$ y u no puede ser ninguno de los extremos de P, puesto que los dos no son M-apareados.

Sin pérdida de generalidad, supongamos que $u = v_{2i}$; esto implica que la arista $e'' = v_{2i-1}v_{2i} \in M$. Por tanto, e y e'' son dos aristas en M que tienen en común a u, lo cual contradice el supuesto de que M es apareamiento.

$\therefore M \oplus P$ es un apareamiento.

Ahora veamos la cardinalidad de $M \oplus P$. Sabemos por el Lema 1.1 que P tiene $2r + 1$ aristas, donde $r \in \mathbb{N} \cup \{0\}$, de las cuales r son aristas de M y las $r + 1$ restantes no están en M.

$\therefore |M \oplus P| = |M \setminus E(P)| + |E(P) \setminus M| = |M| - r + r + 1 = |M| + 1.$ □

En la Figura 1.2(a) se muestra la trayectoria M-aumentante P, donde
$P : v_1, v_2, v_3, v_5, v_4, v_7, v_{10}, v_9, v_8, v_6$ y $M = \{v_2v_3, v_4v_5, v_7v_{10}, v_8v_9\}$.
La trayectoria P puede ser usada para construir un nuevo apareamiento M', el cual tiene una arista más que M y está dado por:
$M' = \{v_1v_2, v_3v_5, v_4v_7, v_{10}v_9, v_8v_6\}$, ilustrado en la Figura 1.2(b).

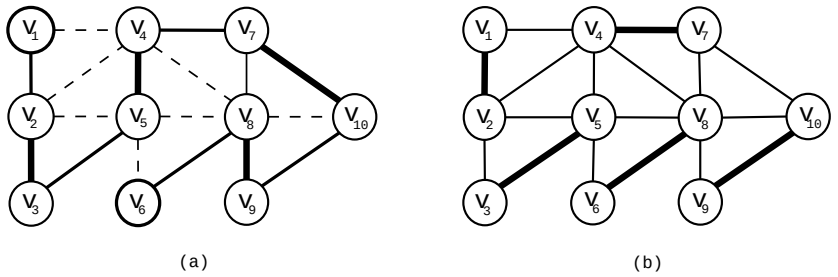

Figura 1.2: Un apareamiento obtenido a partir de una trayectoria aumentante

La clave en la mayoría de los algoritmos usados para encontrar un apareamiento máximo en una gráfica es la siguiente caracterización.

Teorema 1.3. Sea G una gráfica y $M \subseteq E(G)$ un apareamiento de G. Entonces M es máximo si y sólo si G no contiene trayectorias M-aumentantes.

Demostración.
$\Rightarrow$) Sea M un apareamiento máximo de G y supongamos que G tiene una trayectoria P, la cual es M-aumentante. Por el Teorema 1.2, se puede construir un apareamiento con la diferencia simétrica de las aristas de M y P tal que:
$|M \oplus P| = |M| + 1 > |M|$, contradiciendo la hipótesis de que M es máximo.
$\therefore$ G no tiene trayectorias M-aumentantes.

$\Leftarrow$) Supongamos que G no tiene trayectorias M-aumentantes y que M no es un apareamiento máximo, es decir, hay un apareamiento M' de G tal que $|M'| > |M|$.

Consideremos la subgráfica generadora H de la gráfica G con el siguiente conjunto de aristas: $E(H) = (M \setminus M') \cup (M' \setminus M)$. H posiblemente no es conexa.

Como dos aristas de un apareamiento no pueden tener en común un mismo vértice, se tiene que $\Delta(H) \leq 2$, ya que cada vértice de H es extremo a lo más de una arista de M y de otra arista de M'.

Por tanto, cada componente conexa de H puede ser: un vértice de grado cero, el cual puede ser M-apareado y M'-apareado, o bien, no apareado por M ni por M'; una trayectoria o un ciclo simple.

Consideremos las componentes conexas de H que tienen aristas, es decir, de los dos últimos tipos mencionados. Las aristas de cualquiera de esas componentes deben estar distribuidas de manera alternada en M y en M', por las razones expuestas anteriormente.

Por tanto, cada ciclo simple de H tiene longitud par, ya que en cada ciclo debe haber tantas aristas de M como de M'.

Pero $|M'| > |M|$, así que hay una trayectoria que tiene más aristas en M' que en M, la cual es M-aumentante en H y por ende, también lo es en G, contradiciendo el supuesto de que G no tiene trayectorias M-aumentantes.
$\therefore$ M es un apareamiento máximo de G.

□

Listado 1 Método general para encontrar apareamientos máximos

Entrada: Una gráfica G y un apareamiento $M \subseteq E(G)$, posiblemente $M = \varnothing$.
Salida: Un apareamiento máximo M_{max}.

```
1: while exista una trayectoria M-aumentante do
2:     Construye tal trayectoria P.
3:     M := M ⊕ P
4: end while
5: M_max := M
```

El Teorema 1.3 propone un método general para encontrar apareamientos máximos en una gráfica G, el cual está ilustrado en el Listado 1.

El problema clave es ahora el siguiente:
¿Cómo encontrar una trayectoria M-aumentante, para un apareamiento M dado, si es que ésta existe?

La solución a este problema la daremos de la siguiente forma:

- Lo reduciremos al problema RP en una gráfica bipartita dirigida.
- Resolveremos el problema RP de manera constructiva.

Capítulo 2

Reducción al problema RP

En este capítulo se establece la forma en la cual se reduce el problema de encontrar trayectorias aumentantes en una gráfica al problema RP en una gráfica dirigida bipartita. Para ello analizaremos por separado dos casos: la reducción para gráficas bipartitas y para gráficas en general.

En ambos casos construiremos una gráfica bipartita dirigida y resolveremos el problema RP en ella, de modo que la solución a este problema sirva para encontrar una solución al problema original.

2.1 Reducción para el caso de gráficas bipartitas

Sea $G = (A, B; E(G))$ una gráfica bipartita y $M \subseteq E(G)$ un apareamiento de G. Denotamos por V_M al conjunto de los vértices no M-apareados en G. Construimos a partir de G y M una gráfica *dirigida* G_M de la siguiente manera:

- Para cada arista $ab \in E(G)$ donde $a \in A$ y $b \in B$
  - Si $ab \in M$ entonces agregamos el arco (a, b) a G_M.
  - Si $ab \notin M$, agregamos el arco (b, a) a G_M.
- Agregamos dos vértices nuevos s y t junto con los arcos siguientes:
  - Para cada vértice $b \in B$ que no sea M-apareado agregamos el arco (s, b) a G_M.
  - Para cada vértice $a \in A$ que no sea M-apareado agregamos el arco (a, t) a G_M.

Explícitamente, la gráfica dirigida G_M está dada por:
$V(G_M) = V(G) \cup \{s, t\}$
$F(G_M) = \{(a, b) : ab \in M\} \cup \{(b, a) : ab \notin M\}$
$\cup \{(s, b) : b \in B \text{ y } b \in V_M\} \cup \{(a, t) : a \in A \text{ y } a \in V_M\}$

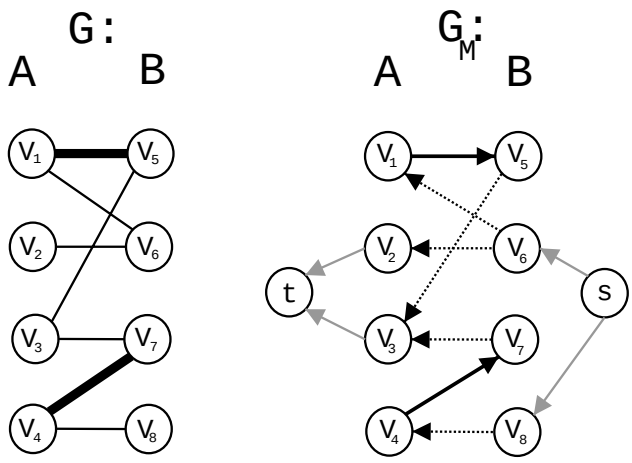

Figura 2.1: Construcción de G_M a partir de una gráfica bipartita G y un apareamiento M

En la Figura 2.1 se ilustra la construcción de la gráfica dirigida G_M: los arcos continuos y remarcados corresponden a aristas de M, los arcos punteados corresponden a aristas que no están en M; el resto son los que tienen como extremo inicial a s, además de los que tienen como extremo final a t.

Observamos que G_M es bipartita y podemos denotarla por:

$$G_M = (A \cup \{s\}, B \cup \{t\}; F(G_M)).$$

Teorema 2.1. Sea $G = (A, B; E(G))$ una gráfica bipartita y sea M un apareamiento de G. Sea G_M la gráfica dirigida construida a partir de G y M. Entonces existe una trayectoria M-aumentante en G si y sólo si t es alcanzable desde s en G_M.

Demostración.
$\Rightarrow$) Sea $P : v_0, v_1, \ldots, v_k$ una trayectoria M-aumentante en G, entonces v_0 y v_k están en V_M y por el Lema 1.1, tenemos que $|P|$ es impar.
Como G es bipartita, se tiene que v_0 y v_k pertenecen a distintos conjuntos de la bipartición de G. Supongamos, sin pérdida de generalidad, que $v_0 \in B$ y $v_k \in A$. Entonces, por construcción de G_M, se tiene que $(s, v_0), (v_k, t) \in F(G_M)$.

Puesto que P es M-aumentante, las aristas en P de la forma $v_{2i}v_{2i+1}$ no están en M, mientras que aquellas de la forma $v_{2i-1}v_{2i}$ están en M. Además, los vértices que aparecen en P de la forma v_{2i} están en B y aquellos de la forma v_{2i+1} están en A.

Así, por construcción de G_M, los arcos de la forma $(v_{2i}, v_{2i+1}), (v_{2i-1}, v_{2i})$ aparecen en $F(G_M)$, por lo que $[v_0, v_1, \ldots, v_k]$ es una trayectoria dirigida de v_0 a v_k

en G_M.
$\therefore P' : s, v_0, v_1, \ldots, v_k, t$ es una trayectoria dirigida de s a t en G_M.
$\therefore t$ es alcanzable desde s en G_M.

$\Leftarrow$) Sea $P' : s, v_0, \ldots, v_k, t$ una trayectoria dirigida de s a t en G_M.
Por construcción de G_M, se tiene que $v_0 \in B$, $v_k \in A$ y ambos vértices están en V_M. Como G_M es bipartita y los vértices s y t están en distintos conjuntos de la bipartición, se tiene que $|P'|$ es impar y por tanto la trayectoria dirigida $\bar{P} : v_0, \ldots, v_k$ en G_M también tiene longitud impar.

Los vértices en $\bar{P}$ de la forma v_{2i} están en B y aquellos de la forma v_{2i+1} están en A, por lo que las aristas de la forma $v_{2i}v_{2i+1}$ están en $E(G) \setminus M$ y las aristas de la forma $v_{2i-1}v_{2i}$ están en M. Entonces $P : v_0, v_1, \ldots, v_k$ es una trayectoria en G cuyas aristas están distribuidas de manera alternada en M y $E(G) \setminus M$, además de que sus extremos, v_0 y v_k, no son M-apareados.
$\therefore P$ es M-aumentante. □

En las gráficas de la Figura 2.1 se tiene en G_M, por ejemplo, la trayectoria dirigida $[s, v_6, v_1, v_5, v_3, t]$, por lo que la trayectoria M-aumentante correspondiente en G es $[v_6, v_1, v_5, v_3]$.

Por el resultado anterior, basta encontrar en G_M una trayectoria dirigida de s a t, $P' : s, v_0, v_1 \ldots, v_k, t$ para obtener automáticamente en G una trayectoria M-aumentante $P : v_0, v_1, \ldots, v_k$.

Para saber si existe tal trayectoria en G_M, es decir, si t es alcanzable desde s, usaremos el algoritmo de búsqueda en profundidad DFS (*Depth First Search*) en esta gráfica con el vértice inicial s.

2.2 Reducción para el caso general

Sea G una gráfica y M un apareamiento de G. Para la construcción de la gráfica dirigida G_M tenemos un problema: en primera instancia no podemos partir $V(G)$ en dos conjuntos A y B, pues G no necesariamente es bipartita. Así que la construcción de G_M se hará de la siguiente manera:

- Para cada vértice $v \in V(G)$ agregamos a G_M los vértices $[v, A]$ y $[v, B]$.
- Para cada arista $vw \in E(G)$
  - Si $vw \in M$ entonces agregamos $([v, A], [w, B])$ y $([w, A], [v, B])$ a G_M.
  - Si $vw \notin M$, agregamos $([v, B], [w, A])$ y $([w, B], [v, A])$ a G_M.
- Agregamos dos vértices nuevos, s y t, junto con los arcos siguientes:
  - Para cada vértice $v \in V_M$ agregamos $(s, [v, B])$ y $([v, A], t)$ a G_M.

De manera explícita, la gráfica dirigida G_M está dada por:

$$V(G_M) = \{[v, A], [v, B] : v \in V(G)\} \cup \{s, t\}$$

$$F(G_M) = \{([v, A], [w, B]), ([w, A], [v, B]) : vw \in M\}$$

$$\cup \{([v, B], [w, A]), ([w, B], [v, A]) : vw \notin M\}$$

$$\cup \{(s, [v, B]), ([v, A], t) : v \in V_M\}$$

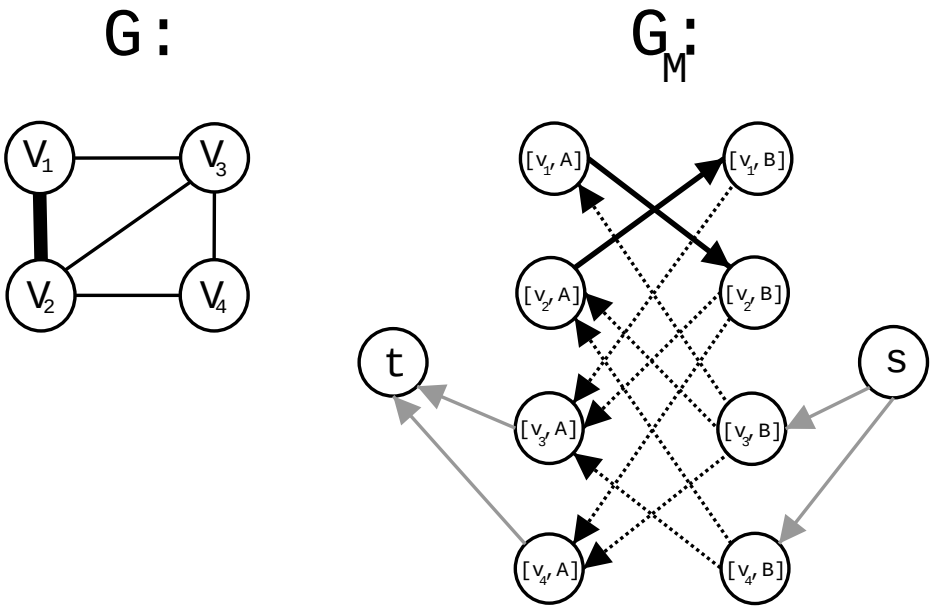

Figura 2.2: Construcción de G_M a partir de G y un apareamiento M

En la Figura 2.2 se ilustra un ejemplo de la construcción de la gráfica G_M a partir de una gráfica y un apareamiento que consta únicamente de una arista, la cual ha sido remarcada.

Análogamente al caso de gráficas bipartitas, hemos dirigido las aristas en M "de los vértices en A a los vértices en B" y las aristas en $E(G) \setminus M$ "de los vértices en B a los vértices en A". También agregamos arcos "de s a los vértices no M-apareados en B" y "de los vértices no M-apareados en A a t".

Observamos que G_M es bipartita y podemos denotarla por:

$$G_M = (\{[v, A] : v \in V(G)\} \cup \{s\}, \{[v, B] : v \in V(G)\} \cup \{t\}; F(G_M)).$$

Como los vértices $[v, A]$ y $[v, B]$ en $V(G_M)$ corresponden al mismo vértice $v \in V(G)$, no es suficiente encontrar alguna trayectoria dirigida de s a t en G_M para hallar una trayectoria M-aumentante en G. Las trayectorias que buscamos ahora deben tener características especiales: requerimos un nuevo tipo de trayectorias en G_M que contengan sólo a uno de los vértices $[v, A]$ y $[v, B]$ para cualquier $v \in V(G)$.

Definición 2.1. Sea G_M la gráfica dirigida construida a partir de la gráfica G y un apareamiento $M \subseteq E(G)$. Una **trayectoria** P en G_M es **original** si satisface lo siguiente:

$$\forall\ [v, A] \in V(G_M) : [v, A] \in V(P) \Rightarrow [v, B] \notin V(P).$$

En otras palabras, una trayectoria original debe contener a lo más alguno de los dos vértices $[v, A]$ y $[v, B]$, no a ambos, puesto que sólo estas trayectorias dirigidas pueden corresponder a trayectorias en G y además, según el siguiente teorema, son M-aumentantes.

Teorema 2.2. Sea G_M la gráfica dirigida construida a partir de la gráfica G y un apareamiento $M \subseteq E(G)$. Entonces existe una trayectoria M-aumentante en G si y sólo si existe una trayectoria original de s a t en G_M.

Demostración.
$\Rightarrow$) Sea $P : v_0, v_1, \ldots, v_k$ una trayectoria M-aumentante en G, entonces se tiene que $v_0, v_k \in V_M$ y por el Lema 1.1, tenemos que $|P|$ es impar. Así, por construcción de G_M, tenemos que $(s, [v_0, B]), ([v_k, A], t) \in F(G_M)$.

Puesto que P es M-aumentante, las aristas de la forma $v_{2i}v_{2i+1}$ no están en M, mientras que aquellas de la forma $v_{2i-1}v_{2i}$ están en M. Esto implica que los arcos $([v_{2i}, B], [v_{2i+1}, A])$ así como aquellos de la forma $([v_{2i-1}, A], [v_{2i}, B])$ están en $F(G_M)$.

Además, $v_i \neq v_j$ para $0 \leq i < j \leq k$, ya que P es trayectoria, por lo que $\big[[v_0, B], [v_1, A], \ldots, [v_{k-1}, B], [v_k, A]\big]$ es una trayectoria original de $[v_0, B]$ a $[v_k, A]$ en G_M y por tanto, una trayectoria original de s a t en G_M es $P' : s, [v_0, B], [v_1, A], \ldots, [v_{k-1}, B], [v_k, A], t$.

$\Leftarrow$) Sea $P' : s, [v_0, B], [v_1, A], \ldots, [v_{k-1}, B], [v_k, A], t$ una trayectoria original de s a t en G_M. Entonces $v_i \neq v_j$ para $0 \leq i < j \leq k$.

Por construcción de G_M, como $(s, [v_0, B])$ y $([v_k, A], t)$ están en $F(G_M)$, se tiene que $v_0, v_k \in V_M$. Como G_M es bipartita y los vértices s y t están en distintos conjuntos de su bipartición, se tiene que $|P'|$ es impar y por tanto la trayectoria $\bar{P} : [v_0, B], [v_1, A], \ldots, [v_{k-1}, B], [v_k, A]$ en G_M también tiene longitud impar.

Además, como los arcos de la forma $([v_{2i}, B], [v_{2i+1}, A])$ y aquellos de la forma $([v_{2i-1}, A], [v_{2i}, B])$ están en $F(G_M)$, tenemos que las aristas de la forma

$v_{2i}v_{2i+1}$ están en $E(G) \setminus M$, mientras que aquellas de la forma $v_{2i-1}v_{2i}$ están en M.

Entonces $P : v_0, v_1, \ldots, v_{k-1}, v_k$ es una trayectoria en G cuyas aristas están distribuidas de manera alternada en M y $E(G)\setminus M$, además de que sus extremos, v_0 y v_k, no son M-apareados. Por lo tanto, P es M-aumentante. □

En las gráficas de la Figura 2.2 se tiene en G_M, por ejemplo, la trayectoria original $\big[s, [v_3, B], [v_1, A], [v_2, B], [v_4, A], t\big]$ por lo cual, la trayectoria M-aumentante correspondiente en G es $\big[v_3, v_1, v_2, v_4\big]$.

Sin embargo, para este caso general, no es posible emplear DFS tal cual, dado que este algoritmo encuentra trayectorias que no necesariamente son originales, aunque en el fondo lo que se busca también es saber si t es alcanzable desde s en G_M. Esto se revisará en la siguiente parte con mayor detalle, donde será necesario hacer modificaciones al algoritmo DFS para adaptarlo al problema de apareamiento máximo en gráficas generales.

Capítulo 3

Solución al problema RP

En este capítulo se da solución al problema RP en la gráfica dirigida bipartita construida en el capítulo anterior, enfocándonos con mayor detalle en la solución del caso general.

Como se mencionaba anteriormente, el algoritmo DFS puede ser utilizado en el caso particular de las gráficas bipartitas, puesto que lo único que se necesita es verificar si uno de los vértices nuevos es alcanzable desde el otro en aquella gráfica. En cambio, esto no es suficiente para el caso general, ya que también debe determinarse si la trayectoria encontrada es original; para este último caso el algoritmo DFS no nos serviría tal y como está especificado.

Así, el objetivo fundamental de este capítulo será la modificación de DFS para verificar la existencia de trayectorias originales entre los vértices nuevos de la gráfica dirigida construida.

3.1 Solución para el caso de gráficas bipartitas

Sea G una gráfica, M un apareamiento de G y G_M la gráfica dirigida construida a partir de G y M. Aplicamos el algoritmo DFS a la gráfica G_M con el vértice inicial s.

Si en algún momento de la ejecución se etiqueta el vértice t, entonces t es alcanzable desde s y por tanto existe una trayectoria de s a t en G_M. Podemos terminar la ejecución de DFS y recuperar esta trayectoria considerando los predecesores directos en la búsqueda empezando con t y terminando cuando se encuentre algún vértice cuyo predecesor directo es s.

Después de obtener la solución al problema RP en G_M, se hace la transformación como se indica en el Teorema 2.1 a una solución del problema original, el cual consiste en encontrar una trayectoria M-aumentante en G.

Ese mismo teorema nos garantiza que si no existe una trayectoria de s a t en G_M, entonces no existen trayectorias M-aumentantes en G, por lo que basta con verificar en la ejecución de DFS que no se etiqueta al vértice t, para concluir que M es un apareamiento máximo.

3.2 Solución para el caso general

El algoritmo que describimos en esta sección lo llamaremos Algoritmo Modificado de Búsqueda en Profundidad (MDFS). Primero haremos una descripción de manera informal, mencionando algunas similitudes y diferencias con DFS y después daremos su especificación.

Consideremos la gráfica dirigida G_M. Decimos que el vértice $[v, B]$ es el opuesto del vértice $[v, A]$ y viceversa. Denotaremos por $[v, \bar{A}]$ al opuesto de $[v, A]$ y haremos lo mismo con $[v, \bar{B}]$.

Al igual que DFS, se construye un árbol enraizado en el vértice s y se divide el conjunto de arcos examinados de G_M en varias categorías:

1. **Arcos de descubrimiento o de árbol**. Son aquellos que conducen al descubrimiento de nuevos vértices $[v, X]$, donde $X \in \{A, B\}$, cuyos vértices opuestos no aparecen como sus predecesores durante la búsqueda.
2. **Arcos de retroceso débiles**. Son aquellos que conducen al descubrimiento de nuevos vértices $[v, A]$, cuyos vértices opuestos sí aparecen como sus predecesores durante la búsqueda.
3. **Arcos de retroceso**. Son aquellos que conducen a vértices ya descubiertos y están dirigidos de sucesores a predecesores.
4. **Arcos de avance**. Son aquellos que están dirigidos de predecesores a sucesores durante la búsqueda pero no son de árbol.
5. **Arcos de cruce**. Son aquellos que unen vértices que no son ni predecesores ni sucesores unos de otros durante la búsqueda.

Como podemos observar, MDFS agrega una categoría más a las cuatro que genera DFS y que son similares a las anteriores; la nueva categoría es la de Arcos de retroceso débiles.

MDFS utiliza un pila K para la organización de la búsqueda. La operación $\mathsf{top}(K)$ indica el último vértice agregado a K, mediante la operación push, la cual, como sabemos, inserta un elemento en el tope de K.

En cada paso, MDFS considera un arco $(\mathsf{top}(K), [w, Y])$, donde $Y \in \{A, B\}$, el cual no se ha tomado en cuenta anteriormente.
Sea $e = ([v, X], [w, \bar{X}])$ tal arco. Tenemos varios casos:

1. Si $X = A$, entonces $e = ([v, A], [w, B])$, es decir, $vw \in M$, por lo que el vértice $[w, B]$ no ha sido visitado, puesto que e es el único arco de G_M cuyo extremo final es ese vértice, por tanto e es un arco de árbol. Veamos la Figura 3.1(a).

2. Si $X = B$ entonces $e = ([v, B], [w, A])$, es decir, $vw \notin M$. Por tanto tenemos varios subcasos:

   a) Si $[w, A] \in K$, entonces este vértice es predecesor de $[v, B]$ durante la búsqueda. Como ambos están en K, se tiene una trayectoria de $[w, A]$ a $[v, B]$ antes de considerar el arco e, por tanto e es un arco de retroceso. Esto se muestra en la Figura 3.1(b).

   b) Si $[w, A] \notin K$ y $[w, B] \in K$, tenemos dos posibilidades:

      i) Si $[w, A]$ ha estado en K previamente, entonces $[w, B]$ no aparece como predecesor ni como sucesor de $[w, A]$ durante la búsqueda y por tanto e es un arco de cruce. Veamos la Figura 3.1(c).

      ii) Si $[w, A]$ nunca ha estado en K, entonces $[w, B]$ aparece como predecesor de $[w, A]$ durante la búsqueda y por tanto e es un arco de retroceso débil. Esto se observa en la Figura 3.1(d).

   c) Si $[w, A] \notin K$ y $[w, B] \notin K$, tenemos dos posibilidades:

      i) Si $[w, A]$ ha estado en K previamente, entonces puede ocurrir que $[v, B]$ aparezca como predecesor de $[w, A]$ durante la búsqueda; en ese caso e es un arco de avance, en caso contrario e es un arco de cruce. Esto se indica en la Figura 3.1(e).

      ii) Si $[w, A]$ nunca ha estado en K, como tampoco está su opuesto en K, la trayectoria de s a $[w, A]$ es original, por tanto e es un arco de árbol. Esto se ilustra en la Figura 3.1(f).

También se consideran como arcos de árbol aquellos cuyo extremo inicial es s y los que tienen como extremo final a t.

Definición 3.1. Decimos que MDFS construye una trayectoria original P si el árbol creado por MDFS contiene a P.

Definición 3.2. Decimos que MDFS encuentra una trayectoria original P', $P : s, P', [v, B], [w, A]$ si MDFS construye la trayectoria original $P' : s, \ldots, [v, B]$ y el arco $([v, B], [w, A])$ es considerado como arco de retroceso débil.

La operación push se efectúa en los casos 1 y 2.*c.ii*, con los vértices $[w, B]$ y $[w, A]$, respectivamente. Esta operación no se lleva a cabo en los casos 2.*a* y 2.*b.i*. Estos cuatro casos DFS los trata de la misma manera. A continuación analizaremos los casos restantes y descubriremos las diferencias entre ambos algoritmos.

Caso 2.*b.ii*. Como $[w, A]$ no ha estado en K, DFS efectuaría la inserción de $[w, A]$ en K. Sin embargo, como $[w, B] \in K$ y MDFS debe construir trayectorias

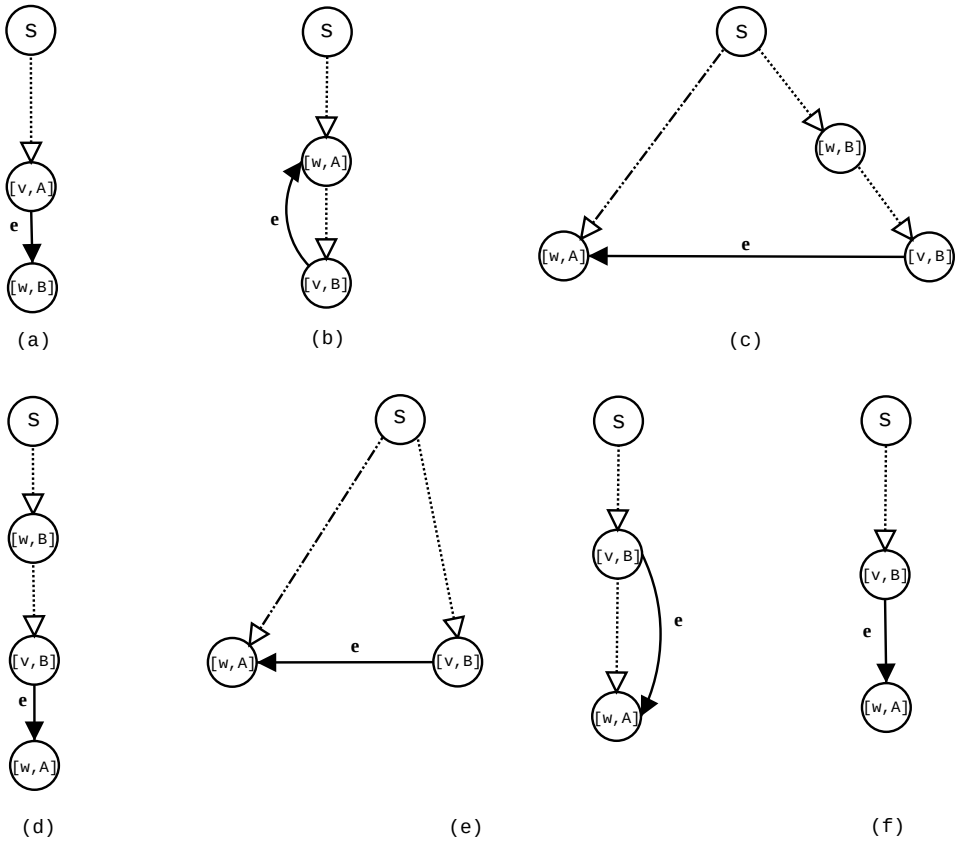

Figura 3.1: Esquema para la descripción informal de MDFS

originales en G_M, entonces MDFS no realiza esa operación.

Caso 2.*c.i.* Como $[w, A]$ ha estado previamente en K, DFS no insertaría este vértice en K. Pero el tratamiento del caso anterior puede llevarnos a la siguiente situación, la cual se ilustra en la Figura 3.2.

El algoritmo MDFS encuentra una trayectoria de $[w, A]$ a un vértice $[u, A]$. Es decir, de la trayectoria $Q : [w, A], \ldots, [x, B], [u, A]$ en G_M, MDFS construye la trayectoria $Q' : [w, A], \ldots, [x, B]$ y el arco $([x, B], [u, A])$ es considerado como arco de retroceso débil. Esto se da porque el vértice $[u, B]$ está en K y si se metiera a la pila el vértice $[u, A]$ ya no se tendría una trayectoria original. Pero después puede ocurrir que el vértice $[u, B]$ ya no esté en K y eventualmente se tenga que considerar al arco $e = ([v, B], [w, A])$. Si el vértice $[u, B]$ no aparece ni en P ni en Q, entonces la trayectoria $P' : s, P, [v, B], [w, A], Q, [u, A]$ es original y por tanto la acción que debe realizarse a continuación es la inserción de $[u, A]$ en K.

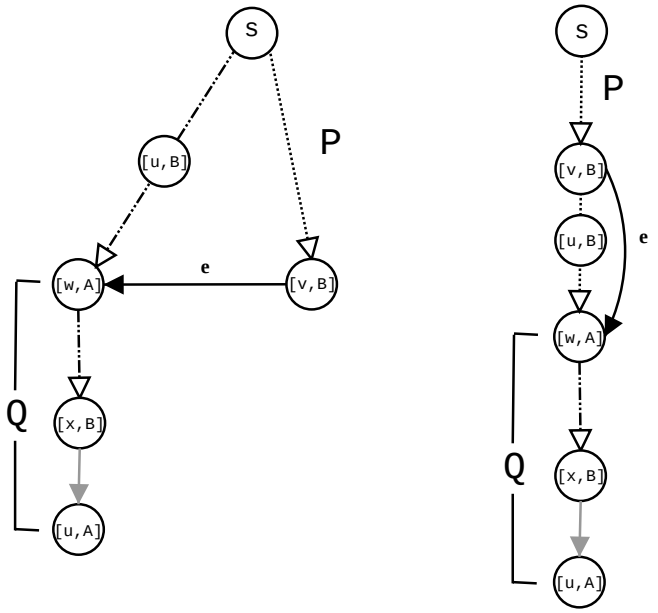

Figura 3.2: Esquema para la especificación del caso $2.c.i$

Notamos que hay una diferencia más entre ambos algoritmos, que tiene que ver con la forma en la cual se administra la pila; en el algoritmo DFS ésta contiene exactamente la trayectoria actual de búsqueda, mientras en el algoritmo MDFS difiere justamente cuando se presenta este caso: implícitamente tenemos un arco artificial o de extensión $([v, B], [u, A])_{[w,A]}$, etiquetado así para recordar el primer vértice que se agrega a la trayectoria entre $[v, B]$ y $[u, A]$. Por tanto, la nueva trayectoria de búsqueda es P'.

Describimos ahora el algoritmo MDFS de manera formal, teniendo en cuenta esta interrogante: ¿Cómo encontramos el vértice $[u, A]$ en el caso $2.c.i$? Suponemos, por lo pronto, que MDFS está organizado de modo que para cada vértice $[w, A] \in V(G_M)$ se mantiene la siguiente condición:

Después de efectuar la operación $\mathsf{pop}([w, A])$, MDFS tiene calculado un conjunto $L_{[w,A]}$ que contiene aquellos vértices $[u, A] \in V(G_M)$ tales que:

- MDFS encuentra una trayectoria $P : [w, A], \ldots, [u, A]$ en la cual no está $[u, B]$.
- $[u, A]$ no ha sido insertado en K.
- $[u, B]$ ha sido eliminado de K.

Listado 2 Algoritmo MDFS

Entrada: La gráfica dirigida G_M construida a partir de una gráfica G y un apareamiento M posiblemente vacío.

Salida: Una trayectoria original en G_M de s a t, si ésta existe.

```
1: push(s);
2: while K ≠ ∅ y no se haya encontrado una trayectoria de s a t do
3:    search;
4: end while
```

procedure search;

```
1: if top(K) = [t, B] then
2:    reconstruir una trayectoria original de s a t;
3: else
4:    marcar top(K) como visitado;
5:    for all [w, Y] ∈ N(top(K)) do
6:       if Y = B then
7:          {Caso 1}
8:          push([w, B]);
9:          search;
10:      else if [w, A] ∈ K then
11:         {Caso 2.a: no se inserta a [w, A] en K}
12:      else if [w, B] ∈ K then
13:         {Caso 2.b: no se inserta a [w, A] en K}
14:      else if [w, A] está marcado como visitado then
15:         {Caso 2.c.i}
16:         while L_[w,A] ≠ ∅ do
17:            elegir un vértice [u, A] ∈ L_[w,A];
18:            push([u, A]);
19:            search;
20:         end while
21:      else
22:         {Caso 2.c.ii}
23:         push([w, A]);
24:         search;
25:      end if
26:   end for
27:   pop
28: end if
```

Antes de esa operación, ponemos $L_{[w,A]} = \varnothing$.

Así, en la descripción de MDFS, omitimos el cálculo de los conjuntos $L_{[w,A]}$ para cada $[w, A] \in V(G_M)$. Más adelante veremos cómo se lleva a cabo dicho cálculo y probaremos además que $|L_{[w,A]}| \leq 1$.

Intuitivamente, cada que vez que se presenta el caso $2.b.ii$, se recuerda cuál es el vértice que "viola la originalidad" de la trayectoria actual de búsqueda, y cuando sale de la pila su vértice opuesto se hace el cálculo del conjunto $L_{[w,A]}$, con un vértice $[w, A]$ acorde con las condiciones mencionadas, y se utiliza ese conjunto cuando ocurre el caso $2.c.i$.

Denotamos al conjunto de vecinos de un vértice en G_M como:

$$N(v) = \{u \in V(G_M) : (v, u) \in F(G_M)\}.$$

Dada la bipartición de G_M, consideramos al vértice s como $[s, A]$ y al vértice t como $[t, B]$. En el Listado 2 se muestra la descripción de MDFS con todos los casos analizados.

3.3 Ejemplos

Ejemplo 1. Consideremos la gráfica G de la Figura 3.3(a) con el apareamiento indicado. La gráfica G_M correspondiente obtenida a partir de G y ese apareamiento se muestra en la Figura 3.3(b).

Si aplicamos DFS a la gráfica G_M, obtenemos la siguiente trayectoria, ilustrada en la Figura 3.4:

$$\big[s, [1, B], [2, A], [3, B], [4, A], [5, B], [6, A], [7, B], [5, A], [4, B], [3, A], [2, B], [1, A], t\big].$$

Observamos que esta trayectoria no es original, puesto que aparecen varios pares de vértices opuestos como por ejemplo: $[5, A]$ y $[5, B]$.

A continuación aplicaremos MDFS a G_M para tratar de encontrar una trayectoria original de s a t.

El resultado parcial, hasta el momento en que el vértice $[7, B]$ es descubierto, es el mismo que en la aplicación de DFS, es decir, la trayectoria original que construye MDFS es: $\big[s, [1, B], [2, A], [3, B], [4, A], [5, B], [6, A], [7, B]\big]$. La pila actual queda como se ilustra en la Figura 3.5(a).

El siguiente arco que se considera es $([7, B], [5, A])$. Pero $[5, B]$ está actualmente en la pila, por lo cual no podemos insertar $[5, A]$ en ella. MDFS encuentra una trayectoria de $[6, A]$ a $[5, A]$, a saber, $P_1 : [6, A], [7, B], [5, A]$.

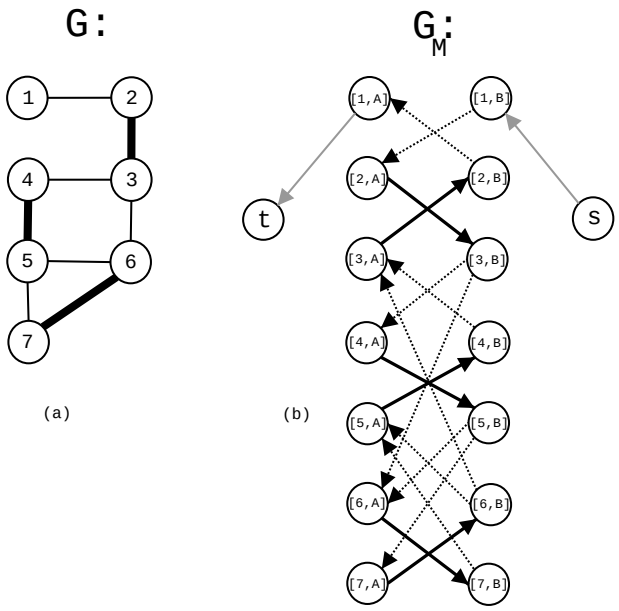

Figura 3.3: Gráficas para el Ejemplo 1

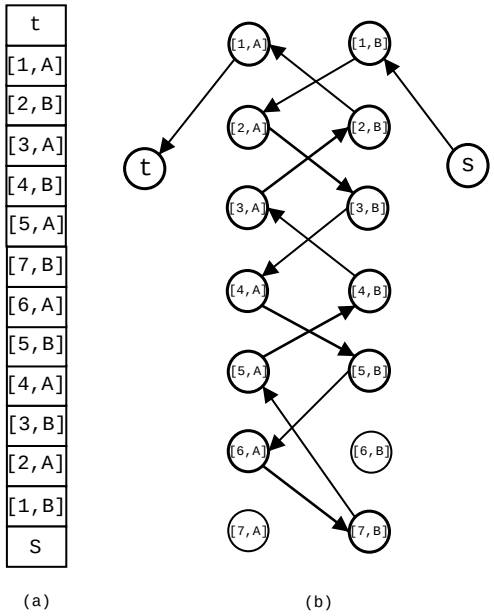

Figura 3.4: Aplicación de DFS al Ejemplo 1

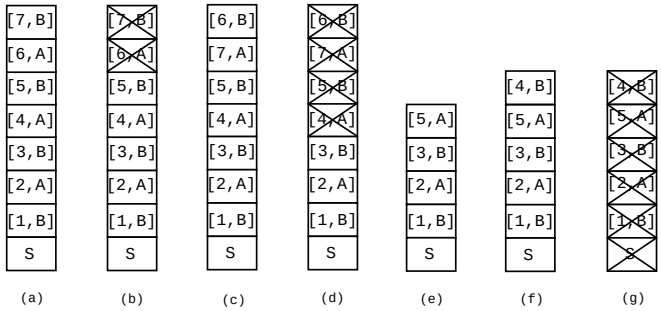

Figura 3.5: Pila de MDFS en su aplicación al Ejemplo 1

Hemos revisado todos los elementos en $N([7, B])$, entonces sacamos a $[7, B]$ de la pila; lo mismo pasa con el vértice $[6, A]$. El estado actual de la pila lo observamos en la Figura 3.5(b).

El siguiente arco es $([5, B], [7, A])$. El vértice $[7, B]$ no aparece en la pila en este momento así que insertamos el vértice $[7, A]$.

Luego tenemos el arco $([7, A], [6, B])$ y como no está el vértice $[6, A]$ introducimos en la pila el vértice $[6, B]$. Veamos la Figura 3.5(c).

Ahora examinamos el arco $([6, B], [5, A])$. Tenemos el inconveniente de que $[5, B]$ está en la pila y por tanto no hacemos la inserción correspondiente. MDFS encuentra una trayectoria de $[7, A]$ a $[5, A]$, a saber, $P_2 : [7, A], [6, B], [5, A]$. Lo mismo sucede cuando se revisa el arco $([6, B], [3, A])$, puesto que $[3, B]$ aparece en la pila. MDFS encuentra una trayectoria de $[7, A]$ a $[3, A]$, a saber, $P_3 : [7, A], [6, B], [3, A]$.

Hemos revisado todos los elementos en $N([6, B])$, $N([7, A])$ y $N([5, B])$, por tanto quitamos de la pila a los vértices $[6, B]$, $[7, A]$ y $[5, B]$ y hacemos entonces la siguiente actualización: $L_{[6,A]} = \{[5, A]\} = L_{[7,A]}$, dadas las trayectorias encontradas P_1 y P_2.

Posteriormente, notamos que hemos revisado todos los elementos de $N([4, A])$, por lo que la pila de MDFS queda como se muestra en la Figura 3.5(d).

Consideramos ahora el arco $([3, B], [6, A])$. El vértice $[6, A]$ no está actualmente en la pila, pero ya había estado ahí, por tanto no hacemos la inserción de este vértice; pero como MDFS encontró la trayectoria original P_1 y además, tenemos que $L_{[6,A]} = \{[5, A]\}$, entonces introducimos a la pila al vértice $[5, A]$, por lo que tenemos implícitamente el arco de extensión $([3, B], [5, A])_{[6,A]}$ representado en la pila, y al final ese arco se sustituye por P_1 en el árbol. Ahora hacemos $L_{[6,A]} = \varnothing$ y $L_{[7,A]} = \varnothing$. En la Figura 3.5(e) se observa el estado de la

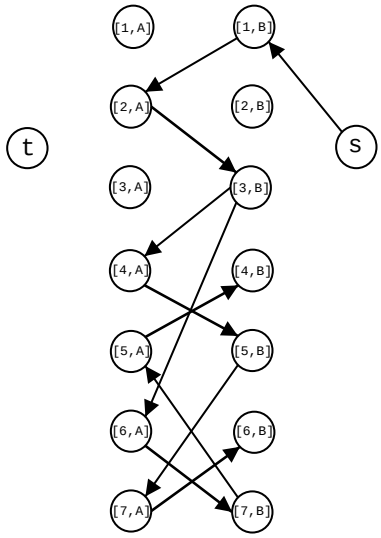

Figura 3.6: Árbol creado por MDFS en el Ejemplo 1

pila actual.

El arco que sigue es $([5, A], [4, B])$, así que se introduce a $[4, B]$ en la pila, como se muestra en la Figura 3.5(f).

Luego tenemos el arco $([4, B], [3, A])$, pero $[3, B]$ está en la pila, por tanto no hacemos la inserción correspondiente. MDFS encuentra una trayectoria de $[5, A]$ a $[3, A]$ y otra de $[6, A]$ a $[3, A]$, a saber, $P_4 : [5, A], [4, B], [3, A]$ y $P_5 : [6, A], [7, B], [5, A], [4, B], [3, A]$.

Hemos revisado todos los elementos en $N([4, B])$, $N([5, A])$ y $N([3, B])$, por tanto quitamos de la pila a los vértices $[4, B]$, $[5, A]$ y $[3, B]$ y en este momento hacemos la siguiente actualización: $L_{[7,A]} = L_{[6,A]} = L_{[5,A]} = \{[3, A]\}$, dadas las trayectorias encontradas P_3, P_4 y P_5.

Finalmente, vaciamos la pila, como se muestra en la Figura 3.5(g), ya que hemos revisado todos los elementos de $N([2, A])$, $N([1, B])$ y $N(s)$. Notamos que el vértice t no fue uno de los vértices visitados por MDFS, pues nunca fue insertado en la pila, así que no existe en G_M una trayectoria original de s a t y por el Teorema 2.2, no existe una trayectoria M-aumentante en G y por tanto M es máximo. La Figura 3.6 muestra el árbol construido por MDFS para este ejemplo.

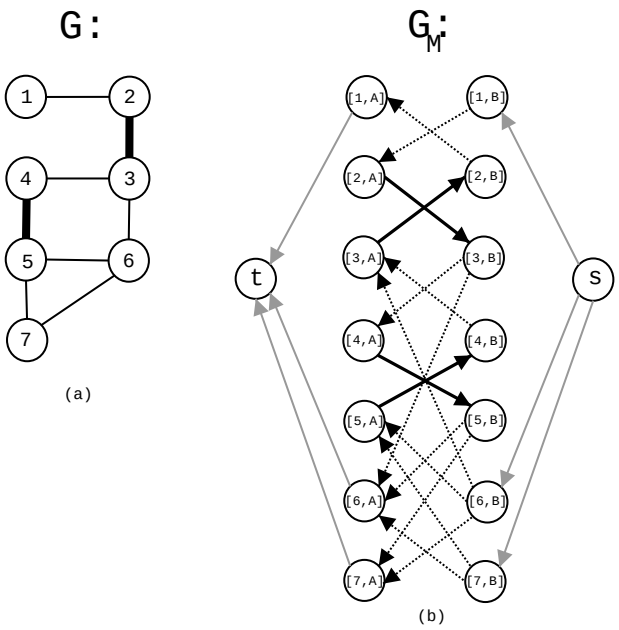

Figura 3.7: Gráficas para el Ejemplo 2

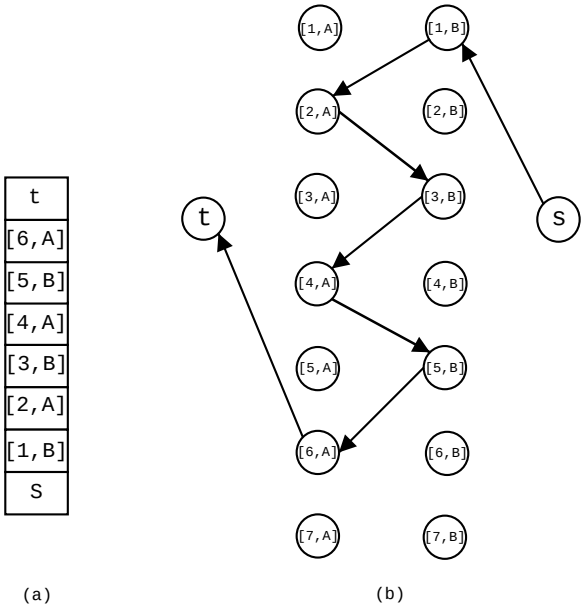

Figura 3.8: Aplicación de MDFS para el Ejemplo 2

Ejemplo 2. Consideremos la gráfica G de la Figura 3.7(a) con el apareamiento indicado. La gráfica G_M correspondiente obtenida a partir de G y su apareamiento se muestra en la Figura 3.7(b).

La pila manejada en la ejecución y el árbol construido por MDFS pueden visualizarse en la Figura 3.8(a) y 3.8(b), respectivamente.

Este ejemplo es más sencillo que el primero, pues con los arcos $(s, [1, B])$, $([1, B], [2, A])$, $([2, A], [3, B])$, $([3, B], [4, A])$, $([4, A], [5, B])$ y $([5, B], [6, A])$, se construye una trayectoria original.

Finalmente, se toma el arco $([6, A], t)$ y se completa una trayectoria original de s a t en G_M. Así, por el Teorema 2.2, se tiene una trayectoria M-aumentante en G, a saber, $P : 1, 2, 3, 4, 5, 6$.

En el siguiente capítulo nos centraremos en la justificación de este algoritmo, la cual consiste esencialmente en verificar formalmente que el algoritmo MDFS construye trayectorias originales en la gráfica dirigida G_M, para cualquier gráfica G y cualquier apareamiento M dado en G.

Capítulo 4

Justificación de MDFS

En este capítulo veremos la justificación del algoritmo MDFS. Dicha justificación está basada en la del algoritmo DFS con las diferencias que previamente analizamos.

El objetivo es demostrar que MDFS construye trayectorias originales en todo momento y que, al final, si existe una trayectoria original del vértice s al t en la gráfica G_M, MDFS debe ser capaz de proporcionarla. Para ello, probaremos varios resultados, en los cuales se hacen afirmaciones importantes sobre la ejecución de este algoritmo.

4.1 Resultados

El primer resultado que veremos afirma que la primera operación que viola la originalidad de una trayectoria es una inserción de algún vértice cuya segunda componente es A. Consideramos para estos resultados la gráfica dirigida G_M construida a partir de una gráfica G y un apareamiento M de G.

Teorema 4.1. Sea $([v, A], [w, B]) \in F(G_M)$. Si MDFS construye una trayectoria original de s a $[v, A]$ entonces la inserción del vértice $[w, B]$ en la pila de MDFS mantiene la originalidad de la trayectoria $\big[s, \ldots, [v, A], [w, B]\big]$.

Demostración. Después de la inserción del vértice $[v, A]$ en la pila de MDFS, el algoritmo considera el único arco cuyo extremo inicial es $[v, A]$, a saber, $([v, A], [w, B])$, puesto que este arco, por construcción de G_M, corresponde a una arista de M.

Así, se efectúa la inserción del vértice $[w, B]$ en la pila. Si esta operación violara la originalidad de la trayectoria $\big[s, \ldots, [v, A], [w, B]\big]$ entonces el vértice $[w, A]$ estaría en esta trayectoria y por consiguiente, como $([w, A], [v, B])$ es el único arco cuyo extremo inicial es $[w, A]$, también el vértice $[v, B]$ estaría ahí. Pero esto implicaría que la inserción de $[v, A]$ en la pila no mantendrá la originalidad

de la trayectoria de s a $[v, A]$ construida, contradiciendo la hipótesis.
Por tanto, la inserción del vértice $[w, B]$ en la pila de MDFS mantiene la originalidad de la trayectoria $\big[s, \ldots, [v, A], [w, B]\big]$. □

Observamos que el Teorema 4.1 puede extenderse a aquellos arcos de la forma $(s, [w, B])$ y $([v, A], t)$, con $[w, B], [v, A] \in V(G_M)$. En los primeros porque los vértices $w \in V(G)$ no son M-apareados y por construcción de G_M, los arcos de la forma $(s, [w, B])$ son los únicos cuyo extremo inicial es s y no hay arcos cuyo extremo inicial tenga segunda componente A. Por esta razón, podemos suponer la construcción de trayectorias originales donde al menos aparezca el vértice s y los vértices $[w, B]$ tales que w no es M-apareado.

Con respecto a los arcos de la forma $([v, A], t)$, los vértices $v \in V(G)$ no son M-apareados y por construcción de G_M, esos son los únicos cuyo extremo final es t. Además, lo que buscamos son trayectorias originales de s a t.

Un esquema para la ilustración del Teorema 4.1 se encuentra en la Figura 4.1. En la parte (a) se muestra una trayectoria original de s a $[v, A]$, la cual se extiende en (b) a una trayectoria de s a $[w, B]$.

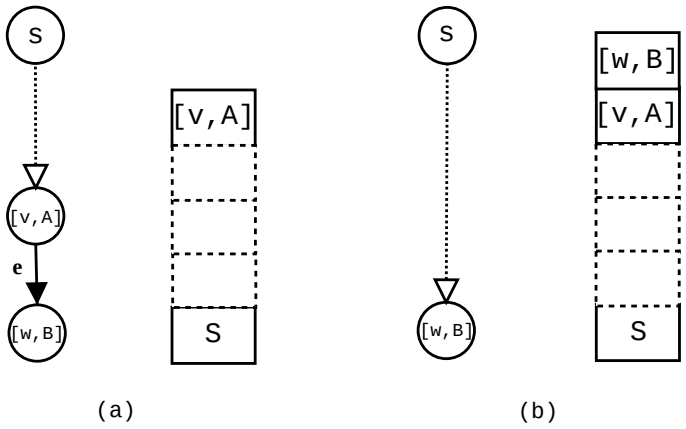

Figura 4.1: Ilustración del Teorema 4.1

El siguiente teorema muestra que en el caso específico en que exista una trayectoria original de s a un vértice $[x, A]$ en G_M, el algoritmo MDFS puede construirla.

Teorema 4.2. Sea $[u, B] \in V(G_M)$ el siguiente vértice a insertar en la pila de MDFS. Supóngase que MDFS construye trayectorias originales al menos hasta la remoción de $[u, B]$. Sea $[x, A] \in V(G_M)$ tal que en el momento cuando se inserta $[u, B]$, hay una trayectoria original $P : [u, B], [v, A], \ldots, [x, A]$ donde ningún vértice de P ni sus opuestos están en la pila. Entonces la inserción de $[x, A]$ ocurre antes de la remoción de $[u, B]$.

Demostración. Sea $P : [u, B], [v, A], [v', B], \ldots, [x, A]$ una trayectoria de mínima longitud tal que $[x, A]$ no ha sido insertado antes de la remoción de $[u, B]$. El algoritmo MDFS debe haber considerado el arco $e = ([u, B], [v, A])$ antes de quitar a $[u, B]$ de la pila. Si se hubiera efectuado la inserción de $[v, A]$, por el Teorema 4.1, la inserción de $[v', B]$ se puede efectuar sin alterar la originalidad de la trayectoria construida, y por la elección de P, el vértice $[x, A]$ es insertado en la pila antes de que sea removido $[v', B]$ y por ende, antes de la remoción de $[u, B]$, contradiciendo la elección de P.

Así, MDFS no efectúa la inserción de $[v, A]$ y como por hipótesis este vértice y su opuesto no están en la pila, MDFS entra en el caso *2.c.i* y ejecuta el ciclo correspondiente. Consideremos el momento en que MDFS termina esa ejecución, es decir, cuando $L_{[v,A]} = \varnothing$.

Sea $[z, A]$ el primer vértice en P que no ha sido insertado en la pila. Tal vértice existe, ya que al menos el vértice $[x, A]$ cumple esa condición. Descomponemos a P de la siguiente manera: $P : [u, B], [v, A], P_1, [z, A], P_2, [x, A]$. Entonces, por el tratamiento del caso *2.c.i*, $[v, A]$ ya había estado en la pila y como actualmente no aparece ahí, se tiene lo siguiente:

- MDFS encuentra la trayectoria $P_1 : [v, A], \ldots, [z, A]$.
- $[z, A]$ no ha sido insertado en la pila.
- $[z, B]$ ha sido removido de la pila, pues por hipótesis $[z, B]$ no estaba en ella cuando se insertó a $[u, B]$. Como $[z, A]$ no fue insertado, MDFS estaba en el caso *2.b*, es decir, $[z, B]$ estaba en la pila en el momento de examinar aquel vértice.

$\therefore [z, A] \in L_{[v,A]}$.
$\therefore L_{[v,A]} \neq \varnothing$, una contradicción con el hecho considerado: $L_{[v,A]} = \varnothing$.
$\therefore$ Todo vértice en P es insertado en la pila.
$\therefore [x, A]$ es insertado antes de la remoción de $[u, B]$.

□

El Teorema 4.2 implica las siguientes dos posibilidades:

- Tanto la inserción del vértice $[x, A]$ como su remoción de la pila se llevan a cabo antes de la inserción del vértice $[u, B]$. En este caso ya se tendría construida una trayectoria original de s a $[x, A]$ en G_M y P no es tomada en cuenta por MDFS.
- Tanto la inserción de $[x, A]$ como su remoción de la pila se llevan a cabo entre la inserción y remoción de $[u, B]$. En este caso P sí es tomada en cuenta, puesto que P forma parte de la trayectoria original de s a $[x, A]$ en G_M construida por MDFS.

En la Figura 4.2 se ilustra el Teorema 4.2. En la parte (a) se muestra una trayectoria original de s a $[u, B]$ la cual se extiende en la parte (b) por medio de otra trayectoria P hasta algún vértice $[x, A]$, si esta trayectoria existe en G_M.

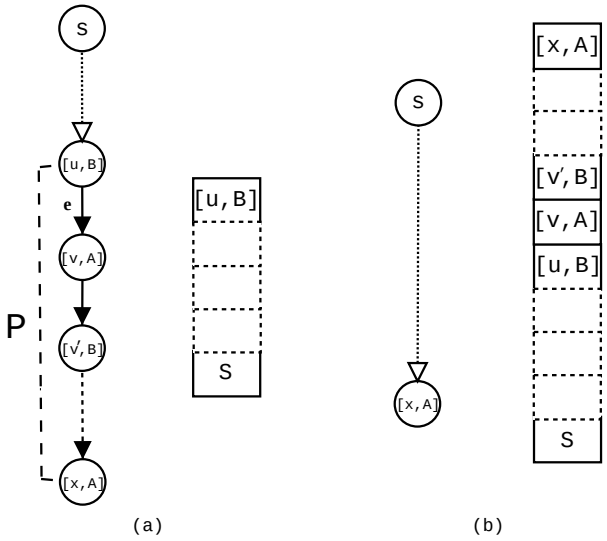

Figura 4.2: Ilustración del Teorema 4.2

Para cada $w \in V(G_M)$ denotamos:

$$r(w) = \begin{cases} [v, \bar{X}] & \text{si} \quad w = [v, X] \\ t & \text{si} \quad w = s \\ s & \text{si} \quad w = t \end{cases}$$

Si $P : w_1, w_2, \ldots, w_k$ es una trayectoria en G_M entonces $(w_i, w_{i+1}) \in F(G_M)$ para $1 \leq i < k$ donde $w_i = [v_i, X]$, $w_{i+1} = [v_{i+1}, \bar{X}]$ y $X \in \{A, B\}$.

Luego, $r(w_i) = [v_i, \bar{X}]$, $r(w_{i+1}) = [v_{i+1}, X]$ y por construcción de G_M, se tiene que $(r(w_{i+1}), r(w_i)) \in F(G_M)$ para $1 \leq i < k$.

Si s está en P entonces $s = w_1$, puesto que el ingrado de s en G_M es cero. De igual forma, si t está en P entonces $t = w_k$, puesto que el exgrado de t en G_M es cero. Cuando se dan estos casos, se tiene que $w_2 = [v_2, B]$ y $w_{k-1} = [v_{k-1}, A]$ por la construcción de G_M. Ahora podemos establecer la siguiente definición:

Definición 4.1. Sea $P : w_1, w_2, \ldots, w_k$ una trayectoria en G_M. La **trayectoria de respaldo** de P, la cual denotamos por $r(P)$, está dada por

$$r(P) : r(w_k), r(w_{k-1}), \ldots, r(w_1).$$

Ejemplo: En la gráfica G_M de la Figura 4.3(a) se ilustra la trayectoria original: $P : s, [v_3, B], [v_1, A], [v_2, B], [v_4, A], t$ y en la Figura 4.3(b) se muestra su

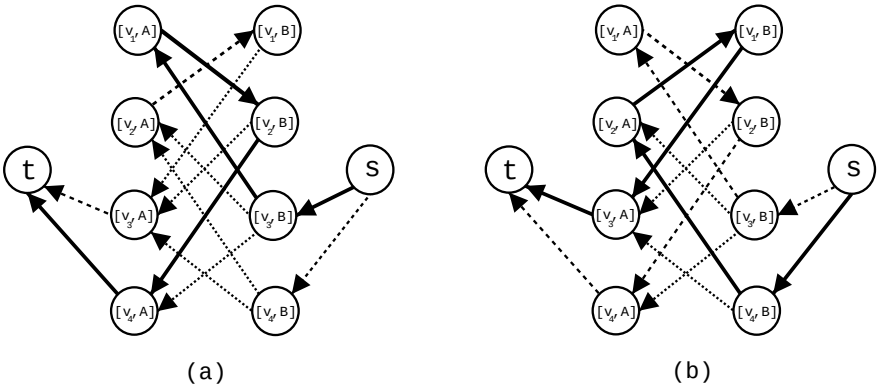

Figura 4.3: Una trayectoria en G_M y su trayectoria de respaldo

trayectoria de respaldo $r(P) : s, [v_4, B], [v_2, A], [v_1, B], [v_3, A], t$.

Observamos que para toda trayectoria P en G_M, se tiene que $r(r(P)) = P$. El siguiente teorema permite determinar una situación en la cual el algoritmo MDFS construye las trayectorias P y $r(P)$, para cualquier trayectoria P en G_M.

Teorema 4.3. Sea $[u, B] \in V(G_M)$ el siguiente vértice a insertar en la pila de MDFS. Supóngase que MDFS construye trayectorias originales al menos hasta la remoción de $[u, B]$. Si existe en G_M una trayectoria original $P : [v, A], \ldots, [w, B]$ donde ningún vértice de P ni sus opuestos están en la pila y además se tiene que $([u, B], [v, A])$, $([w, B], [u, A]) \in F(G_M)$, entonces las inserciones de todos los vértices de P y sus opuestos ocurren antes de la remoción de $[u, B]$.

Demostración. Sea $[z, X]$ en P. Si $X = A$, consideramos la trayectoria P' formada por el arco $([u, B], [v, A])$ y la parte de P con la cual $[z, A]$ es alcanzable desde $[v, A]$. Así, tenemos la trayectoria $P' : [u, B], [v, A], \ldots, [z, A]$. Por el Teorema 4.2, la inserción de $[z, A]$ ocurre antes de la remoción de $[u, B]$.

Si $X = B$, consideramos al único predecesor de $[z, B]$, el cual tiene como segunda componente A. Sea $[y, A]$ tal vértice.
Del caso anterior, sabemos que $[y, A]$ debe ser insertado antes de la remoción de $[u, B]$. Entonces, por el Teorema 4.1, construir una trayectoria original de $[u, B]$ a $[y, A]$ implica que ésta puede extenderse con el vértice $[z, B]$ mateniendo su originalidad, así que $[z, B]$ es insertado antes de que $[y, A]$ sea eliminado de la pila y por tanto, antes de la remoción de $[u, B]$ de la misma.

$\therefore$ Todo vértice $[z, X]$ en P es insertado antes de la remoción de $[u, B]$.

Ahora, para cada $[z, X]$ en P nos fijamos en su opuesto $x = [z, \bar{X}]$. Consideramos la trayectoria de respaldo de P, $r(P) : [w, A], \ldots, [v, B]$ y el arco $([u, B], [w, A])$, el cual está en $F(G_M)$, ya que por hipótesis $([w, B], [u, A]) \in F(G_M)$.

Si $x = [z, A]$ consideramos la parte de $r(P)$ con la cual $[z, A]$ es alcanzable desde $[w, A]$ y el arco $([u, B], [w, A])$. De esta manera, tenemos la trayectoria $P' : [u, B], [w, A], \ldots, [z, A]$. Por el Teorema 4.2, la inserción de $[z, A]$ ocurre antes de la remoción de $[u, B]$.

Si $x = [z, B]$, consideramos su único predecesor, cuya segunda componente es A. Sea $[y, A]$ tal vértice.
Del caso anterior, sabemos que $[y, A]$ debe ser insertado antes de la remoción de $[u, B]$. Entonces, por el Teorema 4.1, construir una trayectoria original de $[u, B]$ a $[y, A]$ implica que ésta puede extenderse con el vértice x mateniendo su originalidad, así que x es insertado antes de que $[y, A]$ sea eliminado de la pila y por tanto, antes de la remoción de $[u, B]$ de la misma.

$\therefore$ Para todo vértice $[z, X]$ en P, $[z, \bar{X}]$ es insertado antes de la remoción de $[u, B]$. □

En la Figura 4.4 se ilustra el Teorema 4.3. En la parte (a) se tiene una trayectoria de s a $[u, B]$ construida por MDFS, se considera el arco $([u, B], [v, A])$ y se construye posteriormente la trayectoria $P : [v, A], \ldots, [w, B]$, lo cual se ilustra en la parte (b). Notamos que el vértice $[u, A]$ no es insertado en la pila ya que su opuesto aparece en ella.

Luego se considera el arco $([u, B], [w, A])$ y se construye la trayectoria de respaldo de P, $r(P) : [w, A], \ldots, [v, B]$, lo cual se observa en la parte (c). Nuevamente, $[u, A]$ no es metido en la pila porque su opuesto aparece ahí.

Como MDFS ha encontrado las trayectorias $Q_1 : [v, A], P, [w, B], [u, A]$ y
$Q_2 : [w, A], r(P), [v, B], [u, A]$ en las cuales no aparece $[u, B]$, cuando este vértice es retirado de la pila, se tiene que $L_{[w,A]} = L_{[v,A]} = \{[u, A]\}$ y puesto que $[u, B]$ es predecesor de los vértices $[v, A]$ y $[w, A]$, entonces $[u, B]$ se elimina de la pila después de que ellos han sido eliminados.

De esta forma observamos que $\forall [u, A] \in V(G_M)$, bajo las condiciones del Teorema 4.3, si $|L_{[u,A]}| > 0$, entonces $[u, A]$ ha sido insertado y eliminado de la pila durante la ejecución del algoritmo MDFS.

4.2 Justificación del algoritmo

A partir de los resultados vistos en la sección anterior, podemos establecer las condiciones que constituyen los invariantes en la ejecución del algoritmo MDFS.

Observamos que el único de ellos que interesa para la justificación de MDFS es el primero. Sin embargo, necesitamos también la prueba de los otros dos para tener una implementación eficiente del algoritmo, la cual revisaremos más

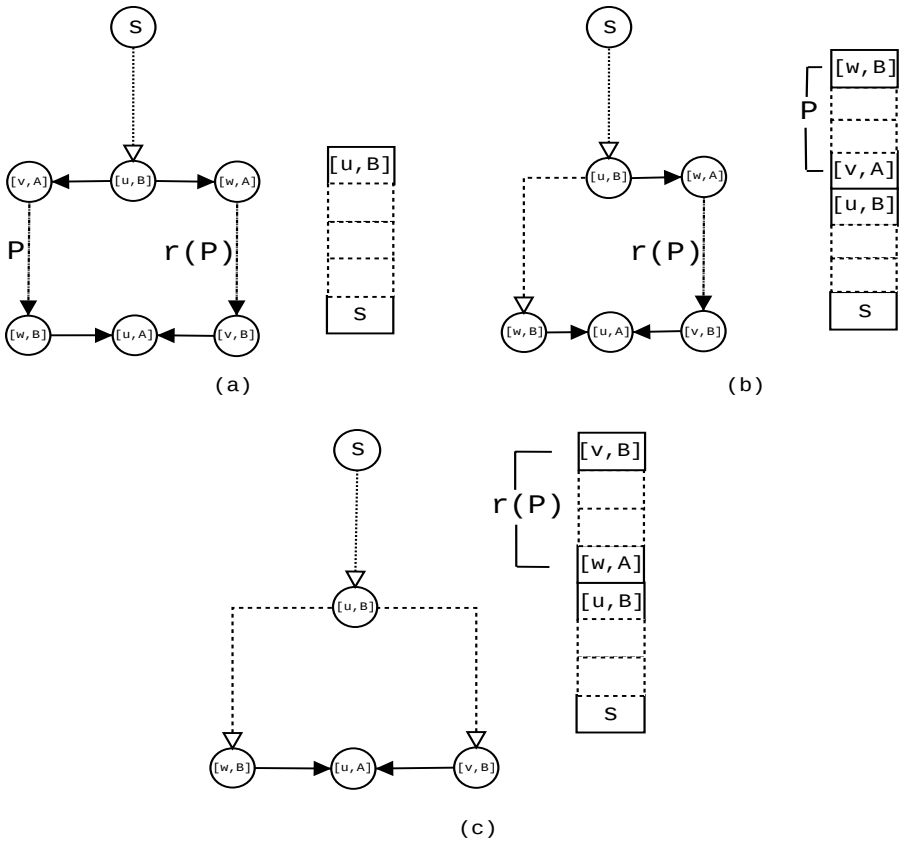

Figura 4.4: Ilustración del Teorema 4.3

adelante. Además, la prueba del primer invariante será más fácil si probamos todos los invariantes de manera simultánea.

Para cada vértice $[w, X] \in V(G_M)$, donde $X \in \{A, B\}$, denotamos por $K_{[w,X]}$ al conjunto de vértices que están en la pila de MDFS cuando $[w, X]$ es colocado en el tope de la misma.

Teorema 4.4. MDFS satisface los siguientes invariantes:

1. MDFS construye solamente trayectorias originales.
2. $|L_{[w,A]}| \leq 1$, $\forall\ [w, A] \in V(G_M)$.
3. Si se produce la asignación $L_{[w,A]} := [u, A]$, entonces después de la inserción del vértice $[u, A]$, siempre se cumple que $L_{[w,A]} = L_{[u,A]}$.

Demostración. Consideramos el primer momento en la ejecución de MDFS en el cual no se cumple alguno de los tres invariantes. Partimos entonces la

demostración en tres casos:

Caso 1. El primer invariante no se cumple, es decir, en algún momento MDFS construye una trayectoria que no es original.

Solamente una inserción en la pila de MDFS puede alterar la originalidad de una trayectoria; sin embargo, no puede afectar el cumplimiento de los otros dos invariantes dada la definición de los conjuntos $L_{[w,A]}$ para cada $[w, A] \in V(G_M)$ y además, una inserción puede reducir la cardinalidad de esos conjuntos, mas no aumentarla.

El Teorema 4.1 implica que la primera inserción que altera la originalidad de una trayectoria es la de algún vértice cuya segunda componente es A, es decir, al considerar un arco $([v, B], [w, A]) \in F(G_M)$ que corresponde a una arista $vw \in E(G) \setminus M$.
Si $[w, A]$ no ha sido marcado como visitado, se aplica el caso $2.c.ii$ de MDFS y entonces $[w, A]$ es introducido a la pila. Para que la trayectoria construida no sea original tiene que presentarse la siguiente situación, ilustrada en la Figura 4.5(a):
En la trayectoria de búsqueda actual hay una parte de ella, llamémosle Q, que contiene a $[w, B]$ y como este vértice no está en la pila, Q fue anexada a aquella trayectoria por una aplicación del caso $2.c.i$, gracias a la inserción de un vértice $[u, A] \in V(G_M)$. Ya que estamos considerando la primera ocasión en que no se cumple este invariante, $[u, B]$ no está en Q y dada la necesidad en su momento de la aplicación del caso $2.c.i$, $[u, B]$ fue insertado en la pila antes de que Q fuera explorada y por tanto $[w, B]$ ya fue insertado y removido. Como la trayectoria construida entre $[u, B]$ y el predecesor de $[u, A]$ en Q satisfacía las hipótesis del Teorema 4.3, en particular $[w, A]$ fue insertado antes de la remoción de $[u, B]$, contradiciendo el supuesto de que $[w, A]$ no ha sido marcado como visitado.

Por tanto, podemos suponer que $[w, A]$ ha sido marcado como visitado. De este modo, se aplica el caso $2.c.i$ de MDFS, tomando un vértice $[u, A] \in L_{[w,A]}$ para insertarlo en la pila. Por el segundo invariante, $|L_{[w,A]}| \leq 1$, así que $L_{[w,A]} = [u, A]$, por tanto, MDFS extiende la trayectoria actual de búsqueda por medio de una trayectoria $Q : [w, A], \ldots, [u, A]$ de la cual sólo se inserta este último vértice. Por definición de $L_{[w,A]}$ y el Teorema 4.3, todos los vértices de Q han sido insertados y removidos de la pila, así como sus vértices opuestos.

Por tanto, la única situación, ilustrada en la Figura 4.5(b), en la cual la inserción de $[u, A]$ puede alterar la originalidad de la trayectoria actual es la siguiente:
Hay un vértice $[p, X]$ en Q y una parte de la trayectoria actual de búsqueda Q' que es generada por una aplicación del caso $2.c.i$ tal que $[p, X] \in V(Q')$ o $[p, \bar{X}] \in V(Q')$.

Como un arco correspondiente al apareamiento actual tiene la característica de que el extremo inicial determina al extremo final, podemos elegir $[p, X]$ tal

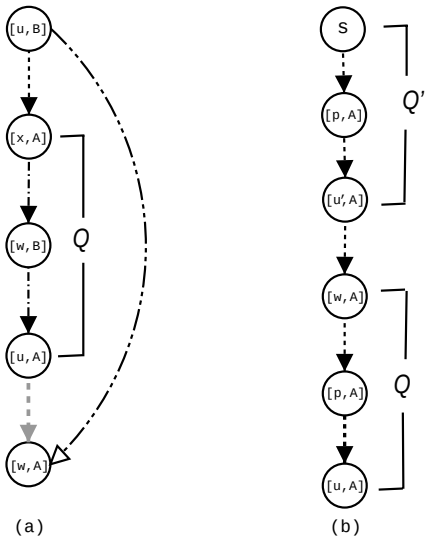

Figura 4.5: Ilustración del Caso 1 para el Teorema 4.4

que $[p, A] \in V(Q')$.

Consideramos el vértice $[u', A]$ cuya inserción provoca la anexión de Q' a la trayectoria actual de búsqueda. Como $[p, A]$ está en Q', por definición de $L_{[p,A]}$ y el segundo invariante, antes de la inserción de $[u', A]$, se tiene que $L_{[p,A]} = [u', A]$. Luego, por el tercer invariante, después de la inserción de $[u', A]$ en la pila, siempre se cumple que $L_{[p,A]} = L_{[u',A]}$. Además, por la elección de $[p, A]$, se tiene en algún momento que $L_{[p,A]} = [u, A]$. Así, se tiene que $L_{[u',A]} = [u, A]$ y por tanto $[u', A]$ tendría que haber sido removido de la pila, pero esto contradice el hecho de que $[u', A]$ está en ella.
$\therefore$ No existe una situación en la cual la inserción de un vértice con segunda componente A altere la originalidad de una trayectoria.
$\therefore$ El primer invariante se cumple.

Caso 2. El segundo invariante no se cumple, es decir, en algún momento uno de los conjuntos $L_{[w,A]}$, con $[w, A] \in V(G_M)$, tiene al menos dos elementos.

Entonces existen $[p_1, A], [p_2, A] \in V(G_M)$ tales que $L_{[w,A]} = \{[p_1, A]\}$ antes de la remoción del opuesto de $[p_2, A]$ y luego de ella, se tiene que el conjunto $L_{[w,A]} = \{[p_1, A], [p_2, A]\}$; por la definición de este conjunto, el algoritmo MDFS ha encontrado dos trayectorias $Q : [p_1, B], \ldots, [p_1, A]$ y $Q' : [p_2, B], \ldots, [p_2, A]$ con $[w, A] \in V(Q) \cap V(Q')$.

Si MDFS encontró Q' después de la remoción de $[p_1, B]$, entonces la única forma en la cual $[w, A]$ pudo ser agregado a Q' es por la aplicación del caso 2.*c.i* de MDFS con respecto a dos vértices $[u, A], [v, A] \in V(G_M)$ tales que

$[u, A] \in L_{[v,A]}$, luego, $[v, A]$ tuvo que ser insertado en la pila y por ende, la trayectoria actual de búsqueda fue extendida con la trayectoria $\bar{Q} : [v, A], \ldots, [u, A]$ que también contiene a $[w, A]$. Así que antes de la inserción de $[u, A]$ se tuvo que $[u, A] \in L_{[w,A]}$ y se llevó a cabo después de la remoción de $[p_1, B]$, por tanto, entre ambas operaciones $[u, A], [p_1, A] \in L_{[w,A]}$, contradiciendo el supuesto inicial de que por primera vez no se satisfacía este invariante.

Por tanto, MDFS encontró Q' antes de la remoción de $[p_1, B]$. Notamos que $[p_1, B]$ no está en Q', de lo contrario, por el Teorema 4.3, $[p_1, A]$ sería insertado antes de la remoción de $[p_2, B]$ y por tanto $[p_1, A] \notin L_{[w,A]}$ después de la remoción de $[p_2, B]$, una contradicción con el supuesto inicial.
Sea $[x, A]$ el primer vértice en Q' tal que $[x, A] \in V(Q)$ o $[x, B] \in V(Q)$, el cual existe dado que $[w, A] \in V(Q) \cap V(Q')$. Podemos ver a Q' como $Q' : [p_2, B], \ldots, [x, A], \ldots, [p_2, A]$ y tenemos dos casos:

- $[x, A] \in Q$. Descomponemos la trayectoria Q como:
  $Q : [p_1, B], \ldots, [x, A], \ldots, [p_1, A]$ y consideramos la trayectoria
  $R : [p_2, B], \ldots, [x, A], \ldots, [p_1, A]$ formada por la parte de Q' desde $[p_2, B]$ hasta $[x, A]$ y la parte de Q desde $[x, A]$ hasta $[p_1, A]$.

- $[x, B] \in Q$. Descomponemos la trayectoria Q como:
  $Q : [p_1, B], \ldots, [x, B], \ldots, [p_1, A]$ y consideramos la trayectoria
  $R : [p_2, B], \ldots, [x, A], \ldots, [p_1, A]$ formada por la parte de Q' desde $[p_2, B]$ hasta $[x, A]$ y el resto, tomando la trayectoria de respaldo de la parte de Q desde $[p_1, B]$ hasta $[x, B]$, la cual va por construcción de G_M desde $[x, A]$ hasta $[p_1, A]$.

En cualquier caso, la trayectoria R considerada satisface las hipótesis del Teorema 4.2, por tanto, $[p_1, A]$ es insertado antes de la remoción de $[p_2, B]$ y como $[w, A]$ está en la intersección de Q y Q', no puede pasar que $[p_1, A]$ esté en el conjunto $L_{[w,A]}$ después de la remoción de $[p_2, B]$, contradiciendo el supuesto inicial, por lo que no pueden darse las condiciones que éste sostiene.
$\therefore$ El segundo invariante se cumple.

Caso 3. El tercer invariante no se cumple, es decir, después de la asignación $L_{[w,A]} := [u, A]$ y la inserción de $[u, A]$, hay un momento en el cual se tiene que $L_{[w,A]} \neq L_{[u,A]}$.

Supongamos que se efectúa la asignación $L_{[w,A]} := [u, A]$. Entonces, luego de la inserción de $[u, A]$, por definición de $L_{[w,A]}$ y $L_{[u,A]}$, se tiene que $L_{[w,A]} = L_{[u,A]} = \varnothing$. La única forma en la cual puede modificarse esta igualdad es por medio de la remoción de un vértice con segunda componente B. Sea $\mathsf{pop}([p, B])$ la siguiente remoción que incrementa la cardinalidad de $L_{[w,A]}$ o $L_{[u,A]}$.

Sea $K' = K_{[w,A]} \cap K_{[u,A]}$. Por la definición de $L_{[w,A]}$, se tiene que $[u, B] \in K_{[w,A]}$ y además, al ser insertado $[u, A]$, también se tiene que $[u, B] \notin K_{[u,A]}$, por el primer invariante y por tanto $[u, B] \in K_{[w,A]} \setminus K' = K_{[w,A]} \setminus K_{[u,A]}$.

De acuerdo a la posición de $[p, B]$ con respecto a $K_{[w,A]}$ y a $K_{[u,A]}$, distinguimos tres casos:

- $[p, B] \in K_{[w,A]} \setminus K_{[u,A]}$. Esto implica que $[p, B]$ en particular no está en $K_{[u,A]}$ y por tanto es removido de la pila antes de la inserción de $[u, A]$, contradiciendo la elección de $[p, B]$.

- $[p, B] \in K_{[u,A]} \setminus K_{[w,A]}$. Como $[u, B] \in K_{[w,A]} \setminus K_{[u,A]}$, existe un primer vértice $[q, B] \in K_{[w,A]} \setminus K_{[u,A]}$ tal que $[q, A] \in K_{[u,A]} \setminus K_{[p,B]}$. Esto se ilustra en la Figura 4.6(a).

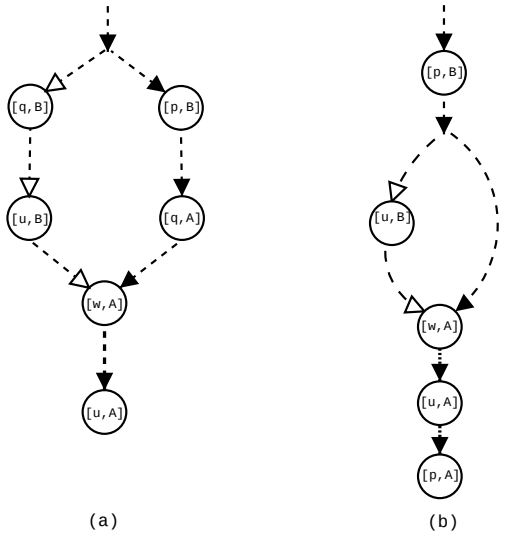

Figura 4.6: Ilustración del Caso 3 para el Teorema 4.4

  Consideramos la trayectoria de respaldo de la trayectoria de $[p, B]$ a $[q, A]$. Tal trayectoria, que por construcción de G_M va desde $[q, B]$ hasta $[p, A]$, satisface las hipótesis del Teorema 4.2, por lo cual, la inserción de $[p, A]$ ocurre antes de la remoción de $[q, B]$. Como $[q, B] \in K_{[w,A]} \setminus K_{[u,A]}$, la inserción de $[p, A]$ se efectúa antes de la remoción de $[p, B]$ y entonces esta última operación no puede incrementar las cardinalidades de los conjuntos $L_{[w,A]}$ o $L_{[u,A]}$, contradiciendo la elección de $[p, B]$.

- $[p, B] \in K'$. Sea $[q, B] \in K'$ el vértice más cercano al tope de K' para el cual $[q, A]$ no ha sido insertado cuando MDFS inserta el vértice $[u, A]$ en la pila. Podemos tomar tal vértice ya que al menos $[p, B]$ tiene esa propiedad. Esto se ilustra en la Figura 4.6(b).

  Consideramos la trayectoria de respaldo a aquella construida de $[q, B]$ a $[u, B]$. Tal trayectoria, que por construcción de G_M va desde $[u, A]$ hasta $[q, A]$, no contiene a $[q, B]$; por tanto, después de la remoción de $[q, B]$, por

el segundo invariante, se tiene que $L_{[u,A]} = [q, A]$, pues por el Teorema 4.3, MDFS encuentra esa trayectoria.

Por el supuesto inicial, $[q, B] = [p, B]$. Como MDFS encuentra también una trayectoria de $[w, A]$ a $[u, A]$ que no contiene a $[q, B]$, se tiene que $L_{[w,A]} = [q, A] = [p, A]$, por el segundo invariante. Por tanto, tenemos que $L_{[u,A]} = [q, A] = L_{[w,A]}$, lo cual implica que los conjuntos $L_{[u,A]}$ y $L_{[w,A]}$, bajo las condiciones dadas, no difieren en elementos.
$\therefore$ Se cumple el tercer invariante.

□

Con el Teorema 4.4 podemos concluir que MDFS construye solamente trayectorias originales en G_M. En el siguiente resultado retomamos este hecho junto con la capacidad de MDFS para construir una trayectoria de s a t, si es que ésta existe.

Teorema 4.5. MDFS construye solamente trayectorias originales en G_M. Además, si existe una trayectoria original de s a t en G_M, MDFS es capaz de construirla.

Demostración. La primera parte es una consecuencia del primer invariante del Teorema 4.4.

Ahora supongamos que existe una trayectoria original P de s a t en G_M, $P : s, [v'_o, B], [v_1, A], [v'_1, B], \ldots, [v'_{r-1}, B], [v_r, A], t$. Es claro por observaciones anteriores que MDFS considera el arco $(s, [v'_0, B])$ y efectúa sin problemas la inserción de $[v'_0, B]$, puesto que v'_0 no es M-apareado. Por tanto, los vértices $[v'_0, B]$ y $[v_r, A]$ satisfacen las hipótesis del Teorema 4.2 con respecto a la trayectoria $P' : [v'_o, B], [v_1, A], [v'_1, B], \ldots, [v'_{r-1}, B], [v_r, A]$, por lo cual, MDFS inserta $[v_r, A]$ antes de la remoción de $[v'_0, B]$ y también, por las observaciones citadas, MDFS inserta el vértice t en la pila inmediatamente después de hacer lo propio con $[v_r, A]$, así que t también es insertado antes de que sea eliminado $[v'_0, B]$ y por ende, antes de la eliminación de s.
$\therefore$ MDFS construye la trayectoria P. □

Observamos que cuando el tope de la pila indica la presencia del vértice t, el algoritmo concluye su ejecución. Como el número de vértices es finito y MDFS inserta en la pila a cada vértice a lo más una vez, se garantiza también que el algoritmo acaba en caso de no existir una trayectoria de s a t en G_M.

Con esto, demostramos que la especificación del algoritmo MDFS es correcta. Sin embargo, falta cubrir algunos detalles basados en los invariantes de MDFS ya vistos y que están relacionados con la eficiencia del algoritmo y las estructuras de datos empleadas para la implementación.

Capítulo 5

Implementación de MDFS

En este capítulo describiremos una forma para implementar el algoritmo MDFS. Haremos énfasis principalmente en dos aspectos fundamentales del algoritmo que no son triviales de implementar: la manipulación de los conjuntos $L_{[w,A]}$, para cada vértice $[w, A]$ de G_M, y la recuperación de la trayectoria construida de s a t en la misma gráfica para nuestro problema de apareamientos máximos.

5.1 Consideraciones iniciales

Conservaremos el árbol T creado por MDFS en todo momento, así como la pila actual, la cual representará a la única trayectoria en T del vértice s, la raíz de T, al vértice que se encuentra en el tope de la pila. Las remociones de vértices en la pila se harán explícitamente y las inserciones se reflejarán tanto en la pila como en T, agregando al vértice insertado como una nueva hoja.

5.2 Manipulación de los conjuntos $L_{[w,A]}$

El segundo y tercer invariantes del Teorema 4.4 resultan ser esenciales para la implementación del algoritmo MDFS. Por definición de los conjuntos $L_{[w,A]}$ para cada $[w, A] \in V(G_M)$, sólo tenemos que actualizarlos después de una inserción o eliminación de algún vértice en la pila de la siguiente manera:

1) Si $L_{[w,A]} = [u, A]$ para algún $[u, A] \in V(G_M)$, después de la inserción de $[u, A]$ se asigna $L_{[w,A]} := \varnothing$.

2) Si existe un $[u, A] \in V(G_M)$ que no ha sido insertado para el cual se ha encontrado una trayectoria $P : [w, A], \ldots, [u, A]$ en la que no aparece su opuesto $[u, B]$, después de la remoción de éste, se asigna $L_{[w,A]} := [u, A]$.

Luego de la remoción de $[u, B]$, es importante encontrar todos aquellos vértices $[w, A] \in V(G_M)$ a los cuales se les pueda aplicar la actualización bajo las condiciones de 2), como en la trayectoria ilustrada en la Figura 5.1. Esta

tarea se puede hacer con un recorrido sobre G_M utilizando el sentido inverso de los arcos desde $[u, A]$ hasta el sucesor directo de $[u, B]$. Sin embargo, mediante el siguiente análisis podremos refinar este procedimiento que denominaremos búsqueda inversa.

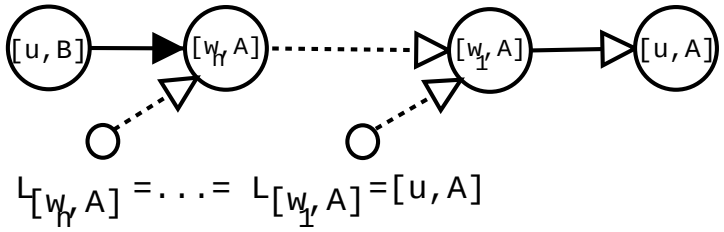

Figura 5.1: Actualización de los conjuntos $L_{[w,A]}$

Primero, caracterizaremos las trayectorias P que no contienen a $[u, B]$ de la forma $P : [w, A], \ldots, [u, A]$. Supongamos que P está constituida por los arcos $a_1, a_2, \ldots, a_t$ en $F(G_M)$, recorridos en ese orden. Entonces a_t es un arco de retroceso débil. Además, si quisieramos recorrer P usando el sentido inverso de sus arcos, encontraríamos secuencias de arcos de árbol seguidos posiblemente por arcos de cruce, de avance o retroceso, luego de otra secuencia de arcos de árbol y así sucesivamente. Esto puede suceder dada la posibilidad de que se presente continuamente el caso $2.c.i$ del algoritmo MDFS.

Consideraremos, después de la remoción de $[u, B]$, los siguientes conjuntos de vértices:

$R_{[u,A]} = \{[v, B] \in V(G_M) \mid ([v, B], [u, A])$ es arco de retroceso débil$\}$

y para algunos vértices $[q, A] \in V(G_M)$:

$E_{[q,A]} = \{[v, B] \in V(G_M) \mid ([v, B], [q, A])$ es arco de cruce, avance o retroceso$\}$.

También es conveniente, dado el tercer invariante, considerar para cada $[q, A] \in V(G_M)$, el conjunto de vértices $[p, A]$ para los cuales en algún momento se tuvo que $L_{[p,A]} = [q, A]$, puesto que durante la búsqueda inversa podríamos no tomar en cuenta partes de algunas trayectorias, generadas también por el caso $2.c.i$ de MDFS. Este conjunto lo denotaremos como $D_{[q,A]}$.

Supongamos que $D_{[q,A]} \subseteq D_{[q',A]}$, entonces cada vértice $[p, A]$ que en algún momento tuvo en su conjunto $L_{[p,A]}$ al vértice $[q, A]$, cumple posteriormente que $L_{[p,A]} = [q', A]$; por el segundo invariante, $[q, A]$ debió ser insertado antes de presentarse esta igualdad, luego por el tercer invariante, se cumple que $L_{[p,A]} = L_{[q,A]}$, es decir, $[q, A] \in D_{[q',A]}$. De aquí se sigue que cuando $[q', A]$ es insertado en la pila, se tiene siempre que $L_{[q,A]} = L_{[q',A]}$.

Por tanto, si $D_{[q,A]} \subseteq D_{[q',A]}$ y conocemos el conjunto $L_{[q',A]}$, entonces automáticamente podemos determinar $L_{[q,A]}$. Por ello, conviene saber si hay conjuntos $D_{[q,A]}$ con el mayor número de elementos posible.

Decimos que el conjunto de vértices $D_{[q,A]}$ es **actual** o está actualizado si no existe $[q', A] \in V(G_M) \setminus \{[q, A]\}$ tal que $D_{[q,A]} \subseteq D_{[q',A]}$. Con esta definición podemos calcular los conjuntos $L_{[p,A]}$ de la siguiente manera:

1) Buscar un vértice $[q, A]$ tal que $[p, A] \in D_{[q,A]}$ y $D_{[q,A]}$ es actual. Si $[q, A]$ existe, verificar si ha sido insertado en la pila en algún momento.
   (a) En caso afirmativo, se tiene que $L_{[p,A]} = \varnothing$.
   (b) En caso negativo, $L_{[p,A]} = [q, A]$.
2) Si no existe tal vértice, entonces $L_{[p,A]} = \varnothing$.

Así, la manipulación de los conjuntos $D_{[q,A]}$ permitirá la solución del primer subproblema. Para una mejor organización de la búsqueda inversa, requerimos saber para cada vértice $[p, A]$ si en algún momento existió una modificación en su conjunto $L_{[p,A]}$, por lo cual definimos el siguiente conjunto:

$L = \{[p, A] \in V(G_M) : L_{[p,A]} \neq \varnothing \text{ previamente}\}$.

Supongamos que $D_{[q,A]}$ y $D_{[q',A]}$ son dos conjuntos actuales que contienen un vértice arbitrario $[p, A]$, entonces $L_{[p,A]} = [q, A]$ y $L_{[p,A]} = [q', A]$. Luego, por el primer invariante tenemos que $[q, A] = [q', A]$. Esta observación implica que cada vértice $[p, A]$ está contenido en a lo más un conjunto $D_{[q,A]}$. Más adelante veremos cuál es la importancia de esto.

5.2.1 Búsqueda Inversa

Ahora describiremos la búsqueda que se lleva a cabo cuando se tiene identificado un vértice $[u, B]$ que será eliminado de la pila de MDFS, con el fin de obtener todas aquellas trayectorias encontradas cuyo vértice final es $[u, A]$.

Tales trayectorias se determinan en varias etapas: en la primera, se identificarán todas aquellas trayectorias que tengan arcos de retroceso débil cuyo extremo final es $[u, A]$ y secuencias de arcos de árbol; a partir de la segunda, se hace lo propio con aquellas que tengan arcos de las categorías restantes. Es decir, en la primera etapa, se consideran los arcos de retroceso débil $([v, B], [u, A])$. Posteriormente, para las etapas $i > 1$, se consideran como puntos de partida los vértices $[v, B] \in E_{[q,A]}$, donde los vértices $[q, A]$ son aquellos para los cuales se determinó que $L_{[q,A]} = [u, A]$ en la etapa $i - 1$. Empezaremos con un vértice $[v, B]$ que cumpla las características mencionadas, siguiendo en sentido inverso los arcos de árbol; en algún momento alcanzaremos al vértice $[u, B]$ que está a punto de salir de la pila. Si en ese proceso encontramos algún $[p, A] \in L$, calculamos el conjunto $D_{[r,A]}$ actual que contiene a $[p, A]$ y saltamos todos los

vértices hasta llegar a $[r, B]$ para continuar. Esto es válido ya que para cada $[x, A] \in D_{[r,A]}$ se tiene que $L_{[x,A]} = L_{[r,A]}$ y por el tercer invariante se tiene forzosamente que $L_{[x,A]} = [u, A]$ también.

En este proceso, vamos a marcar determinados arcos que servirán para la recuperación posterior de la trayectoria de s a t, si ésta existe. Entonces mantendremos una variable $P_{[r,A]}$, para todo $[r, A] \in V(G_M)$ tal que $L_{[r,A]} \neq \varnothing$ por primera vez, para conservar la información referente al arco cuyo extremo inicial está en un conjunto $E_{[q,A]}$ o en $R_{[q,A]}$, e interrumpe la secuencia de arcos de árbol que incluye al arco cuyo extremo final es $[r, A]$. Tan pronto como se determina el arco $P_{[r,A]}$, el vértice $[r, A]$ es agregado a L, por lo cual $[q, A]$ queda deteminado por MDFS sin problema alguno.

5.3 Nueva especificación de MDFS

La solución revisada del primer subproblema para la implementación de MDFS induce una nueva especificación para éste, en términos de los casos que considera y las operaciones en la pila de inserción y eliminación de vértices. Recordamos que en cada paso, MDFS considera un arco $e = ([v, X], [w, Y])$ con $X \in \{A, B\}$, $Y = \bar{X}$ y $[v, X]$ es el elemento ubicado en el tope de la pila. A continuación realizamos un análisis más detallado de cada caso del algoritmo MDFS.

1. Si $Y = B$, entonces $e = ([v, A], [w, B])$ es arco de árbol, por lo cual insertamos $[w, B]$ en la pila e invocamos recursivamente al algoritmo.

2. Si $Y = A$, entonces $e = ([v, B], [w, A])$; tenemos varios subcasos:

   (a) Si el vértice $[w, A]$ está en la pila, entonces e es arco de retroceso y por tanto agregamos a $[v, B]$ al conjunto $E_{[w,A]}$. Notamos que $[w, A] \notin L$, ya que no lo hemos eliminado de la pila.

   (b) Si el vértice $[w, A]$ no aparece en la pila pero $[w, B]$ sí, tenemos dos posibilidades:

      i. Si $[w, A]$ ha estado en la pila antes, entonces e es arco de cruce y por tanto agregamos el vértice $[v, B]$ al conjunto $E_{[w,A]}$. Por el Teorema 4.1, $[w, A] \notin L$.

      ii. Si $[w, A]$ no ha estado nunca en la pila, entonces e es un arco de retroceso débil y por tanto agregamos el vértice $[v, B]$ al conjunto $R_{[w,A]}$.

   (c) Si los vértices $[w, A]$ y $[w, B]$ no están en la pila, tenemos dos posibilidades:

      i. Si $[w, A]$ ha estado en la pila antes, entonces e es arco de avance o de cruce. De cualquier forma, siempre que $L_{[w,A]} \neq \varnothing$, tendremos que almacenar la información de que e deberá ser usado. Ponemos en el tope de la pila junto con el vértice $[v, B]$ el arco

$e = ([v, B], [w, A])$; después introducimos en la pila al vértice que está en $L_{[w,A]}$ e invocamos recursivamente al algoritmo.

Si $L_{[w,A]} = \varnothing$, debemos tomar en cuenta lo siguiente:
Si $[w, A] \in L$, nada hay por hacer, porque el elemento en $L_{[w,A]}$ en algún momento fue insertado en la pila y $L_{[w,A]}$ quedó tal y como está ahora; en otro caso, $L_{[w,A]}$ siempre ha sido vacío y, entonces, por la categoría de e, agregamos a $[v, B]$ al conjunto $E_{[w,A]}$.

ii. Si el vértice $[w, A]$ no ha estado nunca en la pila, entonces e es arco de árbol, por tanto insertamos al vértice $[w, A]$ en la pila e invocamos recursivamente al algoritmo.

Si $[v, B]$ es el vértice hallado en el tope de la pila y $[v, A]$ no ha sido insertado en ella, a partir del momento en que se tienen revisados todos los elementos en $N([v, B])$, conviene realizar una búsqueda inversa para recuperar posibles trayectorias encontradas que terminen en $[v, A]$ y que, dada la presencia de $[v, B]$ en la pila, no fueron construidas por MDFS. Esto lo hacemos precisamente porque $[v, B]$ está a punto de abandonar la pila.

Esta búsqueda se lleva a cabo en varias etapas. En la primera se consideran las trayectorias con arcos de retroceso débil de la forma $([x, B], [v, A])$ y se calculan también los vértices $[y, A]$ tales que $L_{[y,A]} = [v, A]$, para usarlos como puntos de partida en las siguientes etapas de la búsqueda hasta que alcancemos el vértice $[v, B]$. Dichos vértices se tendrán almacenados en un conjunto L_{sig}. En las siguientes etapas se consideran aquellas trayectorias en las cuales hay arcos de las categorías restantes cuyos extremos finales están en L_{sig}.

En todas las etapas se van marcando los arcos que no son de árbol con las variables $P_{[u,A]}$ y se actualiza el conjunto $D_{[v,A]}$ con los vértices que se vayan encontrando. Una vez teminada la aplicación de la búsqueda inversa, se sigue la aplicación del algoritmo de manera normal.

En el Listado 3 se muestra la nueva especificación del procedimiento search del algoritmo MDFS con las ideas ya expuestas. El cálculo de los conjuntos $L_{[w,A]}$ se hace implícitamente con las consideraciones de la sección previa.

En el Listado 4, se muestra la especificación del procedimiento inverseSearch, correspondiente a la búsqueda inversa. El procedimiento interno, recoverPath, recupera vértices a lo largo de una determinada trayectoria que empieza en $[u, B]$ y termina en $[u, A]$, el punto de partida para la búsqueda inversa. Toma como entrada el $([q, B], [u, A])$, que no es de árbol, y el vértice $[x, B]$ que indica donde se detiene el proceso.

Listado 3 Procedimiento search del Algoritmo MDFS

```
procedure search;
 1: if top(K) = [t, B] then
 2:    reconstruir una trayectoria original de s a t;
 3: else
 4:    Sea [v, X] = top(K);
 5:    marcar [v, X] como visitado;
 6:    for all [w, Y] ∈ N([v, X]) do
 7:       if Y = B then
 8:          {Caso 1}
 9:          push([w, B]);
10:          search;
11:       else if [w, A] ∈ K then
12:          {Caso 2.a}
13:          E_[w,A] := E_[w,A] ∪ {[v, X]};
14:       else if [w, B] ∈ K then
15:          {Caso 2.b}
16:          if [w, A] está marcado como visitado then
17:             {Caso 2.b.i}
18:             E_[w,A] := E_[w,A] ∪ {[v, X]};
19:          else
20:             {Caso 2.b.ii}
21:             R_[w,A] := R_[w,A] ∪ {[v, X]};
22:          end if
23:       else if [w, A] está marcado como visitado then
24:          {Caso 2.c.i.}
25:          if L_[w,A] ≠ ∅ then
26:             agregar en K junto con [v, B] el arco ([v, B], [w, A]);
27:             push(L_[w,A]);
28:             L_[w,A] := ∅
29:             search;
30:          else
31:             if [w, A] ∉ L then
32:                E_[w,A] := E_[w,A] ∪ {[v, X]};
33:             end if
34:          end if
35:       else
36:          {Caso 2.c.ii}
37:          push([w, A]);
38:          search;
39:       end if
40:    end for
41:    if X = B y [v, A] no está marcado como visitado then
42:       {Comenzamos la Búsqueda Inversa}
43:       inverseSearch;
44:    end if
45:    pop;
46: end if
```

Listado 4 Procedimiento inverseSearch del Algoritmo MDFS

procedure inverseSearch;

```
1: D_[v,A] := ∅;
2: L_sig := ∅;
3: for all [q, B] ∈ R_[v,A] do
4:    recoverPath(([q,B],[v,A]),[v,B]);
5: end for
6: while L_sig ≠ ∅ do
7:    elegir un vértice [k, A] ∈ L_sig;
8:    L_sig := L_sig \ {[k, A]};
9:    for all [q, B] ∈ E_[k,A] do
10:      recoverPath(([q,B],[k,A]),[v,B]);
11:   end for
12: end while
```

procedure recoverPath(([q,B],[u,A]),[x,B]);

```
1: P_actual := ([q, B], [u, A]);
2: [z, B] := [q, B]; {[z, B] será el punto de partida en el ciclo interno.}
3: while no sea alcanzado [x, B] do
4:    for all [y, A] en la trayectoria de [x, B] a [z, B] do
5:       {Consideramos los arcos con su sentido inverso}
6:       if [y, A] ∈ L then
7:          sea D_[r,A] el conjunto actual que tiene a [y, A];
8:          D_[v,A] := D_[v,A] ∪ D_[r,A];
9:          [z, B] := [r, B];
10:      else
11:         D_[v,A] := D_[v,A] ∪ {[y, A]};
12:         L := L ∪ {[y, A]};
13:         P_[y,A] := P_actual;
14:         L_sig := L_sig ∪ {[y, A]};
15:      end if
16:   end for
17: end while
```

5.3.1 Ejemplo

Consideremos la Gráfica G_M ilustrada en la Figura 5.2. El resultado parcial, hasta el momento en que el vértice $[7, B]$ es descubierto, se muestra en la Figura 5.3(a) junto con la pila actual.

El siguiente arco examinado es $([7, B], [5, A])$, el cual es de retroceso débil, puesto que $[5, B]$ está en la pila. Por esta razón hacemos $R_{[5,A]} = \{[7, B]\}$. Luego, como $[7, A]$ no está visitado y $R_{[7,A]} = \varnothing$, no se realiza la búsqueda inversa y sacamos los vértices $[7, B]$ y $[6, A]$ de la pila. Después se considera el arco $([5, B], [7, A])$ que es de árbol, al igual que el arco $([7, A], [6, B])$. Así tenemos el resultado parcial de la Figura 5.3(b).

Luego se examina el arco $([6, B], [5, A])$, que es de retroceso débil, ya que $[5, A]$ no está visitado y $[5, B]$ está en la pila. Por tanto, $R_{[5,A]} = \{[7, B], [6, B]\}$. El siguiente arco a considerar es $([6, B], [3, A])$, que también es de retroceso débil, ya que $[3, A]$ no está visitado y $[3, B]$ está en la pila. Tenemos, por tanto, que $R_{[3,A]} = \{[6, B]\}$. Como $[6, A]$ ya está visitado, no se lleva a cabo la búsqueda inversa y salen de la pila $[6, B]$ y $[7, A]$.

Hemos revisado todos los elementos de $N([5, B])$ y $[5, A]$ sigue sin ser visitado. Es momento de aplicar una búsqueda inversa partiendo inicialmente de $[5, A]$. Sabemos que $R_{[5,A]} = \{[7, B], [6, B]\}$. Cuando se considera la trayectoria que contiene al vértice $[7, B]$, se tiene lo siguiente:

$D_{[5,A]} = \{[6, A]\}$; $L = \{[6, A]\}$; $P_{[6,A]} = ([7, B], [5, A])$; $L_{sig} = \{[6, A]\}$.

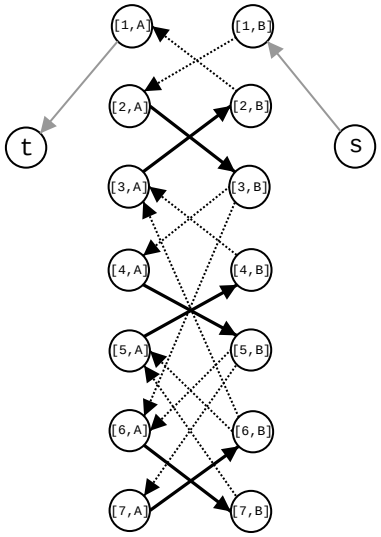

Figura 5.2: Gráfica del Ejemplo

Cuando se considera la trayectoria que contiene al vértice $[6, B]$, se tiene:

$D_{[5,A]} = \{[6, A], [7, A]\}$; $L = \{[6, A], [7, A]\}$;
$P_{[7,A]} = ([6, B], [5, A])$; $L_{sig} = \{[6, A], [7, A]\}$.

La búsqueda inversa se detiene ahí, ya que $E_{[6,A]} = E_{[7,A]} = \varnothing$. Lo que tenemos hasta ahora de manera implícita es que $L_{[6,A]} = L_{[7,A]} = [5, A]$; además, identificamos los arcos que violan la originalidad de las trayectorias encontradas de $[5, B]$ a $[5, A]$, a saber, $P_{[7,A]}$ y $P_{[6,A]}$. Inmediatamente sacamos a $[5, B]$ y $[4, A]$ de la pila. Se considera ahora al arco $([3, B], [6, A])$, pero $[6, A]$ ya está visitado y además, $L_{[6,A]} = [5, A]$, pues $[6, A] \in D_{[5,A]}$, por tanto, insertamos $[5, A]$. El resultado parcial se muestra en la Figura 5.3(c). Notamos que aquí hemos agregado al arco $([3, B], [6, A])$ al lado de $[6, A]$ en la pila, lo cual representa en el árbol construido, con el arco artificial $([3, B], [5, A])_{[6,A]}$, que $[6, A]$ es el primer vértice de la trayectoria desde $[3, B]$ hasta $[5, A]$.

Continuamos desde $[5, A]$. Tenemos el arco de árbol $([5, A], [4, B])$, así que insertamos $[4, B]$. Revisamos ahora el arco $([4, B], [3, A])$, que resulta ser de retroceso débil, por lo cual tenemos que $R_{[3,A]} = \{[6, B], [4, B]\}$. El vértice $[4, A]$ ya está visitado, así que sacamos sin problemas $[4, B]$ y $[5, A]$ de la pila, obteniendo de esta manera el resultado parcial mostrado en la Figura 5.3(d).

Hemos revisado todos los elementos de $N([3, B])$ y $[3, A]$ no está visitado. Es momento de efectuar una búsqueda inversa partiendo de $[3, A]$ y recordamos que $R_{[3,A]} = \{[6, B], [4, B]\}$. Así, cuando se considera la trayectoria que contiene a $[6, B]$ se tiene lo siguiente:

$D_{[3,A]} = D_{[5,A]} = \{[6, A], [7, A]\}$ ya que $[7, A] \in L$ y $[7, A] \in D_{[5,A]}$.

Continuamos la búsqueda inversa desde $[5, B]$ y tenemos que:

$D_{[3,A]} = \{[6, A], [7, A], [4, A]\}$; $L = \{[6, A], [7, A], [4, A]\}$;
$P_{[4,A]} = ([6, B], [3, A])$; $L_{sig} = \{[4, A]\}$.

Después se considera la trayectoria que contiene a $[4, B]$ y se tiene que:

$D_{[3,A]} = \{[6, A], [7, A], [4, A], [5, A]\}$; $L = \{[6, A], [7, A], [4, A], [5, A]\}$;
$P_{[5,A]} = ([4, B], [3, A])$; $L_{sig} = \{[4, A], [5, A]\}$.

Como $E_{[4,A]} = E_{[5,A]} = \varnothing$, la búsqueda inversa termina ahí, pues implícitamente tenemos que $L_{[n,A]} = [3, A]$ con $n = 4, \ldots, 7$ y que los arcos $P_{[4,A]}$ y $P_{[5,A]}$ son los que violan la originalidad de las trayectorias de $[3, B]$ a $[3, A]$. Inmediatamente sacamos a $[3, B]$ de la pila, así como a $[2, A]$ y $[1, B]$, puesto que $R_{[1,A]} = \varnothing$. Por último, sacamos a s de la pila y concluimos que no existe trayectoria original de s a t en G_M, ya que t no fue visitado.

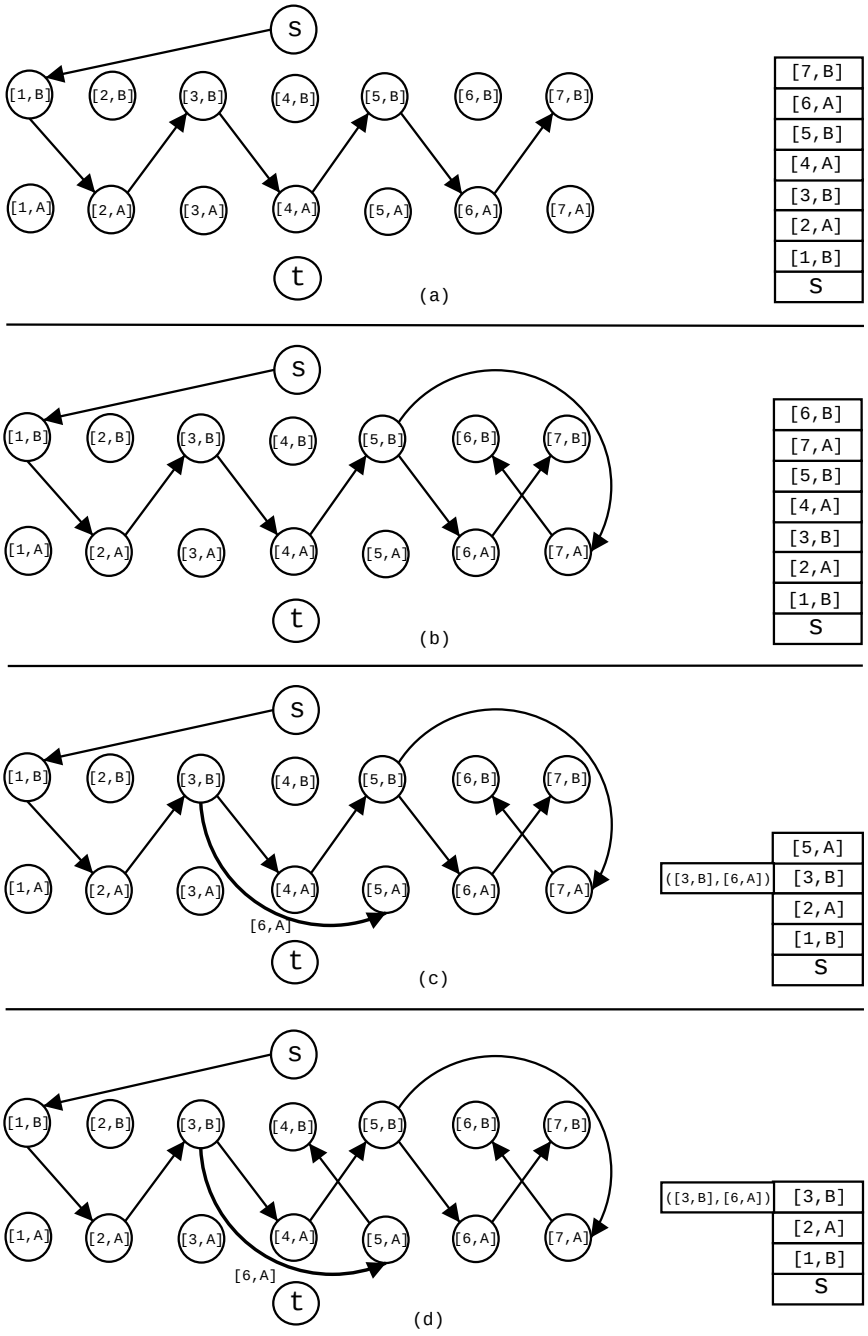

Figura 5.3: Aplicación de MDFS con la nueva especificación

5.4 Reconstrucción de la trayectoria original

Haremos algunas consideraciones adicionales sobre la implementación de MDFS antes de atacar la reconstrucción de la trayectoria original. Usaremos una etiqueta $parent(v)$ para obtener el único predecesor directo de cada vértice v en el árbol T de MDFS. Actualizando esta etiqueta cada vez que se agregue un vértice a la pila haremos la construcción implícita de T.

Recordamos que, por lo general, en la pila de MDFS colocamos vértices, salvo cuando ocurre el caso $2.c.i$, en el cual agregamos arcos junto con vértices. En este caso decimos que el vértice insertado está expandido por el arco que lo acompaña en la pila. Usaremos una etiqueta $expanded(v)$ para indicar si el vértice v está expandido por algún arco o no. Como la reconstrucción la vamos a hacer siguiendo las etiquetas *parent*, parece que conviene dejar las cosas como estaban y sólo agregar vértices a la pila, indicando que estén o no expandidos según sea el caso con las etiquetas *expanded*.

Definimos ahora el proceso para reconstruir la trayectoria original de s a t en G_M, el cual se realiza luego de detectar en el tope de la pila al vértice t; se tienen dos casos:

1. Si la trayectoria sólo consiste de vértices que no están expandidos, simplemente seguimos las etiquetas *parent*.

2. Si encontramos en algún momento un vértice $[v, B]$ expandido por el arco $([v, B], [w, A])$, esto implica que el vértice $[u, A]$ tal que $parent([u, A]) = ([v, B])$, fue insertado por una aplicación del caso $2.c.i$, es decir, existe una trayectoria original $Q : [w, A], \ldots, [u, A]$, la cual vamos a reconstruir de la siguiente manera:

   (a) Consideramos el arco $P_{[w,A]}$ que interrumpe la secuencia de arcos de árbol que incluye al arco con extremo final $[w, A]$. Sea $P^i_{[w,A]}$ el extremo inicial de $P_{[w,A]}$ y sea $P^f_{[w,A]}$ su extremo final.

   (b) Tomamos como vértice inicial a $[w, A]$ y reconstruimos la parte de Q desde $[w, A]$ hasta $P^i_{[w,A]}$, siguiendo nuevamente las etiquetas *parent*.

      i. Si $P^f_{[w,A]} = [u, A]$, entonces tenemos ya reconstruida la trayectoria Q.

      ii. Si no, repetimos el proceso ahora tomando como vértice inicial a $P^f_{[w,A]}$ hasta recuperar Q completamente.

   (c) Para integrar todas las partes de Q, necesitamos invertir el orden en el que fueron recuperadas, puesto que comenzamos con $[w, A]$ y terminamos con la parte de Q que termina en $[u, A]$.

Este método se ilustra en el Listado 5 y trabaja en forma recursiva. Recibe como entrada dos vértices $[t, X]$ y $[s, Y]$, que son los extremos final e inicial, res-

pectivamente, de la parte de la trayectoria a reconstruir. La primera invocación a este método se hace con los vértices t y s de G_M.

Listado 5 Procedimiento para reconstruir una trayectoria original de s a t

procedure reconstructPath(G_M,$[t,X]$,$[s,Y]$);
Entrada: La gráfica dirigida G_M y los vértices final e inicial, $[t,X]$ y $[s,Y]$, respectivamente.
Salida: La trayectoria original $\bar{P} : [s,Y], \ldots, [t,X]$.

```
1:  P := ∅;
2:  if t = s then
3:     P = P ∪ {[s,Y]};
4:  else
5:     if expanded(parent([t,X])) = false then
6:        P := P ∪ {[t,X]} ∪ reconstructPath(G_M, parent([t,X]), [s,Y]);
7:     else
8:        Sea ([v,B],[w,A]) el arco que expande a parent([t,X]);
9:        Q_1 := reconstructPath(G_M, [t,X], P^f_[w,A]);
10:       Q_2 := reconstructPath(G_M, P^i_[w,A], [w,A]);
11:       Q_3 := reconstructPath(G_M, [v,B], [s,Y]);
12:       P := P ∪ Q_1 ∪ Q_2 ∪ Q_3;
13:    end if
14: end if
15: P̄ := P;
```

5.5 Especificación detallada de MDFS

Vamos a retomar algunas de las observaciones hechas anteriormente para especificar de forma más detallada el algoritmo y con ello generar una implementación.

Sea $[v,X] \in V(G_M)$. Utilizaremos varias etiquetas, además de las ya mencionadas, las cuales son:

- $inStack([v,X])$. Indica si $[v,X]$ está actualmente colocado en la pila o no.
- $visited([v,X])$. Indica si $[v,X]$ ha sido marcado como visitado y por tanto, ha estado en la pila o está ahí actualmente.
- $expArc([v,X])$. Marca una referencia al arco que expande a $[v,X]$, si es que existe.
- $inL([v,X])$. Determina si $[v,X] \in L$ sin tener que hacer una búsqueda en todo este conjunto.

Definición 5.1. Sean $[v,X],[w,Y] \in V(G_M)$. Decimos que ambos vértices son iguales si $v = w$ y $X = Y$. Esto lo denotamos como $[v,X] = [w,Y]$.

Tenemos que un vértice $[w, A] \in V(G_M)$ a lo más está contenido en un conjunto $D_{[u,A]}$. Por esta razón, para representar estos conjuntos nos basaremos en el tipo de datos abstracto Conjuntos Ajenos usando compresión de trayectorias. Como en la búsqueda inversa solamente se actualiza un solo conjunto $D_{[u,A]}$, todas las uniones de conjuntos que se realizan se traducen en modificaciones sobre este conjunto, por lo cual utilizamos unión simple. La forma de determinar los conjuntos $L_{[w,A]}$ se reduce a lo siguiente:

Primero obtenemos el conjunto $D_{[u,A]}$ donde se encuentra $[w, A]$. Esto mediante la operación find.

- Si find regresa un vértice $[u, A]$ distinto a $[w, A]$ y además $[u, A]$ no está marcado como visitado entonces $L_{[w,A]} = [u, A]$.
- En caso contrario, se tiene que $L_{[w,A]} = \varnothing$.

Todos los conjuntos, excepto aquellos de la forma $D_{[u,A]}$, los implementamos como listas ligadas, de modo que la inserción y eliminación de elementos se hagan siempre en la primera posición.

En el Listado 6 se muestra la especificación completa de MDFS. Identificamos el vértice s por $[s, A]$ y al vértice t por $[t, B]$.

En el Listado 7 tenemos la especificación detallada del procedimiento search del algoritmo MDFS. Puesto que está diseñado en forma recursiva, establecemos condiciones con la instrucción **return** para detener la recursión, que esencialmente consisten en verificar si $found = true$, es decir, si ya tenemos a t en el tope de la pila. En el Listado 8 se define el procedimiento detallado de búsqueda inversa.

Listado 6 Algoritmo MDFS detallado

Entrada: La gráfica dirigida G_M construida a partir de una gráfica G y un apareamiento M.
Salida: Una variable lógica $found$ que indica si existe una trayectoria original en G_M de s a t.

```
1: parent([s, A]) := [s, A];
2: push([s, A]);
3: inStack([s, A]) := true;
4: found := false;
5: while K ≠ ∅ y found = false do
6:    search;
7: end while
```

Sólo nos resta analizar el tiempo de ejecución de MDFS. Primero, veamos el proceso inicial para visitar vértices. Por cada $v \in V(G_M)$ se revisan todos los arcos en los cuales v es extremo inicial. Puede ocurrir que se realice una operación find por cada uno de esos arcos. Entonces, en el peor caso, se pueden

Listado 7 Procedimiento search detallado del Algoritmo MDFS

```
procedure search;
 1: [v, X] := top(K);
 2: if [v, X] = [t, B] then
 3:    found := true;  visited([v, X]) := true;
 4:    return;
 5: else
 6:    visited([v, X]) := true;
 7:    for all [w, Y] ∈ N([v, X]) do
 8:       if Y = B then
 9:          parent([w, B]) := [v, A];
10:          push([w, B]);  inStack([w, B]) := true;
11:          search;
12:          if found = true then
13:             return;
14:          end if
15:       else if inStack([w, A]) = true then
16:          E_[w,A] := E_[w,A] ∪ {[v, X]};
17:       else if inStack([w, B]) = true then
18:          if visited([w, A]) = true then
19:             E_[w,A] := E_[w,A] ∪ {[v, X]};
20:          else
21:             R_[w,A] := R_[w,A] ∪ {[v, X]};
22:          end if
23:       else if visited([w, A]) = true then
24:          [u, A] := find([w, A]);
25:          if u ≠ w y visited([u, A]) = false then
26:             expArc([v, B]) := ([v, B], [w, A]);  expanded([v, B]) := true;
27:             parent([u, A]) := [v, B];
28:             push([u, A]);  inStack([u, A]) := true;
29:             search;
30:             if found = true then
31:                return;
32:             end if
33:          else
34:             if inL([w, A]) = false then
35:                E_[w,A] := E_[w,A] ∪ {[v, X]};
36:             end if
37:          end if
38:       else
39:          parent([w, A]) := [v, B];
40:          push([w, A]);  inStack([w, A]) := true;
41:          search;
42:          if found = true then
43:             return;
44:          end if
45:       end if
46:    end for
47:    if X = B y visited([v, A]) = false then
48:       inverseSearch;
49:    end if
50:    pop;  inStack([v, B]) := false;
51: end if
```

Listado 8 Procedimiento inverseSearch detallado del Algoritmo MDFS

procedure inverseSearch;

```
1: D_[v,A] := ∅;
2: L_sig := ∅;
3: for all [q, B] ∈ R_[v,A] do
4:    recoverPath(([q,B],[v,A]),[v,B]);
5: end for
6: while L_sig ≠ ∅ do
7:    Sea [k, A] ∈ L_sig;
8:    L_sig := L_sig \ {[k, A]};
9:    for all [q, B] ∈ E_[k,A] do
10:      recoverPath(([q,B],[k,A]),[v,B]);
11:   end for
12: end while
```

procedure recoverPath(([q,B],[u,A]),[x,B]);

```
1: P_actual := ([q, B], [u, A]);
2: [z, B] := [q, B];
3: while z ≠ x do
4:    [y, A] := parent([z, B]);
5:    if inL([y, A]) = true then
6:       [r, A] := find([y, A]);
7:       [u, A] := find([v, A]);
8:       union([u, A], [r, A]);
9:       [z, B] := [r, B];
10:   else
11:      [u, A] := find([v, A]);
12:      union([u, A], [y, A]);
13:      inL([y, A]) := true;
14:      P_[y,A] := P_actual;
15:      L_sig := L_sig ∪ {[y, A]};
16:      [z, B] := parent([y, A]);
17:   end if
18: end while
```

hacer m operaciones de este tipo en total, ya que cada arco es considerado exactamente una vez y por tanto, este proceso toma tiempo $O(n + m \log_{(1+m/n)} n)$, donde n y m son el número de vértices y aristas, respectivamente, de la gráfica G a partir de la cual se construye G_M y suponiendo que $m \geq n$.

Para una búsqueda inversa a partir de un vértice $[v, A] \in V(G_M)$ se llevan a cabo a lo más $O(n)$ operaciones find y se revisan a lo más $O(m)$ arcos. Como el número de búsquedas inversas a efectuarse no puede ser mayor a n, entonces el tiempo de ejecución de todas ellas es $O(nm + n^2 \log n)$. Por lo tanto, el tiempo total de ejecución de MDFS en el peor caso es:

$$O(n + m \log_{(1+m/n)} n + nm + n^2 \log n) = O(nm).$$

En la siguiente parte revisaremos los algoritmos para el problema de apareamientos máximos. El algoritmo general requiere de la aplicación de MDFS una vez por cada fase que se tenga.

Capítulo 6

Algoritmos de apareamiento

En este capítulo vamos a revisar los algoritmos completos para resolver el problema de apareamiento máximo en gráficas. Como fue hecho anteriormente, veremos primero el caso especial de las gráficas bipartitas y luego el caso general. En ambos casos, recuperaremos los resultados primordiales de la teoría desarrollada con el fin de favorecer la justificación de los algoritmos.

6.1 Apareamiento en gráficas bipartitas

En esta sección revisaremos dos algoritmos: uno surge inmediatamente de la teoría desarrollada y lo llamaremos básico, mientras que el otro resultará de hacer ciertos cambios al básico con el fin de reducir su complejidad y lo llamaremos algoritmo modificado.

6.1.1 Algoritmo básico de apareamiento en gráficas bipartitas

Consideremos una gráfica bipartita $G = (A, B; E(G))$ y un apareamiento M, el cual inicialmente estará vacío. Cada fase consistirá en encontrar una trayectoria aumentante con respecto al apareamiento actual que se tenga. Esto puede hacerse gracias al Teorema 2.1, el cual garantiza que si existe una trayectoria $\bar{P} : s, v_0, \ldots, v_k, t$ en la gráfica dirigida G_M, entonces es posible obtener de manera constructiva una trayectoria M-aumentante en G, a saber, $P : v_0, \ldots, v_k$. Para ello, aplicaremos una búsqueda en profundidad, partiendo de s y deteniéndose en cuanto t sea alcanzado. Una vez hecho esto, podemos extender el apareamiento M tomando la diferencia simétrica de las aristas de M y de P. Para actualizar el apareamiento actual sólo tenemos que fijarnos en las aristas que aparecen en P: aquellas que están en el apareamiento actual ya no estarán en el nuevo, mientras que aquellas que no están en el actual estarán en el nuevo. De esta forma construiremos un apareamiento que, por el Teorema 1.2, tendrá una arista más que el actual y además, habrá dos vértices apareados más, que

serán justamente los extremos de P.

En caso de que no exista tal trayectoria en G_M, podemos concluir también por el Teorema 2.1 que no existe una trayectoria M-aumentante en G; luego, por el Teorema 1.3, M será máximo y habremos resuelto el problema.

Para la descripción de los algoritmos debemos tomar en cuenta que en cada fase se presenta una reducción del problema de trayectorias aumentantes al problema RP, así que necesitamos solucionar los siguientes tres puntos:

1. La construcción de la gráfica dirigida G_M a partir de la gráfica G y el apareamiento actual M.

2. La búsqueda en profundidad para tratar de encontrar en G_M una trayectoria de s a t.

3. La construcción de la trayectoria M-aumentante y la extensión del apareamiento actual.

Para el algoritmo básico, la forma en la cual se construye la gráfica G_M resulta ser inmediata a partir de su definición. A la búsqueda en profundidad simplemente le agregaremos como condición para detenerse el hecho de etiquetar a t. Bastaría entonces analizar con detalle el tercer punto para justificar todo el algoritmo:
Durante la búsqueda en profundidad asignaremos etiquetas a los vértices para identificar los arcos de árbol, que son los importantes para el proceso. Para cada vértice v en G_M, la etiqueta $parent(v)$ corresponderá al predecesor directo en el árbol que construye DFS. Así, una vez que se detenga la búsqueda cuando se etiquete a t, procederemos a recuperar la trayectoria de $\bar{P} : s, v_0, \ldots, v_k, t$ usando las etiquetas $parent$, empezando con t y con ello automáticamente obtendremos la trayectoria M-aumentante $P : v_0, \ldots, v_k$. Recordamos que $G_M = (A', B'; F(G_M))$, donde $A' = A \cup \{s\}$ y $B' = B \cup \{t\}$. Ahora consideremos un vértice w en la trayectoria de s a t y el vértice $v = parent(w)$. Tenemos tres casos:

- Si $w = t$, entonces $v \in A$ y además es un extremo de la trayectoria M-aumentante correspondiente en G. Como v no es M-apareado, entonces la arista de esa trayectoria cuyo extremo es v estará en $M \oplus P$ y por tanto v será $M \oplus P$-apareado.

- Si $v = s$, entonces $w \in B$ y además es un extremo de la trayectoria M-aumentante correspondiente en G. Como w no es M-apareado, entonces la arista de esa trayectoria cuyo extremo es w estará en $M \oplus P$ y por tanto w será $M \oplus P$-apareado.

- Si $w \neq t$ y $v \neq s$, entonces tanto v como w son vértices de la trayectoria aumentante en G; aquí tenemos dos subcasos:

- Si $v \in A$ entonces $w \in B$ y $vw \in M$, por construcción de G_M. Como vw está en la trayectoria M-aumentante, entonces $vw \notin M \oplus P$, por lo cual podemos eliminar del apareamiento la arista vw.
- Si $v \in B$ entonces $w \in A$ y $vw \notin M$, por construcción de G_M. Como vw está en la trayectoria M-aumentante, entonces $vw \in M \oplus P$, por lo cual podemos agregar al apareamiento la arista vw.

Así, construimos $M \oplus P$, ya que las aristas del apareamiento que no están en la trayectoria quedan intactas y solamente nos fijamos en todas las aristas de la trayectoria M-aumentante para agregar a $M \oplus P$ aquellas que no están en M y desechar las que sí están ahí.

Con esto, y gracias a los resultados obtenidos, podemos concluir que el algoritmo resuelve correctamente el problema del apareamiento máximo en gráficas bipartitas. A continuación tenemos su descripción completa en pseudocódigo.

Listado 9 Procedimiento DFS_Special

procedure DFS_Special;

Entrada: La gráfica dirigida G_M construida a partir de una gráfica G y un apareamiento M.

Salida: Una variable lógica $found$ que indica la existencia de una trayectoria de s a t en G_M.

```
1:  found := false;
2:  parent(s) := s;
3:  push(s);
4:  while K ≠ ∅ y found = false do
5:    pop;
6:    {sea v el vértice eliminado de K.}
7:    if v = t then
8:      marcar a t como visitado;
9:      found := true;
10:   else
11:     if v no está marcado como visitado then
12:       marcar v como visitado;
13:       for all w ∈ N(v) do
14:         if w no está marcado como visitado then
15:           parent(w) := v;
16:         end if
17:         push(w);
18:       end for
19:     end if
20:   end if
21: end while
```

Listado 10 Procedimiento convertInstance

procedure convertInstance;

Entrada: Una gráfica G y un apareamiento M de G.
Salida: La gráfica dirigida G_M.

```
1: V(G_M) := V(G) ∪ {s, t};
2: F(G_M) := ∅;
3: for all ab ∈ E(G) do
4:    {a ∈ A y b ∈ B.}
5:    if ab ∈ M then
6:       F(G_M) := F(G_M) ∪ {(a, b)};
7:    else
8:       F(G_M) := F(G_M) ∪ {(b, a)};
9:    end if
10: end for
11: for all b ∈ B do
12:    if b no es M-apareado then
13:       F(G_M) := F(G_M) ∪ {(s, b)};
14:    end if
15: end for
16: for all a ∈ A do
17:    if a no es M-apareado then
18:       F(G_M) := F(G_M) ∪ {(a, t)};
19:    end if
20: end for
```

Listado 11 Procedimiento augmentMatching

procedure augmentMatching;

Entrada: La gráfica dirigida G_M.
Salida: $M := M \oplus P$.

```
1: w := t;
2: while w ≠ s do
3:    v := parent(w);
4:    if w = t then
5:       marcar v como M-apareado;
6:    else if v = s then
7:       marcar w como M-apareado;
8:    else
9:       if v ∈ A then
10:         M := M \ {vw};
11:      else
12:         M := M ∪ {vw};
13:      end if
14:   end if
15:   w := v;
16: end while
```

En el Listado 9 se observa la descripción de la búsqueda en profundidad, la cual es aplicada a una gráfica G_M y regresa una variable lógica que indica si existe una trayectoria de s a t en dicha gráfica. Se utiliza una pila K como en cualquier otra especificación de DFS.

En el Listado 10 se muestra la descripción del procedimiento que construye la gráfica dirigida G_M a partir de la gráfica G y un apareamiento M.

En el Listado 11 se indica el procedimiento que recupera la trayectoria de s a t en G_M y al mismo tiempo construye el nuevo apareamiento, eliminando aristas de la trayectoria aumentante correspondiente en G si están en el apareamiento actual y añadiéndolas a éste si no están.

En el Listado 12 se muestra el algoritmo que resuelve el problema de apareamiento máximo en gráficas bipartitas. La precondición es que la gráfica de entrada sea bipartita y se haya identificado previamente una bipartición. La postcondición es que la salida será un apareamiento máximo M_{max}. Se manejan dos variables de tipo lógico: $isMaximum$ indica si el apareamiento actual es máximo o no y $found$ indica si existe una trayectoria aumentante en G.

Listado 12 Algoritmo básico de apareamiento máximo en gráficas bipartitas

Entrada: Una gráfica bipartita $G = (A, B; E(G))$.
Salida: Un apareamiento máximo M_{max}.

```
1:  M := ∅;
2:  isMaximum := false;
3:  found := false;
4:  while isMaximum = false do
5:     G_M := convertInstance(G, M);
6:     found := DFS_Special(G_M);
7:     if found = true then
8:        M := augmentMatching(G_M);
9:     else
10:       isMaximum := true;
11:    end if
12: end while
13: M_max := M;
```

Ahora sólo nos resta analizar la complejidad del algoritmo del Listado 12. Su tiempo de ejecución depende del número de iteraciones del ciclo principal y de los tiempos de ejecución de los procedimientos DFS_Special, convertInstance y augmentMatching. Para ver el número de iteraciones del ciclo principal nos basaremos en el siguiente resultado.

Lema 6.1. Sea $G = (A, B; E(G))$ una gráfica bipartita y M un apareamiento de G. Si $|M| = min(|A|, |B|)$ entonces M es máximo.

Demostración. Supongamos, sin pérdida de generalidad, que $min(|A|, |B|) = |A|$, es decir, $|M| = |A|$. Como G es bipartita, cada arista en M une un extremo en A con otro en B. Puesto que M es apareamiento, cada arista de M tiene un extremo en A distinto de las demás, por lo tanto, todo vértice en A es M-apareado.

Ahora supongamos que M no es máximo. Entonces, por el Teorema 1.3, existe una trayectoria M-aumentante en G. Como los extremos de esa trayectoria deben ser no M-apareados, ambos deben estar en B, pero como G es bipartita, la longitud de esa trayectoria es par y entonces por el Lema 1.1, esa trayectoria no sería M-aumentante.
$\therefore M$ es máximo. □

Sea $n = |V(G)|$ y $m = |E(G)|$. Si $G = (A, B; E(G))$ es una gráfica bipartita, entonces $n = |A| + |B|$. Así, tenemos que $min(|A|, |B|) \leq \frac{n}{2}$, de otro modo tendríamos una contradicción:

$$min(|A|, |B|) + max(|A|, |B|) > \tfrac{n}{2} + \tfrac{n}{2} = n.$$

Por el Lema 6.1, el número de iteraciones del ciclo principal para el algoritmo del Listado 12 es a lo más $(\frac{n}{2} + 1)$, puesto que tendría que ejecutarse una iteración más para que el valor de *found* sea $false$ y por ende, el valor de *isMaximum* sea *true*.

Veamos el tiempo de ejecución del procedimiento convertInstance del Listado 10. La construcción de los vértices toma $(n + 2)$ operaciones, así que el tiempo de ejecución es $O(n)$. Cada arista tendrá una etiqueta relativa a la pertenencia o no al apareamiento actual, por lo que el segundo ciclo toma $O(m)$ operaciones. Cada vértice tendrá también una etiqueta indicando si es apareado o simple con respecto al apareamiento actual, así que los tiempos de ejecución del tercer y cuartos ciclos son $O(|B|)$ y $O(|A|)$, respectivamente.
$\therefore$ La construcción de la gráfica G_M para un apareamiento dado M con el procedimiento convertInstance toma tiempo $O(n + m)$.

Veamos el tiempo de ejecución del procedimiento DFS_Special del Listado 9. Para ello calculemos el número de arcos de la gráfica G_M, el cual depende del número de iteración actual del algoritmo del Listado 12. En cada fase o iteración de este algoritmo se marcan dos vértices como M-apareados, que son los extremos de la trayectoria M-aumentante que se haya encontrado, por lo que el número de arcos de G_M se reduce en dos unidades por cada fase.
Así, en la iteración i-ésima del ciclo principal, la gráfica G_M consistirá de $(n+2)$ vértices y $\big(m+n-2(i-1)\big)$ arcos y por consecuencia, como se trata de una búsqueda en profundidad, el tiempo de ejecución de este procedimiento es $O(n + [m + n - 2(i - 1)]) = O(m + 2(n - i + 1))$.

Veamos ahora el tiempo de ejecución del procedimiento augmentMatching del Listado 11. El número de iteraciones de este procedimiento depende de la

longitud de la trayectoria aumentante que se haya encontrado. El peor caso que puede ocurrir es que siempre se tome la trayectoria aumentante de longitud más larga posible, por el hecho de que todas las aristas del apareamiento actual se encuentren en ella. A su vez, la longitud de la trayectoria aumentante depende del tamaño del apareamiento, que en este caso coincide con el número de iteración actual del algoritmo del Listado 12. Así, en la iteración i-ésima del ciclo principal, la longitud de la trayectoria aumentante más larga posible es $(2i-1)$ y, por tanto, el número de iteraciones del procedimiento es $(2i+1)$, puesto que la trayectoria correspondiente en G_M tendría $(2i+1)$ vértices.

Analicemos el tiempo de ejecución del cuerpo del procedimiento: el hecho de mantener etiquetas para vértices y aristas como se indicaba anteriormente nos lleva a hacer actualizaciones no muy eficientes en las aristas, ya que no contaremos con una estructura de datos para almacenar las aristas del apareamiento actual. De esta forma, para cada arista del apareamiento dentro de la trayectoria aumentante encontrada tendríamos que usar una búsqueda lineal para etiquetarla como parte del apareamiento o no, según sea el caso. En las dos iteraciones restantes se tendría tiempo constante.
$\therefore$ El tiempo de ejecución del procedimiento es $O(m(2i-1))$.

Con todo esto, podemos finalmente calcular el tiempo de ejecución del Algoritmo básico de apareamiento para gráficas bipartitas. Su número de iteraciones, como se mencionaba, es a lo más $(\frac{n}{2}+1)$ y en el tiempo de ejecución del cuerpo del ciclo principal influyen los tres tiempos calculados anteriormente, por lo que el tiempo de ejecución del algoritmo en cuestión es:

$$[\frac{n}{2}+1][O(n+m)] + \sum_{i=1}^{\frac{n}{2}+1} O(m+2(n-i+1)) + \sum_{i=1}^{\frac{n}{2}} O(m(2i-1)) = O(mn^2).$$

Una gráfica $G = (A, B; E(G))$ bipartita completa con $|A| = |B| = \frac{n}{2}$ tiene $\frac{n^2}{4}$ aristas, por lo que el peor de los casos el Algoritmo básico de apareamientos para gráficas bipartitas tiene complejidad $O(n^4)$. Una de las principales causas de ello es la forma en la cual se obtiene la información sobre el apareamiento actual: manteniendo etiquetas en aristas para ver si están o no en el apareamiento, y con ello el proceso de incremento en tamaño de un apareamiento tarda más.

Podríamos aprovechar mejor la definición de un apareamiento de la siguiente forma: usaremos etiquetas para cada vértice que indiquen cuál vértice es su pareja, definiendo de este modo el apareamiento actual implícitamente y podrá recuperarse al final simplemente con la información que proporcionan esas etiquetas. Además, haremos algunos cambios al algoritmo básico con respecto a la construcción de las gráficas dirigidas G_M.

6.1.2 Algoritmo modificado de apareamiento en gráficas bipartitas

El algoritmo que revisaremos a continuación procede de manera similar al algoritmo básico con las siguientes diferencias:

- Para cada vértice v en la gráfica G usaremos una etiqueta $mate(v)$ para denotar a aquel vértice w tal que $vw \in M$.
- Solamente efectuaremos una construcción de la gráfica G_M, cuando M es vacío, lo cual es sencillo pues todos los arcos correspondientes a aristas de la gráfica G tendrán extremo inicial en B y extremo final en A. Además, como todo vértice es no apareado con respecto al apareamiento vacío, se tiene que $\forall a \in A : (a,t) \in F(G_M)$ y $\forall b \in B : (s,b) \in F(G_M)$.
- La gráfica G_M será modificada en cada fase únicamente en la trayectoria de s a t que se haya encontrado: si $P : s, v_0, \ldots, v_k, t$ es la trayectoria en cuestión, entonces lo único que tendremos que hacer para obtener la gráfica que usaremos para la siguiente fase es borrar los arcos (s, v_0) y (v_k, t), e invertir el sentido de todos los arcos restantes que están en la trayectoria. El apareamiento también se modificará a partir de la trayectoria P como en el algoritmo básico con la actualización de las etiquetas *mate*.
- Se realizará al final de cada fase un borrado de etiquetas *parent* y se marcarán como no visitados a todos los vértices de G_M.
- La recuperación de las aristas del apareamiento se hará con ayuda de las etiquetas *mate*. Una arista vw estará en el apareamiento final si y sólo si $mate(v) = w$ y $mate(w) = v$.

Para justificar este algoritmo sólo nos faltaría fijarnos en la actualización de las gráficas G_M y de las etiquetas *mate* durante el proceso para incrementar el tamaño del apareamiento actual. Establecemos primero el siguiente resultado.

Lema 6.2. Sea $G = (A, B; E(G))$ una gráfica bipartita y M un apareamiento de G. Si $P : v_0, v_1, \ldots, v_k$ es una trayectoria M-aumentante en G, con $v_0 \in B$ y $v_k \in A$, entonces:

1. $V(G_{M \oplus P}) = V(G_M)$.
2. $F(G_{M \oplus P}) = F(G_M) \setminus \big(\{(s, v_0), (v_k, t)\} \cup \{(v_i, v_{i+1}) \mid v_i v_{i+1} \in E(P)\}\big)$ $\cup \{(v_{i+1}, v_i) \mid v_i v_{i+1} \in E(P)\}$ donde $i = 0, \ldots, k-1$.

Demostración. La igualdad en 1 se sigue directamente ya que:
$V(G_{M \oplus P}) = V(G) \cup \{s, t\} = V(G_M)$.

Demostremos ahora la igualdad en 2. Por definición, tenemos que:

$$\begin{aligned} F(G_{M\oplus P}) = &\{(a,b) : ab \in M \oplus P\} \cup \{(b,a) : ab \notin M \oplus P\} \\ &\cup \{(s,b) : b \in B \text{ y } b \text{ no es } M \oplus P-\text{apareado}\} \\ &\cup \{(a,t) : a \in A \text{ y } a \text{ no es } M \oplus P-\text{apareado}\}. \end{aligned}$$

Sea $(x,y) \in F(G_{M\oplus P})$. Afirmamos que (x,y) está en el conjunto del lado derecho de la igualdad en 2. Tenemos cuatro casos:

1. Si $(x,y) \in \{(a,b) : ab \in M \oplus P\}$ entonces $xy \in M \oplus P$ con $x \in A$, $y \in B$; es decir, $xy \in M \setminus E(P) \cup E(P) \setminus M$. Tenemos dos subcasos:

   (a) Si $xy \in M \setminus E(P)$, en particular tenemos que $xy \notin E(P)$
   $\therefore (x,y) \notin \{(s,v_0),(v_k,t)\} \cup \{(v_i,v_{i+1}) \mid v_iv_{i+1} \in E(P)\}$.
   Como $xy \in M$, tenemos que $(x,y) \in F(G_M)$.

   (b) Si $xy \in E(P) \setminus M$, en particular tenemos que $xy \notin M$, por lo que $(x,y) \notin F(G_M)$.
   $\therefore (x,y) \notin \{(s,v_0),(v_k,t)\} \cup \{(v_i,v_{i+1}) \mid v_iv_{i+1} \in E(P)\}$
   así que $(x,y) \in \{(v_{i+1},v_i) \mid v_iv_{i+1} \in E(P)\}$, ya que $xy \in E(P)$.

   Por lo tanto se cumple la afirmación en este caso.

2. Si $(x,y) \in \{(b,a) : ab \notin M \oplus P\}$ entonces $x \in B$, $y \in A$ y además $yx \notin M \oplus P$. Tenemos dos subcasos:

   (a) Si $yx \notin M \cup E(P)$, en particular tenemos que $yx \notin E(P)$, por tanto:
   $(x,y) \notin \{(s,v_0),(v_k,t)\} \cup \{(v_i,v_{i+1}) \mid v_iv_{i+1} \in E(P)\}$;
   y como $yx \notin M$, entonces $(x,y) \in F(G_M)$.

   (b) Si $yx \in M \cap E(P)$, en particular tenemos que $yx \in M$, por tanto:
   $(y,x) \in F(G_M)$, así que $(x,y) \notin F(G_M)$.
   $\therefore (x,y) \notin \{(s,v_0),(v_k,t)\} \cup \{(v_i,v_{i+1}) \mid v_iv_{i+1} \in E(P)\}$.
   $\therefore (x,y) \in \{(v_{i+1},v_i) \mid v_iv_{i+1} \in E(P)\}$, ya que $yx \in E(P)$.

   Por lo tanto se cumple la afirmación en este caso.

3. Si $(x,y) \in \{(s,b) : b \in B \text{ y } b \text{ no es } M \oplus P-\text{apareado}\}$ entonces $y \in B$, y no es $M \oplus P$-apareado y además $x = s$.
   Demostremos que y no es M-apareado. Supongamos que sí lo es. Entonces existe una arista de M que tiene como extremo y, por lo que y no puede ser extremo de ninguna trayectoria M-aumentante. Si y estuviera en P, habría dos aristas de P que lo comparten como extremo: una está en M y la otra no. Así que esta arista sí aparece en $M \oplus P$ y la otra no, por tanto y sería $M \oplus P$-apareado. Si y no estuviera en P, la arista de M que lo tiene como extremo también estaría en $M \oplus P$ y nuevamente tendríamos una contradicción.
   $\therefore (s,y) \in F(G_M)$.
   De cualquier forma, y no puede ser extremo de P, puesto que entonces sería $M \oplus P$-apareado, así que $y \neq v_0$.
   $\therefore (x,y) \in F(G_M) \setminus \big(\{(s,v_0),(v_k,t)\} \cup \{(v_i,v_{i+1}) \mid v_iv_{i+1} \in E(P)\}\big)$.
   Por lo tanto se cumple la afirmación en este caso.

4. Si $(x,y) \in \{(a,t) : a \in A \text{ y } a \text{ no es } M \oplus P-\text{apareado}\}$ entonces $x \in A$, x no es $M \oplus P$-apareado y además $y = t$.
   Demostremos que x no es M-apareado. Supongamos que sí lo es. Entonces existe una arista de M que tiene como extremo x, por lo que x no puede ser extremo de ninguna trayectoria M-aumentante. Si x estuviera en P, habría dos aristas de P que lo comparten como extremo: una está en M y la otra no. Así que esta arista sí aparece en $M \oplus P$ y la otra no, por tanto x sería $M \oplus P$-apareado. Si x no estuviera en P, la arista de M que lo tiene como extremo también estaría en $M \oplus P$ y nuevamente tendríamos una contradicción.
   $\therefore (x,t) \in F(G_M)$.
   De cualquier forma, x no puede ser extremo de P, puesto que entonces sería $M \oplus P$-apareado, así que $x \neq v_k$.
   $\therefore (x,y) \in F(G_M) \setminus \big(\{(s,v_0),(v_k,t)\} \cup \{(v_i,v_{i+1}) \mid v_iv_{i+1} \in E(P)\}\big)$.
   Por lo tanto se cumple la afirmación en este caso.

$\therefore$ La afirmación se satisface en cualquiera de los cuatro casos.

Sea $(x,y) \in F(G_M) \setminus \big(\{(s,v_0),(v_k,t)\} \cup \{(v_i,v_{i+1}) \mid v_iv_{i+1} \in E(P)\}\big)$
$\quad\quad \cup \{(v_{i+1},v_i) \mid v_iv_{i+1} \in E(P)\}$.
Afirmamos que $(x,y) \in F(G_{M\oplus P})$. Tenemos dos casos:

1. Si $(x,y) \in F(G_M)$ y (x,y) no está en $\bar{P} : s, v_0, \ldots, v_k, t$. Tenemos cuatro subcasos.

   (a) Si $(x,y) \in \{(a,b) \mid ab \in M\}$, como (x,y) no está en $\bar{P}$, entonces xy no está en P.
   $\therefore xy \in M \oplus P$.
   $\therefore (x,y) \in \{(a,b) \mid ab \in M \oplus P\}$.
   $\therefore (x,y) \in F(G_{M\oplus P})$.

   (b) Si $(x,y) \in \{(b,a) \mid ab \notin M\}$, como (x,y) no está en $\bar{P}$, entonces yx no está en P.
   $\therefore yx \notin M \cup P$.
   $\therefore yx \notin M \oplus P$.
   $\therefore (x,y) \in \{(b,a) \mid ab \notin M \oplus P\}$.
   $\therefore (x,y) \in F(G_{M\oplus P})$.

   (c) Si $(x,y) \in \{(s,b) \mid b \in B \text{ y } b \text{ no es } M-\text{apareado}\}$, como (x,y) no está en $\bar{P}$ entonces y no está en P.
   $\therefore y$ no es $M \oplus P$-apareado.
   $\therefore (x,y) \in \{(s,b) \mid b \in B \text{ y } b \text{ no es } M \oplus P-\text{apareado}\}$.
   $\therefore (x,y) \in F(G_{M\oplus P})$.

   (d) Si $(x,y) \in \{(a,t) \mid a \in A \text{ y } a \text{ no es } M-\text{apareado}\}$, como (x,y) no está en $\bar{P}$ entonces x no está en P.
   $\therefore x$ no es $M \oplus P$-apareado.
   $\therefore (x,y) \in \{(a,t) \mid a \in A \text{ y } a \text{ no es } M \oplus P-\text{apareado}\}$.
   $\therefore (x,y) \in F(G_{M\oplus P})$.

2. Si $(x, y) \in \{(v_{i+1}, v_i) \mid v_i v_{i+1} \in E(P)\}$, entonces por construcción, sabemos que los arcos (v_i, v_{i+1}), donde $i = 0, \ldots, k-1$, están en $F(G_M)$. Tenemos entonces dos subcasos:

   (a) Si $v_i \in A$, entonces $v_{i+1} \in B$ y $v_i v_{i+1} \in M$.
   $\therefore v_i v_{i+1} \notin M \oplus P$.
   $\therefore (v_{i+1}, v_i) \in F(G_{M \oplus P})$.

   (b) Si $v_i \in B$, entonces $v_{i+1} \in A$ y $v_i v_{i+1} \notin M$.
   $\therefore v_i v_{i+1} \in M \oplus P$.
   $\therefore (v_{i+1}, v_i) \in F(G_{M \oplus P})$.

$\therefore$ La afirmación se satisface en cualquiera de los dos casos.
Finalmente podemos concluir que la igualdad en 2 se satisface. □

Para el algoritmo modificado, como se mencionaba antes, la única vez en la que se construye explícitamente la gráfica G_M es cuando $M = \varnothing$ y emplearemos la búsqueda en profundidad especificada en el Listado 9 para tratar de encontrar una trayectoria de s a t. Ahora analizaremos con detalle la forma en la cual se modifica el apareamiento actual y la gráfica G_M:
Usaremos las etiquetas *parent* para cada vértice en G_M para recuperar tal trayectoria. Consideremos un vértice w en la trayectoria de s a t y el vértice $v = parent(w)$. Tenemos tres casos:

- Si $w = t$, entonces v será $M \oplus P$-apareado y por tanto el arco (v, w) no estará en la gráfica $G_{M \oplus P}$, así que por el Lema 6.2, podemos eliminar el arco (v, w) de G_M.

- Si $v = s$, entonces w será $M \oplus P$-apareado y por tanto el arco (v, w) no estará en la gráfica $G_{M \oplus P}$, así que por el Lema 6.2, podemos eliminar el arco (v, w) de G_M.

- Si $w \neq t$ y $v \neq s$, entonces tanto v como w son vértices de la trayectoria aumentante correspondiente en G y tenemos dos subcasos:

  – Si $v \in A$ entonces la arista vw no estará en el nuevo apareamiento, pero ambos extremos serán $M \oplus P$-apareados, así que en esta ocasión no actualizamos las etiquetas *mate*.

  – Si $v \in B$ entonces la arista vw estará en el nuevo apareamiento, por lo cual actualizamos las etiquetas de esta forma: $mate(v) = w$ y $mate(w) = v$.

  En ambos subcasos, el arco (v, w) no aparecerá en la gráfica $G_{M \oplus P}$, puesto que cambia la situación de vw con respecto a $M \oplus P$. Así, por el Lema 6.2, podemos eliminar el arco (v, w) de G_M y agregar el arco (w, v).

Así, construimos implícitamente el apareamiento $M \oplus P$, puesto que las etiquetas *mate* de los vértices que no están en la trayectoria quedan intactas y solamente cambian las etiquetas de los vértices que están en la trayectoria M-aumentante, respetándose la definición de apareamiento.

Asimismo, se modifica la gráfica G_M para obtener $G_{M \oplus P}$, lo cual se puede garantizar gracias al Lema 6.2, pues éste afirma que sólo hay que fijarnos en la trayectoria de s a t que encontremos en G_M. Por tanto, podemos concluir que el algoritmo modificado resuelve correctamente el problema del apareamiento máximo en gráficas bipartitas.

A continuación se muestra la descripción en pseudocódigo del algoritmo. Seguimos empleando el procedimiento del Listado 9 para buscar una trayectoria de s a t en G_M. En el Listado 13 se muestra la descripción del algoritmo que construye la gráfica dirigida G_M donde $M = \varnothing$.

Listado 13 Procedimiento convertFirstInstance

procedure convertFirstInstance;

Entrada: Una gráfica G.
Salida: La gráfica dirigida G_M, donde $M = \varnothing$.

```
1: V(G_M) := V(G) ∪ {s, t};
2: F(G_M) := ∅;
3: for all ab ∈ E(G) do
4:    {a ∈ A y b ∈ B.}
5:    F(G_M) := F(G_M) ∪ {(b, a)};
6: end for
7: for all b ∈ B do
8:    F(G_M) := F(G_M) ∪ {(s, b)};
9: end for
10: for all a ∈ A do
11:    F(G_M) := F(G_M) ∪ {(a, t)};
12: end for
```

En el Listado 14 se indica el algoritmo que recupera la trayectoria de s a t en G_M, en caso de existir, y al mismo tiempo construye el nuevo apareamiento con la actualización de las etiquetas *mate* para los vértices en G.

En el Listado 15 se observa el procedimiento que borra las etiquetas de los vértices en la gráfica G_M, para el apareamiento actual M.

En el Listado 16 se observa el procedimiento que recupera el apareamiento máximo a partir de las etiquetas *mate*.

En el Listado 17 se muestra el algoritmo modificado de apareamiento máximo en gráficas bipartitas. La precondición es que la gráfica de entrada sea bipartita y se haya identificado previamente una bipartición. La postcondición es que la salida será un apareamiento máximo M_{max}. Se usan las variables $isMaximum$ y $found$ al igual que en el algoritmo básico, así como una variable entera j que indicará número de elementos en el apareamiento actual.

Listado 14 Procedimiento augmentMatching_update

procedure augmentMatching_update;

Entrada: La gráfica dirigida G_M y la gráfica G.
Salida: $M := M \oplus P$ y $G_M := G_{M \oplus P}$.

```
1:  w := t;
2:  while w ≠ s do
3:     v := parent(w);
4:     if w = t then
5:        marcar v como M-apareado;
6:        F(G_M) := F(G_M) \ {(v, w)};
7:     else if v = s then
8:        marcar w como M-apareado;
9:        F(G_M) := F(G_M) \ {(v, w)};
10:    else
11:       F(G_M) := (F(G_M) \ {(v, w)}) ∪ {(w, v)};
12:       if v ∈ B then
13:          mate(v) := w;
14:          mate(w) := v;
15:       end if
16:    end if
17:    w := v;
18: end while
```

Listado 15 Procedimiento restore

procedure restore;

Entrada: La gráfica dirigida G_M.
Salida: $\forall v \in V(G_M) : parent(v) := null$ y v es marcado como no visitado.

```
1: for all v ∈ V(G_M) do
2:    parent(v) := null;
3:    marcar a v como no visitado;
4: end for
```

Listado 16 Procedimiento recoverMatching

procedure recoverMatching;

Entrada: La gráfica G.
Salida: M es un apareamiento de G.

```
1: for all vw ∈ E(G) do
2:    if mate(v)=w then
3:       M := M ∪ {vw};
4:    end if
5: end for
```

Ahora sólo nos resta analizar la complejidad del algoritmo del Listado 17. Su tiempo de ejecución depende de los tiempos de ejecución de los procedimientos convertFirstInstance, recoverMatching, así como del tiempo de ejecución del ciclo principal, el cual depende de su número de iteraciones y los tiempos de ejecución de los procedimientos DFS_Special, restore y augmentMatching_update.

Listado 17 Algoritmo modificado de apareamiento máximo en gráficas bipartitas

Entrada: Una gráfica bipartita $G = (A, B; E(G))$.
Salida: Un apareamiento máximo M_{max}.

```
1:  M := ∅;
2:  isMaximum := false;
3:  found := false;
4:  i := 0;
5:  min := min(|A|, |B|);
6:  G_M := convertFirstInstance(G);
7:  while isMaximum = false do
8:     found := DFS_Special(G_M);
9:     if found = true then
10:       M := augmentMatching_update(G_M, G);
11:       j := j + 1;
12:       if j = min then
13:          isMaximum := true;
14:       end if
15:    else
16:       isMaximum := true;
17:    end if
18:    restore;
19: end while
20: M_max := recoverMatching(G);
```

El procedimiento convertFirstInstance del Listado 13 toma tiempo $O(n+m)$, donde $n = |V(G)|$ y $m = |E(G)|$. El procedimiento recoverMatching del Listado 16 toma tiempo $O(m)$.

El número de iteraciones del algoritmo modificado, según el Lema 6.1, es a lo más $\frac{n}{2}$. A diferencia del algoritmo básico, nos ahorramos una iteración con el hecho de verificar la cardinalidad del apareamiento actual.

Veamos ahora el tiempo de ejecución del cuerpo del ciclo principal del algoritmo modificado. Como sabemos, el tiempo de ejecución del procedimiento DFS_Special del Listado 9 es $O(m + 2(n - i + 1))$, donde i es el número de la iteración actual. El procedimiento restore en el Listado 15 toma tiempo $O(n)$. Solamente nos quedaría por analizar el tiempo de ejecución del procedimiento augmentMatching_update del Listado 14:

En la i-ésima iteración del ciclo principal, el número de iteraciones de este procedimiento es a lo más $(2i+1)$, como habíamos visto en el algoritmo básico. Tanto en la primera como en la última iteración sólo se elimina un arco en G_M, mientras que en las $(2i-1)$ restantes se elimina uno y se inserta además otro.

El número de vértices en una trayectoria aumentante es par, y además, para el caso de las gráficas bipartitas, hay tantos vértices en A como en B en ella. Por tanto, en G_M, una trayectoria P de s a t tendrá $2i$ vértices a lo más.

Tanto la inserción como la remoción de un arco en cualquier gráfica dirigida tardan a lo más un tiempo equivalente al exgrado de su extremo inicial.

Consideramos los exgrados de los vértices en P. Con esto, el tiempo de ejecución del ciclo de este procedimiento es:

$$\begin{aligned}
&\delta_P^+(s) + 2\sum_{k=1}^{2i} \delta_P^+(v_k) \;\; = \\
&\quad = \;\; \delta_P^+(s) + 2\sum_{k=1}^{i} \delta_P^+(a_k) + 2\sum_{k=1}^{i} \delta_P^+(b_k) \;\; \text{para } a_k \in A, b_k \in B \\
&\quad \leq \;\; [|B| - (i-1)] + 2i + 2|A|i \\
&\quad \leq \;\; [n - 1 - i + 1] + 2i + 2i(n-1) \\
&\quad = \;\; n - i + 2ni.
\end{aligned}$$

Por lo tanto, el tiempo de ejecución total del algoritmo modificado de apareamiento para gráficas bipartitas es:

$$O(n+m)+\sum_{i=1}^{\frac{n}{2}} O(m+2(n-i+1))+\sum_{i=1}^{\frac{n}{2}} O(n-i+2ni)+\sum_{i=1}^{\frac{n}{2}} O(n)+O(m) = O(n^3).$$

6.1.3 Ejemplo

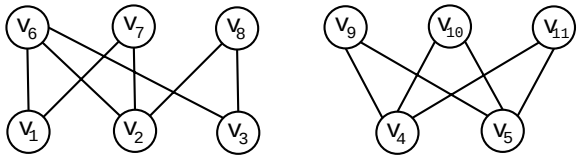

Figura 6.1: Gráfica bipartita para la aplicación del algoritmo modificado de apareamientos

Ahora veamos un ejemplo del algoritmo modificado. Consideremos la gráfica ilustrada en la Figura 6.1, cuya bipartición está dada por $A = \{v_1, v_2, v_3, v_4, v_5\}$

y $B = \{v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$. La aplicación de DFS en la gráfica dirigida G_M con $M = \varnothing$ se observa en la Figura 6.2.

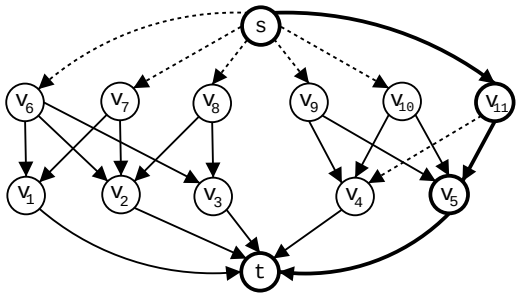

Figura 6.2: Primera fase del algoritmo modificado

Los arcos y vértices remarcados son parte del árbol construido, mientras que los arcos punteados son los que están pendientes por examinar. La trayectoria encontrada es $[s, v_{11}, v_5, t]$ y ponemos entonces:

$$mate(v_{11}) := v_5 \text{ y } mate(v_5) := v_{11}.$$

El nuevo apareamiento se observa en la Figura 6.3.

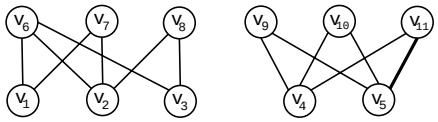

Figura 6.3: Apareamiento al inicio de la segunda fase

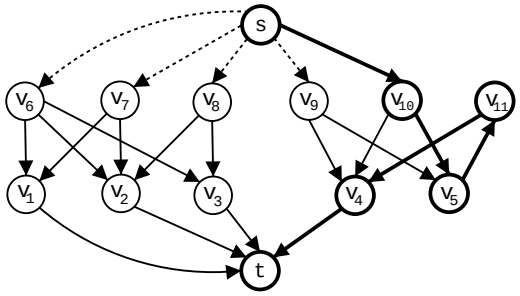

Figura 6.4: Segunda fase del algoritmo modificado

Ilustramos la aplicación de DFS en la Figura 6.4. Notamos que los únicos cambios se dieron en la trayectoria anterior. En este caso la trayectoria encontrada es $[s, v_{10}, v_5, v_{11}, v_4, t]$ y ponemos entonces

$mate(v_{11}) := v_4$, $mate(v_4) := v_{11}$, $mate(v_{10}) := v_5$ y $mate(v_5) := v_{10}$.
El nuevo apareamiento se muestra en la Figura 6.5.

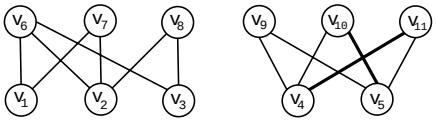

Figura 6.5: Apareamiento al inicio de la tercer fase

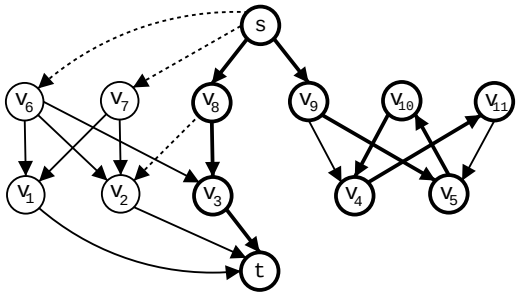

Figura 6.6: Tercera fase del algoritmo modificado

Tenemos la aplicación de DFS en la Figura 6.6. Aquí la trayectoria encontrada es $[s, v_8, v_3, t]$ y ponemos entonces $mate(v_8) := v_3$ y $mate(v_3) := v_8$. El nuevo apareamiento se observa en la Figura 6.7.

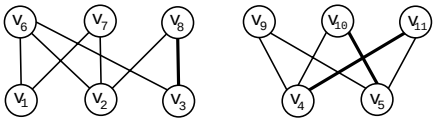

Figura 6.7: Apareamiento al inicio de la cuarta fase

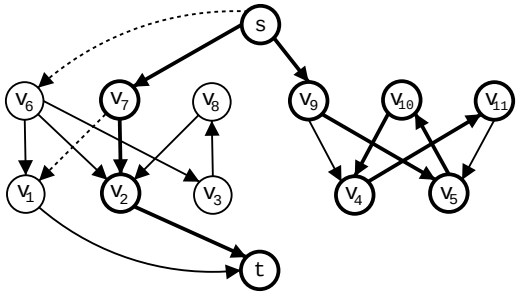

Figura 6.8: Cuarta fase del algoritmo modificado

La aplicación de DFS se observa en la Figura 6.8. Aquí la trayectoria encontrada es $[s, v_7, v_2, t]$ y ponemos entonces $mate(v_7) := v_2$ y $mate(v_2) := v_7$. El nuevo apareamiento se ilustra en la Figura 6.9.

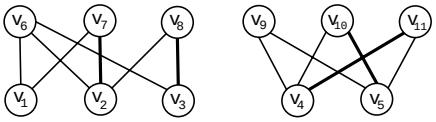

Figura 6.9: Apareamiento al inicio de la quinta fase

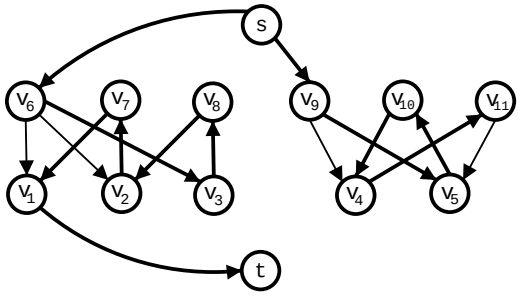

Figura 6.10: Quinta fase del algoritmo modificado

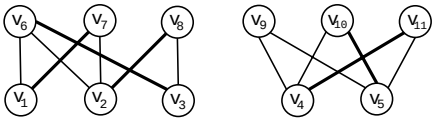

Figura 6.11: Apareamiento máximo construido

La última aplicación de DFS se observa en la Figura 6.10. La trayectoria entontrada es $[s, v_6, v_3, v_8, v_2, v_7, v_1, t]$ y ponemos entonces:

$$mate(v_7) := v_1,\ mate(v_1) := v_7,\ mate(v_8) := v_2,\ mate(v_2) := v_8,$$
$$mate(v_6) := v_3 \text{ y } mate(v_3) := v_6.$$

En este momento la cardinalidad del apareamiento es igual a $|A|$, por tanto el apareamiento construido por el algoritmo modificado es máximo y se observa en la Figura 6.11.

Para terminar con esta sección, cabe señalar que es posible verificar si una gráfica conexa es bipartita con una modificación del algoritmo de búsqueda en amplitud BFS (*Breadth First Search*), además de poder dar una bipartición de ella, con el fin de cubrir las precondiciones de nuestros algoritmos de apareamiento, sin afectar la complejidad total.

6.2 Apareamiento en gráficas generales

Consideremos una gráfica G y un apareamiento M, el cual inicialmente será vacío. Cada fase del algoritmo consistirá en encontrar una trayectoria aumentante con respecto al apareamiento actual. El Teorema 2.2 garantiza que si existe una trayectoria original $\bar{P} : s, [v_0, B], [v_1, A], \ldots, [v_{k-1}, B], [v_k, A], t$ en la gráfica dirigida G_M, entonces es posible obtener una trayectoria M-aumentante en G, a saber, $P : v_0, v_1, \ldots, v_{k-1}, v_k$. Para ello, aplicaremos el algoritmo MDFS partiendo de s hasta que sea alcanzado t. Una vez hecho esto, extendemos el apareamiento M tomando la diferencia simétrica de las aristas de M y P. Así como en el caso de las gráficas bipartitas, solamente nos fijaremos en las aristas de P para actualizar el apareamiento y éste tenga una arista más que el anterior, además de que v_0 y v_k serán los nuevos vértices apareados. La actualización del apareamiento se hace con la etiqueta $mate(v)$, para cada $v \in V(G_M)$, que indica el vértice w tal que $vw \in M$. Usamos también las etiquetas *matched* para indicar que v_0 y v_k estarán apareados hasta el final. En caso de que G_M no tenga una trayectoria original de s a t, por el Teorema 2.2, G no tiene trayectorias M-aumentantes y por tanto, M será máximo.

Para la descripción de este algoritmo nuevamente debemos tomar en cuenta la reducción del problema de encontrar trayectorias aumentantes al problema RP que se lleva a cabo en cada fase, por lo cual necesitamos considerar los siguientes tres puntos:

1. La construcción de la gráfica dirigida G_M a partir de la gráfica G y el apareamiento actual M.
2. La aplicación del algoritmo MDFS para tratar de hallar en G_M una trayectoria original de s a t.
3. La construcción de la trayectoria M-aumentante en G y la actualización del apareamiento.

La construcción de G_M resulta ser inmediata a partir de su definición. Utilizamos la versión del algoritmo MDFS detallada a partir del Listado 6. Ahora vamos a revisar la solución al tercer punto:
Supongamos que la aplicación de MDFS consigue etiquetar a t como visitado. Entonces aplicamos el procedimiento reconstructPath, indicado en el Listado 5, para obtener la trayectoria original $P : s, [v_0, B], [v_1, A], \ldots, [v_{k-1}, B], [v_k, A], t$ con la cual vamos a construir la trayectoria aumentante $P : v_0, v_1, \ldots, v_{k-1}, v_k$ y actualizar M. Consideremos un vértice $[w, X]$ en $\bar{P}$ y el vértice $[v, Y]$ tal que $[v, Y] = parent([w, X])$. Tenemos tres casos:

- Si $w = t$, entonces $Y = A$ y además v es un extremo de P, por tanto v no es M-apareado pero sí es $M \oplus P$-apareado.
- Si $v = s$, entonces $X = B$ y además w es un extremo de P, por tanto w no es M-apareado pero sí es $M \oplus P$-apareado.

- Si $w \neq t$ y $v \neq s$, entonces tanto v como w son vértices de P y tenemos dos subcasos:
  - Si $Y = A$ entonces $X = B$ y por construcción de G_M, $vw \in M$, por tanto $vw \notin M \oplus P$, es decir, vw ya no formará parte del apareamiento actual.
  - Si $Y = B$ entonces $X = A$ y por construcción de G_M, $vw \notin M$, por tanto $vw \in M \oplus P$, es decir, vw formará parte del apareamiento actual. Ponemos $mate(v) = w$ y $mate(w) = v$.

Con esto tenemos el tercer punto resuelto y, además, de un modo similar al caso de las gráficas bipartitas. A continuación tenemos la descripción del algoritmo de apareamiento general. En el Listado 18 se muestra el procedimiento para obtener G_M a partir de la gráfica G y un apareamiento M dado. Identificamos al vértice s con $[s, A]$ y al vértice t con $[t, B]$.

Listado 18 Procedimiento convertGeneralInstance

procedure convertGeneralInstance;

Entrada: Una gráfica G y un apareamiento M de G.
Salida: La gráfica dirigida G_M.

```
1: V(G_M) := {[s, A], [t, B]};
2: F(G_M) := ∅;
3: for all v ∈ V(G) do
4:    V(G_M) := V(G_M) ∪ {[v, A], [v, B]};
5: end for
6: for all vw ∈ E(G) do
7:    if mate(v) = w y mate(w) = v then
8:       F(G_M) := F(G_M) ∪ {([v, A], [w, B]), ([w, A], [v, B])};
9:    else
10:      F(G_M) := F(G_M) ∪ {([v, B], [w, A]), ([w, B], [v, A])};
11:   end if
12: end for
13: for all v ∈ V(G) do
14:   if matched(v) = false then
15:      F(G_M) := F(G_M) ∪ {([s, A], [v, B]), ([v, A], [t, B])};
16:   end if
17: end for
```

El Listado 19 indica el procedimiento para recuperar la trayectoria aumentante en G y actualizar al mismo tiempo el apareamiento por medio de las etiquetas *matched* y *mate*.

En el Listado 20 se muestra el algoritmo de apareamiento máximo en gráficas generales o algoritmo de apareamiento general. Al igual que en los algoritmos especializados en gráficas bipartitas empleamos dos variables lógicas: $isMaximum$

Listado 19 Procedimiento updateMatching

procedure updateMatching;
Entrada: La trayectoria original $\bar{P} : s, [v_0, B], \ldots, [v_k, A], t$
Salida: $M := M \oplus P$, con $P : v_0, \ldots, v_k$ trayectoria M-aumentante.

```
1:  [w, X] := [t, B];
2:  while w ≠ s do
3:     [v, Y] := parent([w, X]);
4:     if w = t then
5:        matched(v) := true;
6:     else if v = s then
7:        matched(w) := true;
8:     else
9:        if Y = B then
10:          mate(v) := w;
11:          mate(w) := v;
12:       end if
13:    end if
14:    [w, X] := [v, Y];
15: end while
```

Listado 20 Algoritmo de apareamiento máximo en gráficas generales

Entrada: Una gráfica G.
Salida: Un apareamiento máximo M_{max}.

```
1:  M := ∅;
2:  isMaximum := false;
3:  found := false;
4:  i := 0;
5:  while isMaximum = false do
6:     i := i + 1;
7:     G_M := convertGeneralInstance(G, M);
8:     found := MDFS(G_M);
9:     if found = true then
10:       P̄ := reconstructPath([t, B], [s, A])
11:       M := updateMatching(P̄);
12:    else
13:       isMaximum := true;
14:    end if
15: end while
16: M_max := recoverMatching(G);
```

indica si el apareamiento actual es máximo y $found$ indica si existe una trayectoria original de s a t en G_M. También utilizamos el procedimiento del Listado 5 para recuperar esa trayectoria y el del Listado 16 para recuperar las aristas del apareamiento máximo.

Sólo nos resta analizar la complejidad del algoritmo de apareamiento general. Primero, es fácil ver que el tiempo de ejecución del procedimiento del Listado 18 tiene tiempo $O(n + m)$, donde $n = V(G)$ y $m = E(G)$.

Sabemos también que el tiempo de ejecución de $MDFS$ es $O(nm)$. Los procedimientos para reconstruir la trayectoria original de s a t y para actualizar el apareamiento también tienen tiempo $O(n + m)$ en el peor caso. Por tanto, el cuerpo del ciclo principal del algoritmo de apareamiento general tiene tiempo de ejecución $O(nm)$ y de aquí concluimos que el tiempo total de ejecución de este algoritmo es $O(n^2m)$, ya que el número de iteraciones es $O(n)$. Si la gráfica tiene $m = O(n^2)$ aristas, el tiempo de ejecución es $O(n^4)$.

Capítulo 7

Resultados Experimentales

En este capítulo presentamos algunos resultados empíricos divididos en dos grupos: experimentos con algoritmos de apareamiento, tanto generales como específicos, aplicados en gráficas bipartitas y experimentos con algoritmos generales de apareamiento en gráficas no bipartitas. Cada uno de esos algoritmos fue implementado en el lenguaje de programación Java Versión 1.4.0.

7.1 Resultados con gráficas bipartitas

Consideramos una gráfica bipartita $G = (A, B; E(G))$, donde A y B forman una bipartición de su conjunto de vértices $V(G)$. Todas las gráficas bipartitas con las cuales trabajamos fueron generadas en forma aleatoria tomando como parámetros dos enteros que representan la cardinalidad de los conjuntos A y B. Estos enteros fueron escogidos de modo que la bipartición definida al momento de generar cada gráfica estuviera lo más balanceada posible, es decir, que la diferencia entre el número de elementos de los conjuntos A y B fuera a lo más uno. Los algoritmos empleados para estos experimentos son:

- Algoritmo Modificado de Apareamiento en Gráficas Bipartitas, el cual revisamos en la Subsección 6.1.2 y lo identificamos como BM.
- Algoritmo de Apareamiento de Edmonds, el cual revisamos en el Apéndice B y lo identificamos como GE.
- Algoritmo de Apareamiento en Gráficas Generales, el cual revisamos en la Sección 6.2 y lo identificamos como GB.

La Tabla 7.1 muestra los tiempos de ejecución en milisegundos de los tres algoritmos aplicados sobre gráficas bipartitas desde 10 hasta 100 vértices. Notamos que los tiempos registrados para GE son en general mejores que los tiempos de BM, y a su vez estos son mejores que los de GB.

Tabla 7.1: Resultados experimentales con gráficas bipartitas

$\|A\|$	$\|B\|$	$\|V(G)\|$	$\|E(G)\|$	**BM**	**GE**	**GB**
5	5	10	16	0	0	0
5	6	11	17	0	0	0
6	6	12	19	0	0	0
6	7	13	18	0	0	0
7	7	14	28	0	0	50
7	8	15	30	0	0	50
8	8	16	35	60	0	60
8	9	17	40	60	0	50
9	9	18	37	60	0	60
9	10	19	44	50	0	60
10	10	20	45	50	0	60
10	11	21	59	110	0	110
11	11	22	60	110	0	110
11	12	23	74	110	0	110
12	12	24	72	110	0	160
12	13	25	81	110	50	160
13	13	26	87	110	60	160
13	14	27	84	160	60	220
14	14	28	99	110	50	220
14	15	29	96	160	50	220
15	15	30	120	170	60	270
15	16	31	118	170	60	280
16	16	32	119	170	60	330
16	17	33	140	170	60	330
17	17	34	152	220	60	330
17	18	35	156	220	60	380
18	18	36	162	270	60	430
18	19	37	174	270	60	440
19	19	38	175	270	60	490
19	20	39	180	270	60	490
20	20	40	204	330	60	660
20	21	41	214	330	60	660
21	21	42	222	330	60	660
21	22	43	226	330	60	710
22	22	44	239	390	60	710
22	23	45	259	390	110	820
23	23	46	256	390	110	880
23	24	47	266	390	110	930
24	24	48	281	390	110	980
24	25	49	295	440	60	1040
Tabla 7.1 - Continuación						

Tabla 7.1 - Continuación

$\|A\|$	$\|B\|$	$\|V(G)\|$	$\|E(G)\|$	**BM**	**GE**	**GB**
25	25	50	331	440	60	1100
25	26	51	327	440	110	1150
26	26	52	347	490	110	1310
26	27	53	369	490	110	1370
27	27	54	362	550	110	1430
27	28	55	389	550	110	1480
28	28	56	398	600	110	1600
28	29	57	394	550	110	1650
29	29	58	424	610	110	1700
29	30	59	429	600	110	1750
30	30	60	465	710	110	1870
30	31	61	462	660	110	1870
31	31	62	482	710	110	1980
31	32	63	467	710	110	1980
32	32	64	501	770	110	2080
32	33	65	573	770	110	2200
33	33	66	529	770	110	2260
33	34	67	558	820	110	2300
34	34	68	612	880	110	2530
34	35	69	591	880	110	2480
35	35	70	625	880	110	2690
35	36	71	623	930	110	2640
36	36	72	596	990	110	2750
36	37	73	689	990	110	3020
37	37	74	679	1040	170	3070
37	38	75	727	1040	170	3140
38	38	76	704	1040	170	3190
38	39	77	750	1040	170	3190
39	39	78	779	1150	170	3400
39	40	79	753	1150	170	3460
40	40	80	799	1210	170	3570
40	41	81	830	1210	170	3570
41	41	82	843	1210	170	3680
41	42	83	845	1270	170	3680
42	42	84	892	1310	170	4120
42	43	85	908	1370	170	4280
43	43	86	934	1370	170	4450
43	44	87	913	1430	170	4560
44	44	88	1000	1490	170	4660
44	45	89	970	1480	170	4650
45	45	90	994	1540	170	5050

Tabla 7.1 - Continuación

Tabla 7.1 - Continuación

$\|A\|$	$\|B\|$	$\|V(G)\|$	$\|E(G)\|$	**BM**	**GE**	**GB**
45	46	91	1010	1590	170	5110
46	46	92	1035	1590	170	5160
46	47	93	1081	1590	170	5170
47	47	94	1066	1650	220	5050
47	48	95	1137	1640	170	5380
48	48	96	1174	1810	220	5810
48	49	97	1172	1710	220	5810
49	49	98	1213	1750	220	6310
49	50	99	1201	1810	220	6200
50	50	100	1247	1870	220	6670

7.2 Resultados con gráficas no bipartitas

Para estos experimentos consideramos gráficas de tres tipos. Estas gráficas no representan el peor caso de ninguno de los tres algoritmos, pero lo importante es que se trata de ejemplares que cuentan con muchos ciclos impares y por tanto se puede esperar que los algoritmos tengan mayores dificultades para encontrar trayectorias aumentantes.

Gráficas de tipo 1. También las denominamos *n-ruedas de tipo 1.* Están constituidas por $n + 1$ vértices y $2n$ aristas. Explícitamente, una n-rueda de tipo 1 es una gráfica G con sus conjuntos de vértices y aristas definidos como sigue:

- $V(G) = \{v_1, v_2, \ldots, v_n, v_{n+1}\}$;
- $E(G) = \{v_i v_{i+1} \mid i = 1, \ldots, n\} \cup \{v_{n+1} v_i \mid i = 1, \ldots, n-1\} \cup \{v_1 v_n\}$.

En la Figura 7.1 se ilustra una 5-rueda de tipo 1.

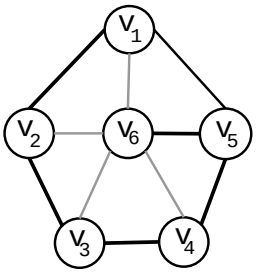

Figura 7.1: Ejemplo de una n-rueda de tipo 1

Gráficas de tipo 2. También las denominamos *n-ruedas de tipo 2*. Están constituidas por $n+3$ vértices y $2n+3$ aristas. Explícitamente, una n-rueda de tipo 2 es una gráfica G con sus conjuntos de vértices y aristas definidos como sigue:

- $V(G) = \{v_1, v_2, \ldots, v_n, v_{n+1}, v_{n+2}, v_{n+3}\}$;
- $E(G) = \{v_i v_{i+1} \mid i = 1, \ldots, n-1\} \cup \{v_{n+1} v_i \mid i = 1, \ldots, n-1\}$
  $\cup \{v_{n+3} v_{n+1}, v_{n+3} v_n, v_{n+2} v_n, v_{n+2} v_1, v_{n+2} v_{n+3}\}$.

En la Figura 7.2 se ilustra una 6-rueda de tipo 2.

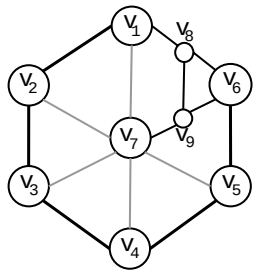

Figura 7.2: Ejemplo de una n-rueda de tipo 2

Gráficas de tipo 3. También las denominamos *n-ruedas de tipo 3*. Están constituidas por $3n+1$ vértices y $5n$ aristas. Explícitamente, una n-rueda de tipo 3 es una gráfica G con sus conjuntos de vértices y aristas definidos como sigue:

- $V(G) = \{v_1, v_2, \ldots, v_n, \ldots, v_{2n}, \ldots, v_{3n}, v_{3n+1}\}$;
- $E(G) = \{v_{(n+1)+i} v_i \mid i = 1, \ldots, n\} \cup \{v_{(n+1)+i} v_{n+1} \mid i = 1, \ldots, n\}$
  $\cup \{v_{(2n+1)+i} v_i \mid i = 1, \ldots, n-1\} \cup \{v_{(2n+1)+i} v_{i+1} \mid i = 1, \ldots, n-1\}$
  $\cup \{v_{(2n+1)+i} v_{(n+1)+i} \mid i = 1, \ldots, n\} \cup \{v_{3n+1} v_1, v_{3n+1} v_n\}$.

En la Figura 7.3 se ilustra una 4-rueda de tipo 3.

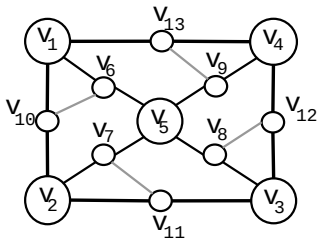

Figura 7.3: Ejemplo de una n-rueda de tipo 3

A continuación tenemos los resultados de los experimentos llevados a cabo con las gráficas definidas. La Tabla 7.2 muestra los tiempos de ejecución en milisegundos de los algoritmos GE y GB aplicados sobre n-ruedas de tipo 1; la Tabla 7.3, para n-ruedas de tipo 2 y la Tabla 7.4, para n-ruedas de tipo 3, donde n=3,. . . ,100.

Tabla 7.2: Resultados experimentales con n-ruedas de tipo 1

Rueda de tipo 1	$\|V(G)\|$	**GE**	**GB**
3-rueda	4	0	0
4-rueda	5	0	0
5-rueda	6	0	0
6-rueda	7	0	0
7-rueda	8	0	0
8-rueda	9	0	0
9-rueda	10	0	0
10-rueda	11	0	50
11-rueda	12	0	50
12-rueda	13	0	50
13-rueda	14	0	110
14-rueda	15	0	110
15-rueda	16	0	110
16-rueda	17	0	110
17-rueda	18	0	110
18-rueda	19	0	110
19-rueda	20	0	110
20-rueda	21	0	110
21-rueda	22	0	160
22-rueda	23	0	170
23-rueda	24	50	160
24-rueda	25	50	170
25-rueda	26	60	160
26-rueda	27	50	170
27-rueda	28	50	220
28-rueda	29	50	220
29-rueda	30	50	270
30-rueda	31	50	280
31-rueda	32	50	270
32-rueda	33	50	270
33-rueda	34	60	330
34-rueda	35	50	270
35-rueda	36	50	330
36-rueda	37	60	330
Tabla 7.2 - Continuación			

Tabla 7.2 - Continuación

Rueda de tipo 1	$\|V(G)\|$	**GE**	**GB**
37-rueda	38	50	380
38-rueda	39	50	380
39-rueda	40	60	390
40-rueda	41	50	440
41-rueda	42	60	430
42-rueda	43	60	440
43-rueda	44	60	490
44-rueda	45	60	490
45-rueda	46	60	490
46-rueda	47	60	540
47-rueda	48	60	600
48-rueda	49	60	600
49-rueda	50	60	660
50-rueda	51	60	660
51-rueda	52	60	660
52-rueda	53	60	710
53-rueda	54	110	710
54-rueda	55	110	770
55-rueda	56	110	770
56-rueda	57	110	830
57-rueda	58	110	880
58-rueda	59	110	930
59-rueda	60	110	930
60-rueda	61	110	940
61-rueda	62	110	990
62-rueda	63	110	980
63-rueda	64	110	1040
64-rueda	65	110	1090
65-rueda	66	110	1100
66-rueda	67	110	1150
67-rueda	68	110	1210
68-rueda	69	110	1260
69-rueda	70	110	1320
70-rueda	71	110	1310
71-rueda	72	110	1320
72-rueda	73	110	1370
73-rueda	74	110	1480
74-rueda	75	110	1490
75-rueda	76	110	1540
76-rueda	77	110	1590
77-rueda	78	110	1640
Tabla 7.2 - Continuación			

Tabla 7.2 - Continuación

Rueda de tipo 1	$\|V(G)\|$	**GE**	**GB**
78-rueda	79	160	1640
79-rueda	80	170	1710
80-rueda	81	160	1750
81-rueda	82	170	1810
82-rueda	83	160	1820
83-rueda	84	170	1870
84-rueda	85	170	1930
85-rueda	86	170	2010
86-rueda	87	170	2080
87-rueda	88	170	2150
88-rueda	89	170	2200
89-rueda	90	170	2200
90-rueda	91	170	2190
91-rueda	92	170	2310
92-rueda	93	170	2360
93-rueda	94	170	2420
94-rueda	95	170	2410
95-rueda	96	170	2470
96-rueda	97	170	2530
97-rueda	98	170	2580
98-rueda	99	170	2690
99-rueda	100	220	2750
100-rueda	101	220	2740

Tabla 7.3: Resultados experimentales con n-ruedas de tipo 2

Rueda de tipo 2	$\|V(G)\|$	**GE**	**GB**
3-rueda	6	0	0
4-rueda	7	0	0
5-rueda	8	0	0
6-rueda	9	0	0
7-rueda	10	0	0
8-rueda	11	0	0
9-rueda	12	0	0
10-rueda	13	0	60
11-rueda	14	0	50
12-rueda	15	0	60
13-rueda	16	0	50
14-rueda	17	0	50
15-rueda	18	0	110
Tabla 7.3 - Continuación			

Tabla 7.3 - Continuación

Rueda de tipo 2	$\|V(G)\|$	**GE**	**GB**
16-rueda	19	0	110
17-rueda	20	0	110
18-rueda	21	0	110
19-rueda	22	0	160
20-rueda	23	0	170
21-rueda	24	0	170
22-rueda	25	0	170
23-rueda	26	60	170
24-rueda	27	50	170
25-rueda	28	50	220
26-rueda	29	50	220
27-rueda	30	60	220
28-rueda	31	50	270
29-rueda	32	50	270
30-rueda	33	50	270
31-rueda	34	50	330
32-rueda	35	50	330
33-rueda	36	60	380
34-rueda	37	110	380
35-rueda	38	110	440
36-rueda	39	50	440
37-rueda	40	50	440
38-rueda	41	110	440
39-rueda	42	110	500
40-rueda	43	110	490
41-rueda	44	110	500
42-rueda	45	110	550
43-rueda	46	110	550
44-rueda	47	110	610
45-rueda	48	110	610
46-rueda	49	110	600
47-rueda	50	110	650
48-rueda	51	110	660
49-rueda	52	110	710
50-rueda	53	110	720
51-rueda	54	110	770
52-rueda	55	110	770
53-rueda	56	110	820
54-rueda	57	110	820
55-rueda	58	110	870
56-rueda	59	110	880
Tabla 7.3 - Continuación			

Tabla 7.3 - Continuación

Rueda de tipo 2	$\|V(G)\|$	**GE**	**GB**
57-rueda	60	110	940
58-rueda	61	110	940
59-rueda	62	110	990
60-rueda	63	170	990
61-rueda	64	110	990
62-rueda	65	160	1040
63-rueda	66	170	1040
64-rueda	67	110	1090
65-rueda	68	170	1090
66-rueda	69	110	1090
67-rueda	70	170	1150
68-rueda	71	170	1150
69-rueda	72	170	1320
70-rueda	73	170	1370
71-rueda	74	170	1380
72-rueda	75	160	1420
73-rueda	76	170	1380
74-rueda	77	160	1420
75-rueda	78	160	1420
76-rueda	79	170	1540
77-rueda	80	160	1590
78-rueda	81	170	1650
79-rueda	82	170	1700
80-rueda	83	170	1700
81-rueda	84	170	1770
82-rueda	85	170	1770
83-rueda	86	170	1810
84-rueda	87	170	1810
85-rueda	88	170	1930
86-rueda	89	170	1980
87-rueda	90	170	2040
88-rueda	91	170	2090
89-rueda	92	170	2080
90-rueda	93	170	2140
91-rueda	94	220	2200
92-rueda	95	220	2310
93-rueda	96	220	2320
94-rueda	97	220	2320
95-rueda	98	220	2360
96-rueda	99	220	2480
97-rueda	100	220	2480
Tabla 7.3 - Continuación			

Tabla 7.3 - Continuación

Rueda de tipo 2	$\|V(G)\|$	**GE**	**GB**
98-rueda	101	220	2590
99-rueda	102	220	2650
100-rueda	103	220	2740

Tabla 7.4: Resultados experimentales con n-ruedas de tipo 3

Rueda de tipo 3	$\|V(G)\|$	**GE**	**GB**
3-rueda	10	0	0
4-rueda	13	0	0
5-rueda	16	0	50
6-rueda	19	0	50
7-rueda	22	0	60
8-rueda	25	50	60
9-rueda	28	50	110
10-rueda	31	50	110
11-rueda	34	50	220
12-rueda	37	50	220
13-rueda	40	50	220
14-rueda	43	110	270
15-rueda	46	110	330
16-rueda	49	110	330
17-rueda	52	110	380
18-rueda	55	110	380
19-rueda	58	110	490
20-rueda	61	160	550
21-rueda	64	160	600
22-rueda	67	160	660
23-rueda	70	170	710
24-rueda	73	170	770
25-rueda	76	220	830
26-rueda	79	220	880
27-rueda	82	220	930
28-rueda	85	220	1040
29-rueda	88	220	1100
30-rueda	91	220	1160
31-rueda	94	220	1320
32-rueda	97	220	1480
33-rueda	100	270	1650
34-rueda	103	280	1700
35-rueda	106	270	1810
Tabla 7.4 - Continuación			

Tabla 7.4 - Continuación

Rueda de tipo 3	$\|V(G)\|$	**GE**	**GB**
36-rueda	109	280	1870
37-rueda	112	280	1980
38-rueda	115	280	2040
39-rueda	118	280	2120
40-rueda	121	280	2200
41-rueda	124	330	2430
42-rueda	127	330	2540
43-rueda	130	330	2650
44-rueda	133	330	2800
45-rueda	136	330	2910
46-rueda	139	330	3020
47-rueda	142	380	3190
48-rueda	145	380	3240
49-rueda	148	380	3350
50-rueda	151	380	3460
51-rueda	154	380	3580
52-rueda	157	380	3800
53-rueda	160	380	4070
54-rueda	163	390	4120
55-rueda	166	380	4280
56-rueda	169	380	4450
57-rueda	172	440	4510
58-rueda	175	440	4620
59-rueda	178	440	4850
60-rueda	181	440	5050
61-rueda	184	440	5280
62-rueda	187	440	5410
63-rueda	190	440	5500
64-rueda	193	440	5660
65-rueda	196	490	5770
66-rueda	199	490	5820
67-rueda	202	500	6060
68-rueda	205	490	6260
69-rueda	208	490	6470
70-rueda	211	500	6610
71-rueda	214	490	6820
72-rueda	217	500	7050
73-rueda	220	500	7230
74-rueda	223	550	7380
75-rueda	226	550	7510
76-rueda	229	550	7640
Tabla 7.4 - Continuación			

Tabla 7.4 - Continuación

Rueda de tipo 3	$\|V(G)\|$	**GE**	**GB**
77-rueda	232	550	7780
78-rueda	235	610	7910
79-rueda	238	610	8020
80-rueda	241	610	8140
81-rueda	244	610	8270
82-rueda	247	610	8400
83-rueda	250	610	8530
84-rueda	253	610	8680
85-rueda	256	610	9010
86-rueda	259	610	9210
87-rueda	262	610	9380
88-rueda	265	610	9560
89-rueda	268	610	9710
90-rueda	271	660	9970
91-rueda	274	660	10230
92-rueda	277	660	10580
93-rueda	280	660	10890
94-rueda	283	660	11130
95-rueda	286	660	11470
96-rueda	289	660	11780
97-rueda	292	660	11930
98-rueda	295	660	12280
99-rueda	298	660	12670
100-rueda	301	710	12950

Observamos que los tiempos de ejecución de GE, el algoritmo de Edmonds, son mucho mejores que los registrados por GB, el algoritmo general de Blum que desarrollamos en este trabajo. Cabe señalar que al verificar las soluciones de ambos algoritmos, las trayectorias aumentantes que encontraba GE por lo general fueron de una o tres aristas, por lo cual ni siquiera fue necesario el empleo de los conjuntos ajenos para resolver el problema en estas gráficas; en cambio, las trayectorias aumentantes halladas por GB al principio consistían de una arista pero a partir de determinada iteración o fase, las trayectorias eran cada vez más largas. Esto tiene que ver directamente con el modo de operación de MDFS que, como su nombre lo indica, resultó de una modificación del algoritmo de búsqueda en profundidad. En contraste, el algoritmo de Edmonds emplea de forma indirecta la búsqueda en amplitud para buscar trayectorias aumentantes, y esto puede explicar por qué las trayectorias en general son más cortas.

Conclusiones

La reducción del problema de trayectorias aumentantes al problema RP, tratada en el Capítulo 2, nos proporciona una herramienta fundamental para la comprensión de los algoritmos del Capítulo 6, que surgen de acuerdo con el material de Blum [5]. La construcción de la gráfica bipartita dirigida a partir de una gráfica general está basada en ideas semejantes a las que se siguen para el caso específico de gráficas bipartitas. También notamos que la forma en como se recuperan las trayectorias aumentantes en ambos casos son parecidas. Por ello remarcamos la importancia de revisar el caso específico antes del caso general. De hecho, siempre que se trabaje con gráficas bipartitas, va a ser preferible aplicar el algoritmo especializado al general por dos razones principales:

- La gráfica bipartita dirigida para el caso general tiene un poco más del doble de vértices y de arcos que la gráfica que se construye para el caso específico, por lo cual se requiere mayor cantidad de memoria para almacenar la primera que para la segunda.
- El algoritmo DFS empleado para la solución en el caso específico es más sencillo que MDFS, en el sentido de que éste maneja una alternativa por cada tipo de arco existente en la gráfica dirigida. Además, al aplicar el algoritmo MDFS puede ocurrir varias veces el caso $2.c.i$ que obliga a usar los conjuntos ajenos, lo cual no ocurre en el caso específico.

El único inconveniente del algoritmo para el caso específico es que es necesario conocer la estructura de la gráfica en cuestión antes de aplicárselo, es decir, tener la certeza de que es bipartita y haberle identificado una bipartición. Con respecto a los resultados prácticos, ambos algoritmos tienen un buen desempeño en las primeras fases, pues se pueden encontrar trayectorias aumentantes de una arista, pero a partir de determinada fase, las trayectorias tienden a ser cada vez más largas hasta la última fase.

Con respecto al algoritmo de Edmonds [9], los tiempos de ejecución en la práctica reflejan una mejor eficiencia que en los algoritmos revisados en el Capítulo 6. Sobre los resultados del Capítulo, mencionamos que el algoritmo de Edmonds sigue una estrategia similar a la búsqueda en amplitud BFS para la búsqueda de trayectorias aumentantes, por lo cual éstas son en general más cortas que las que se obienen aplicando cualquiera de los algoritmos de Blum. Aun

con las gráficas que alcancen el peor caso teórico del algoritmo de Edmonds, es muy probable que esta tendencia siga prevaleciendo en la práctica. Otro detalle importante es que la implementación considerada para este algoritmo solamente construye una gráfica dirigida a partir de la gráfica original, lo cual representa usar menos memoria que el algoritmo general de apareamientos que aquí examinamos, ya que aparte de que la gráfica bipartita dirigida tiene más vértices y aristas, esta construcción se realiza una vez por cada fase.

Sin embargo, nuestro enfoque representa una alternativa para todas aquellas personas que han seguido la historia de los algoritmos de apareamiento y que han encontrado el término *blossoms* en buena parte de la literatura. Para continuar con los trabajos sobre apareamientos con este enfoque, quizá sea conveniente intentar el diseño de algoritmos que busquen trayectorias originales en la gráfica dirigida bipartita a la cual nos referimos en este proyecto pero operando con alguna modificación de la búsqueda en amplitud y reducir de alguna forma el tiempo requerido para la construcción de las gráficas auxiliares, tal como vimos en la solución de apareamientos para gráficas bipartitas. También sería conveniente buscar la forma de reducir el espacio en memoria reservado para las gráficas auxiliares.

Apéndice A

Definiciones Básicas

El objetivo de este capítulo es presentar las definiciones básicas de la teoría de gráficas para una mejor comprensión de los resultados de este proyecto.

A.1 Gráficas

Definición A.1. Una gráfica consiste de un conjunto finito no vacío de vértices y un conjunto de aristas, el cual puede ser vacío y está formado por pares no ordenados de vértices. Denotamos al conjunto de vértices de la gráfica G por $V(G)$ y a su conjunto de aristas por $E(G)$.

Cada arista $(u, v) \in E(G)$ es un par no ordenado que denotamos simplemente por uv. Si $e = uv$ es una arista de G, decimos que u y v son los extremos de e y que los vértices u y v son adyacentes.

Cada gráfica tiene un diagrama asociado. Representamos a los vértices por medio de círculos o puntos y las aristas mediante líneas entre los pares de puntos correspondientes.

Ejemplo.

- Una gráfica definida por los conjuntos:
  $V(G) = \{u, v, w, x, y, z\}$ y $E(G) = \{uv, wx, yz, uw\}$ se representa con el diagrama de la Figura A.1.

Definición A.2. Sea G una gráfica. Un **camino** en G es una secuencia alternada de vértices y aristas $P : v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k$ tal que

$$e_i = v_{i-1}v_i, \text{ para } 1 \leq i \leq k.$$

Si $v_0 = u$ y $v_k = w$, decimos que P es un camino de u a w.

Definición A.3. Sea G una gráfica y sea P un camino en G. La longitud de P, denotada por $|P|$, representa el número de aristas que aparecen en P. A un camino de longitud 0 se le denomina **camino trivial**.

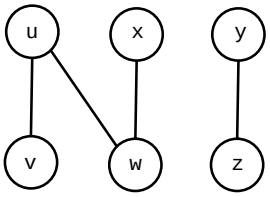

Figura A.1: Representación de una gráfica

Como las aristas de una gráfica constituyen un conjunto, entonces los vértices que aparecen en un camino determinan sus aristas. Por lo tanto, un camino puede ser expresado por su conjunto de vértices como $P : v_0, v_1, \ldots, v_k$ con $v_i v_{i+1} \in E(G)$, para $0 \leq i < k$.

Definición A.4. Sea G una gráfica. Una **trayectoria** en G es un camino $P : v_0, v_1, \ldots, v_k$ en G tal que $v_i \neq v_j$, para $0 \leq i < j \leq k$. Si $v_0 = u$ y $v_k = w$, decimos que P es una trayectoria de u a w.
También expresamos una trayectoria de v_0 a v_k como $[v_0, v_1, \ldots, v_k]$.

Definición A.5. Sea G una gráfica y sea $P : v_0, v_1, \ldots, v_k$ una trayectoria en G tal que $|P| \geq 3$. Decimos que P es un **ciclo simple** si $v_0 = v_k$.

Ejemplos:

1. En la gráfica de la Figura A.2(a) se ilustra la trayectoria $P_1 : u, w, z, x, v$ cuya longitud es cuatro.
2. En la gráfica de la Figura A.2(b) se muestra el camino $P_2 : y, x, w, y, z$, el cual no es trayectoria, ya que se repite el vértice y.
3. En la gráfica de la Figura A.2(c) se observa el ciclo simple $P_3 : w, z, x, w$.

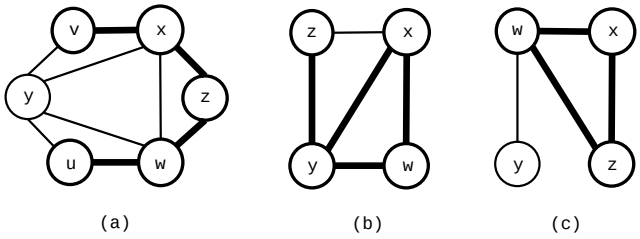

Figura A.2: Caminos en gráficas

Definición A.6. Sea G una gráfica y sea $u \in V(G)$. El **grado** de u en G, denotado por $d_G(u)$ está dado por:

$$d_G(u) = |\{v \in V(G) : uv \in E(G)\}|.$$

Para toda gráfica G, denotamos como $\Delta(G)$ al grado máximo de los vértices de G y como $\delta(G)$ al grado mínimo, esto es:

$$\delta(G) \leq d_G(v) \leq \Delta(G), \ \ \forall\ v \in V(G).$$

Definición A.7. Sean G y H gráficas. Decimos que H es **subgráfica** de G si $V(H) \subseteq V(G)$ y $E(H) \subseteq E(G)$.

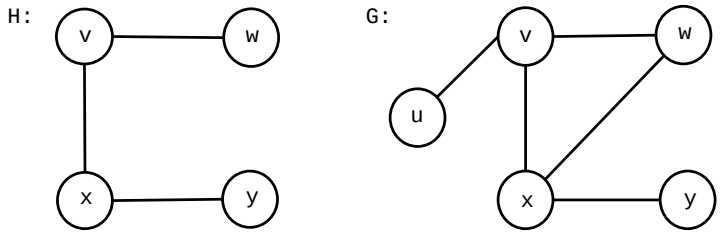

Figura A.3: H es subgráfica de G

Definición A.8. Sea G una gráfica y sea $S \subseteq V(G)$, $S \neq \emptyset$. **La subgráfica inducida** por S, la cual denotamos por $\langle S \rangle$, tiene como conjunto de vértices $V(\langle S \rangle) = S$ y como conjunto de aristas $E(\langle S \rangle) = \{uv \in E(G) : u, v \in S\}$.

Definición A.9. Sea G una gráfica y sea $X \subseteq E(G)$, $X \neq \emptyset$. La subgráfica inducida por X, la cual denotamos por $\langle X \rangle$, tiene como conjunto de vértices $V(\langle X \rangle) = \{v \in V(G) : \exists\ e \in X$ tal que v es extremo de $e\}$ y como conjunto de aristas $E(\langle X \rangle) = X$.

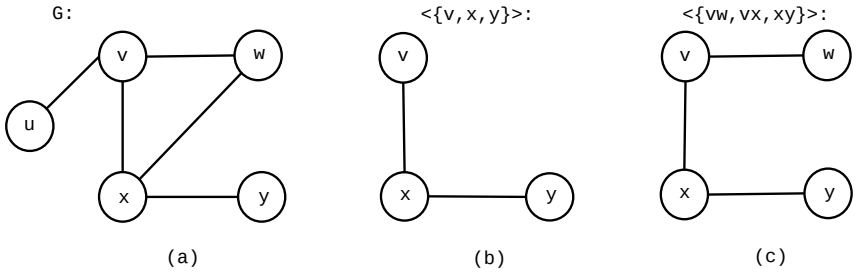

Figura A.4: Subgráficas inducidas por vértices y aristas de G

Ejemplos:

1. La Figura A.4(b) ilustra la subgráfica inducida por el conjunto de vértices $\{v, x, y\}$ de la gráfica que aparece en la Figura A.4(a).

2. La Figura A.4(c) muestra la subgráfica inducida por el conjunto de aristas $\{vw, vx, xy\}$ de la gráfica mostrada en la Figura A.4(a).

Definición A.10. Una subgráfica H de una gráfica G es una **subgráfica generadora** o **subgráfica de expansión** de G si $V(H) = V(G)$.

Definición A.11. Sea G una gráfica y sean $u, v \in V(G)$. Decimos que v es **alcanzable** desde u si existe en G una trayectoria de u a v. Esto constituye una relación de equivalencia sobre $V(G)$, por tanto, induce una partición $V_1, V_2, \ldots, V_k$ de $V(G)$. Las subgráficas $\langle V_1 \rangle, \langle V_2 \rangle, \ldots, \langle V_k \rangle$ son las **componentes conexas** de G. Decimos que G es **conexa** si $k = 1$, es decir, si para cualesquiera dos vértices u, v de G, v es alcanzable desde u. En otro caso, decimos que G no es conexa.

En la Figura A.5 se muestra una gráfica con cuatro componentes conexas, las cuales son: $\langle \{v, w, x, y\} \rangle, \langle \{z, t\} \rangle, \langle \{s\} \rangle$ y $\langle \{u\} \rangle$.

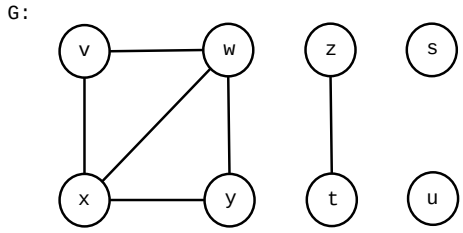

Figura A.5: Componentes conexas de la gráfica G

Definición A.12. Sea G una gráfica. Decimos que G es **bipartita** si $V(G)$ puede ser dividido en dos conjuntos no vacíos A y B tales que:

- $A \cup B = V(G)$;
- $A \cap B = \varnothing$;
- $\forall\ uv \in E(G) :\ u \in A$ y $v \in B$ o viceversa.

Decimos entonces que los conjuntos A y B forman la bipartición de la gráfica G, la cual denotamos por $G = (A, B; E(G))$.

En la Figura A.6 se muestra una gráfica G donde $V(G)$ puede ser dividido en los conjuntos $A = \{u, v, w\}$ y $B = \{x, y, z\}$ de modo que toda arista une un vértice en A con otro vértice en B.

Observamos que en una gráfica conexa bipartita G, para cualquier trayectoria de u a v, se tiene lo siguiente:

- Si la trayectoria de u a v tiene longitud par entonces $u, v \in A$ o $u, v \in B$.
- En otro caso se tiene que $u \in A$ y $v \in B$ o viceversa.

Una característica especial de las gráficas bipartitas es que cualquiera de sus ciclos simples debe contener un número par de aristas, es decir, todos sus ciclos deben tener longitud par.

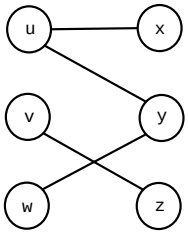

Figura A.6: Una gráfica bipartita

Definición A.13. Sea G una gráfica. Decimos que G es un **árbol** si es conexa y no tiene ciclos simples.

Definición A.14. Sea G una gráfica. Decimos que G es un **bosque** si no tiene ciclos simples.

Las componentes conexas de un bosque son árboles y un árbol es un bosque. Una característica especial de los árboles es que para cualquier par de vértices existe una única trayectoria que los conecta.

A.2 Gráficas dirigidas

Definición A.15. Una gráfica dirigida consiste de un conjunto finito no vacío de vértices y un conjunto de arcos o flechas, el cual puede ser vacío y está formado por pares ordenados de vértices. Denotamos al conjunto de vértices de la gráfica dirigida D por $V(D)$ y a su conjunto de arcos por $F(D)$.

Estas gráficas también tienen un diagrama asociado. Representamos a los vértices análogamente al caso no dirigido pero usamos flechas en lugar de líneas para representar los arcos.

Si $e = (u, v)$ es un arco de D, decimos que u es el extremo inicial de e y v es su extremo final.

La gráfica subyacente de una gráfica dirigida se obtiene de ésta remplazando todos los arcos (u, v) o (v, u) por la arista uv.

Ejemplo:

- Una gráfica dirigida definida por los conjuntos:
  $V(D) = \{u, v, w, x, y, z\}$ y $F(D) = \{(u, v), (v, u), (w, v), (x, y), (y, z)\}$ se representa con el diagrama de la Figura A.7(a). En la Figura A.7(b) se ilustra la gráfica subyacente de la gráfica indicada anteriormente.

Definición A.16. Sea D una gráfica dirigida. Un **camino dirigido** en D es una secuencia alternada de vértices y arcos $P : v_0, a_1, v_1, a_2, v_2 \ldots, v_{k-1}, a_k, v_k$

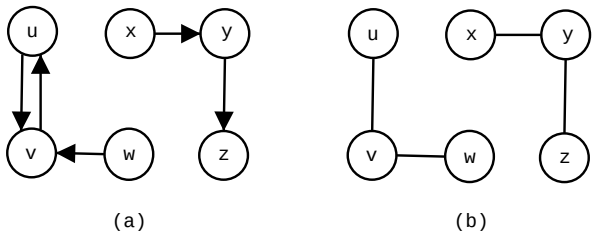

Figura A.7: Una gráfica dirigida y su gráfica subyacente

tal que $a_i = (v_{i-1}, v_i)$, para $1 \leq i \leq k$.
Si $v_0 = u$ y $v_k = w$, decimos que P es un camino dirigido de u a w.

Definición A.17. Sea D una gráfica dirigida y P un camino dirigido en D. La longitud de P, denotada por $|P|$, representa el número de arcos de P.

También podemos simplificar la notación y expresar un camino dirigido como $P : v_0, v_1, \ldots, v_k$ con $(v_i, v_{i+1}) \in F(D)$, para $0 \leq i < k$.

Definición A.18. Sea D una gráfica dirigida. Una **trayectoria dirigida** en D es un camino dirigido $P : v_0, v_1, \ldots, v_k$ en D tal que $v_i \neq v_j$, para $0 \leq i < j \leq k$. Si $v_0 = u$ y $v_k = w$, decimos que P es una trayectoria dirigida de u a w. También expresaremos una trayectoria dirigida de v_0 a v_k como $[v_0, v_1, \ldots, v_k]$.

Definición A.19. Sea D una gráfica dirigida y sea $P : v_0, v_1, \ldots, v_k$ un camino dirigido en D tal que $|P| \geq 2$. Decimos que P es un **ciclo dirigido** o **circuito** si $v_0 = v_k$ y $v_1, v_2, \ldots, v_k$ son vértices distintos.

Definición A.20. Sea D una gráfica dirigida y sean $v, w \in V(D)$. Decimos que v es un **predecesor** de w y que w es un **sucesor** de v si existe una trayectoria dirigida de v a w en D. En este caso también decimos que w es **alcanzable** desde v. Si además $(v, w) \in F(D)$, entonces v es un **predecesor directo** de w y w es un **sucesor directo** de v.

Definición A.21. Sea D una gráfica dirigida y $v \in V(D)$. El **ingrado** de v en D, denotado por $\delta_D^-(v)$, es el número de predecesores directos de v en D. El **exgrado** de v en D, denotado por $\delta_D^+(v)$, es el número de sucesores directos de v en D.

Ejemplos:

1. En la gráfica dirigida de la Figura A.8(a) se ilustra la trayectoria dirigida $P_1 : u, v, x$ de u a x, por lo cual tenemos que x es alcanzable desde u.

2. En la gráfica dirigida de la Figura A.8(b) se muestran los ciclos dirigidos $P_2 : u, w, u$ y $P_3 : w, v, x, w$.

3. En la misma figura, observamos que v tiene ingrado uno y exgrado dos, ya que w es su único predecesor directo y los vértices x y u son sus sucesores directos.

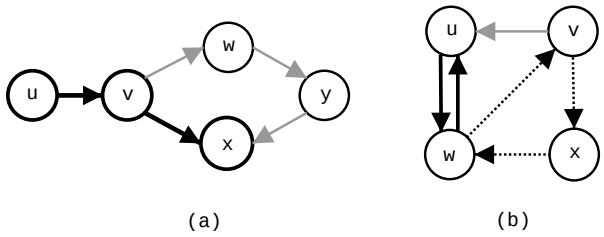

Figura A.8: Caminos en gráficas dirigidas

Definición A.22. Sea D una gráfica dirigida. Decimos que D es bipartita si $V(D)$ puede ser dividido en dos conjuntos no vacíos A y B tales que:

- $A \cup B = V(D)$;
- $A \cap B = \varnothing$;
- $\forall\ (u, v) \in F(D) : u \in A$ y $v \in B$ o viceversa.

Decimos entonces que los conjuntos A y B forman la bipartición de la gráfica dirigida D, la cual denotamos por $D = (A, B; F(D))$.

En una gráfica dirigida bipartita D, para cualesquiera vértices $u, v \in V(D)$ tales que u es alcanzable desde v, se tiene lo siguiente:

- Si la trayectoria dirigida de v a u tiene longitud par entonces $u, v \in A$ o $u, v \in B$.
- En otro caso se tiene que $u \in A$ y $v \in B$ o viceversa.

Definición A.23. Una gráfica dirigida D es **asimétrica** si

$$\forall\ u, v \in V(D)\text{:}\ (u, v) \in F(D) \Rightarrow (v, u) \notin F(D).$$

Definición A.24. Una gráfica dirigida D es **simétrica** si

$$\forall\ u, v \in V(D)\text{:}\ (u, v) \in F(D) \Leftrightarrow (v, u) \in F(D).$$

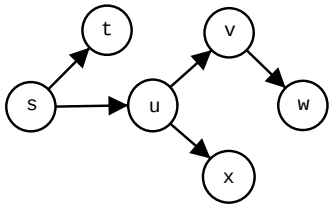

Figura A.9: Un árbol enraizado

Definición A.25. Sea D una gráfica dirigida. Decimos que D es un **árbol dirigido** si es asimétrica y su gráfica subyacente es un árbol.

Definición A.26. Sea D un árbol dirigido. Decimos que D es un **árbol enraizado** si $\exists\ s \in V(D)$ tal que $\forall\ t \in V(D)$, t es alcanzable desde s. A este vértice lo llamamos **raíz** de D y en este caso decimos que D es un árbol enraizado en s. También se le conoce a D como arborescencia.

En la gráfica dirigida de la Figura A.9 se observa que todo vértice es alcanzable desde s. Además, esa gráfica es asimétrica, puesto que para cada par de vértices existe a lo más un arco que los une. También su gráfica subyacente es un árbol, por lo tanto, la gráfica ilustrada es un árbol enraizado cuya raíz es el vértice s.

Apéndice B

El algoritmo de apareamiento de Edmonds

En este capítulo revisaremos la estrategia empleada por Jack Edmonds [9] para resolver el problema de apareamiento máximo para gráficas en general. Tal estrategia está basada en hallar trayectorias aumentantes con respecto a un apareamiento dado.

Una forma de buscar trayectorias aumentantes podría ser la siguiente: Supongamos que M es el apareamiento actual, entonces tomamos un vértice x que no sea M-apareado, revisamos si existe un vértice y adyacente a x que sea M-apareado y continuamos ahora desde el vértice que resulta ser la pareja de y, marcando a estos vértices como visitados. Si encontramos otro vértice que no sea M-apareado, entonces tendríamos una trayectoria M-aumentante. Sin embargo, esta estrategia tiene ciertos inconvenientes, como el que veremos a continuación:

Observemos la gráfica de la Figura B.1. Si empezamos a buscar trayectorias aumentantes desde v_1, vemos que $v_3v_4 \in M$, y por tanto llegamos a v_4. Ahora tenemos dos opciones: tanto v_5 como v_6 están apareados, pues $v_5v_6 \in M$. Si consideramos a v_5, siguiendo la estrategia planteada, solo podríamos seguir hacia v_6 y concluir que no existe trayectoria M-aumentante. En cambio, si tomamos a v_6, seguimos hacia v_5 y después a v_7. Como v_7 no es M-apareado, concluimos que $[v_1, v_3, v_4, v_6, v_5, v_7]$ es una trayectoria M-aumentante.

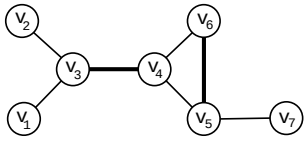

Figura B.1: Una gráfica con dificultades para hallar trayectorias aumentantes

Entonces el orden en que son examinados los vértices puede afectar de manera significativa el resultado, lo cual no debería pasar. No obstante, es posible resolver esta situación. Primero revisamos los aspectos primordiales para comprender el algoritmo de Edmonds y después nos enfocamos en su implementación.

B.1 Fundamentos básicos

Definición B.1. Sea G una gráfica y M un apareamiento de G. Una **flor** de G consiste de:

- Una trayectoria M-alternante $P : u, \ldots, v$ de longitud par tal que u no es M-apareado. Posiblemente $u = v$.
- Un ciclo $B : w_0, w_1, \ldots, w_{r-1}, w_0$ de longitud impar tal que $w_0 = v$, $[w_0, w_1, \ldots, w_{r-1}]$ y $[w_1, \ldots, w_{r-1}, w_0]$ son trayectorias M-alternantes y las aristas w_0w_1 y $w_{r-1}w_0$ no están en M.

Al ciclo B se le conoce como *blossom* de G con respecto a M y al vértice v se le conoce como la *base* de B. En la Figura B.2 se muestra la forma general de una flor en una gráfica. Las aristas gruesas forman parte del apareamiento M.

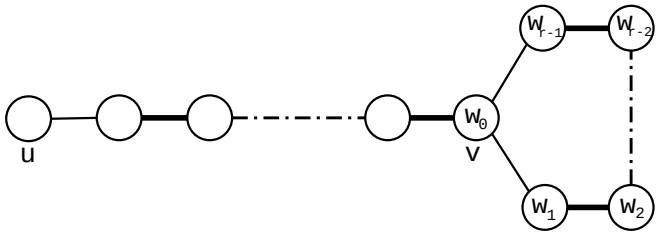

Figura B.2: Forma general de una flor en una gráfica

El algoritmo propuesto por Edmonds, además de buscar trayectorias aumentantes, contempla la posibilidad de encontrar *blossoms* con respecto al apareamiento actual en la gráfica dada G. Cuando esto ocurre, se construye una gráfica G/B a partir de G reduciendo todos los vértices de B en un sólo vértice como sigue:

- $V(G/B) = (V(G) \setminus V(B)) \cup \{b\}$, con b un vértice nuevo.
- $E(G/B) = (E(G) \setminus \{uv : u \in V(B) \text{ o } v \in V(B)\})$
  $\cup \{wb : w \notin V(B) \text{ y } \exists\, y \in V(B) \text{ adyacente a w}\}$.

Al proceso de construcción de esta gráfica se le conoce como reducción o compresión del *blossom* B.

A la gráfica para la cual deseamos obtener un apareamiento máximo le llamamos gráfica original y cualquier gráfica que sea producto de una o varias reducciones de *blossoms* le llamamos gráfica reducida.

El Teorema B.1 proporciona una justificación de las reducciones de *blossoms* y por ende, el fundamento del algoritmo de Edmonds. Una trayectoria aumentante en una gráfica reducida puede ser transformada en una trayectoria aumentante en la gráfica original, expandiendo los *blossoms* en el orden inverso a su reducción y reconectando las partes separadas de la trayectoria de manera conveniente por cada reducción. La demostración se puede consultar en el material presentado en [16, 19].

Teorema B.1. Si $G' = G/B$, donde B es un *blossom* de G, entonces G' contiene una trayectoria aumentante si y sólo si G también contiene una trayectoria aumentante.

Definición B.2. Un árbol T es llamado M-alternante si contiene trayectorias alternantes, las cuales tienen como extremo en común a un vértice distinguido x, el cual es llamado raíz de T y además, es el único vértice no M-apareado en T. Decimos que T está enraizado en x.

Definición B.3. Sea T un árbol M-alternante y v un vértice de T. Decimos que v es **vértice par** si la trayectoria alternante que lo conecta con la raíz de T tiene longitud par. En otro caso, decimos que v es **vértice impar**.

El algoritmo construye un bosque de árboles alternantes, enraizados en vértices que no son apareados con respecto al apareamiento actual y comprime los *blossoms* en cuanto los detecta. Cada vértice v será etiquetado como par o impar de acuerdo con la definición B.3. Sea M el aparemiento actual. Inicialmente los vértices que no son M-apareados se etiquetan como pares y a los apareados no se les etiqueta. Para cualquier vértice v apareado denotamos por $mate(v)$ al vértice w tal que $vw \in M$. El proceso se repite hasta que ya no queden aristas por examinar.

Sea $e = uv$ una arista no examinada tal que u es par. Hay varias posibilidades:

1. Si v no está etiquetado, lo etiquetamos como impar. Puesto que v está apareado, etiquetamos a $mate(v) = w$ como par y de esta forma extendemos el árbol alternante donde está u, con las aristas uv y vw.

2. Si v está etiquetado como par y ambos extremos están en distintos árboles alternantes, consideramos lo siguiente: sean r_u, r_v las raíces de los árboles que contienen respectivamente a u y a v. Entonces la unión de la trayectoria $[r_u, \ldots, u]$ junto con e y la trayectoria $[v, \ldots . r_v]$ forman una trayectoria M-aumentante, puesto que $e \notin M$ y tanto r_u como r_v no son M-apareados. Actualizamos el apareamiento tomando la diferencia simétrica de las aristas de la trayectoria encontrada y las de M para repetir el proceso.

3. Si v está etiquetado como par y ambos extremos están en el mismo árbol alternante, entonces hemos detectado la existencia de un *blossom*, el cual reducimos así: supongamos que x es la raíz del árbol alternante; entonces la base del *blossom* es un vértice b que aparece tanto en la trayectoria alternante de x a u como en la de x a v y es el más lejano a x con esa propiedad. A ese vértice le conocemos como el predecesor común más cercano de u y v. El resto de los vértices del *blossom* serán todos los vértices que están en las trayectorias de b a u y de b a v. Etiquetamos b como par y repetimos el proceso con la gráfica reducida.

B.2 Ejemplo

Consideremos la gráfica de la Figura B.3. Las aristas del apareamiento actual M aparecen remarcadas y tenemos que los vértices v_{13}, v_{14} y v_{15} no son M-apareados y por tanto están etiquetados como vértices pares. Comenzando por v_{13}, podemos marcar a v_7 y v_{12} como impares y por ende, podemos etiquetar v_8 y v_{11} como pares. Seguimos con v_8 y al examinar la arista v_8v_{11} descubrimos el *blossom* $B_1 : v_{13}, v_7, v_8, v_{11}, v_{12}, v_{13}$. La base es v_{13} puesto que es el predecesor más cercano que tienen v_8 y v_{11}.

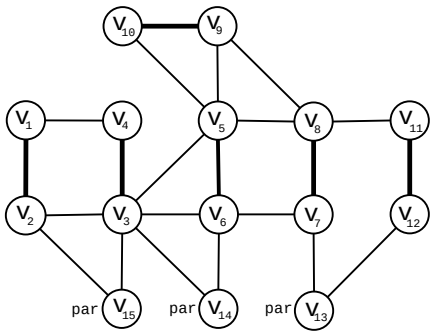

Figura B.3: Ejemplo de aplicación del algoritmo de Edmonds

La construcción de la gráfica $G' = G/B_1$ se observa en la Figura B.4. Notamos que b_1 es el vértice nuevo que representa a todos los vértices del *blossom* B_1 y es etiquetado como par. Empezando con este vértice, podemos marcar a v_9 y v_6 como impares y por tanto a v_{10} y v_5 como pares. Luego revisamos la arista $v_{10}v_5$ y encontramos el *blossom* $B_2 : b_1, v_9, v_{10}, v_5, v_6, b_1$. La base es b_1 pues es el predecesor común más cercano de v_{10} y v_5.

La construcción de la gráfica $G'' = G'/B_2$ se observa en la Figura B.5. Observamos que b_2 es el vértice nuevo que representa a todos los vértices del *blossom* B_2 y además b_2 es etiquetado como par. Partiendo de b_2, examinamos la arista b_2v_3 con lo cual etiquetamos a v_3 como impar y por tanto a v_4 como par.

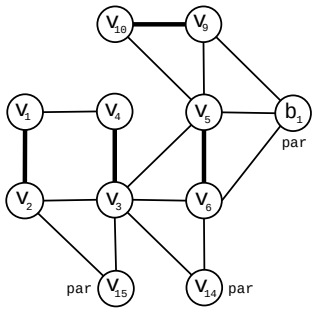

Figura B.4: Gráfica resultante de la primera reducción

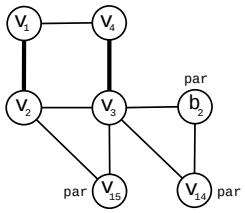

Figura B.5: Gráfica resultante de la segunda reducción

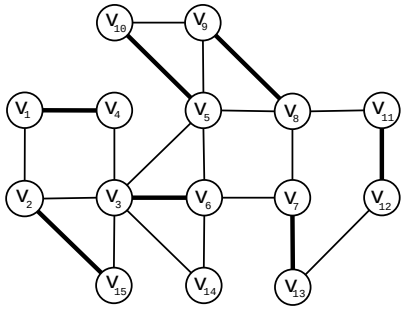

Figura B.6: Apareamiento actualizado en G

Continuamos con éste último, etiquetamos a v_1 como impar y luego a v_2 como par. Finalmente examinamos la arista v_2v_{15} y encontramos la trayectoria aumentante $P'' : b_2, v_3, v_4, v_1, v_2, v_{15}$ pues ambos extremos están en árboles alternantes distintos. Una vez obtenida P'', podemos construir a partir de ella la trayectoria aumentante $P' : b_1, v_9, v_{10}, v_5, v_6, v_3, v_4, v_1, v_2, v_{15}$ en G' y luego, obtenemos la trayectoria aumentante $P : v_{13}, v_7, v_8, v_9, v_{10}, v_5, v_6, v_3, v_4, v_1, v_2, v_{15}$ en G y obtenemos el apareamiento máximo ilustrado en la Figura B.6.

Los inconvenientes de este algoritmo son principalmente la representación de gráficas reducidas y la recuperación de las trayectorias aumentantes en la gráfica original. Sin embargo, se pueden emplear estructuras de datos de conjuntos ajenos para ello y obtener una implementación que permite resolver el problema de apareamiento en gráficas generales con un tiempo de ejecución $O(nm\alpha(m,n))$ en el peor caso, donde n es el número de vértices y m es el número de aristas. La implementación que revisaremos a continuación está basada en [16, 19].

B.3 Implementación

Recordamos que el algoritmo construye un bosque de árboles alternantes enraizados en vértices que no están apareados. En la implementación haremos esto mediante una serie de etiquetas sobre los vértices de la gráfica original, la cual previamente transformamos en una gráfica dirigida de modo que cada arista vw se represente con dos arcos (v,w) y (w,v).

Una de las etiquetas empleadas es *state*, la cual establece el estado actual de un vértice v con respecto al bosque construído: aquellos vértices que no son apareados con respecto al apareamiento actual se etiquetan inicialmente como vértices pares, mientras que a los demás se les asigna una etiqueta que indica que no han sido agregados al bosque. Estos vértices eventualmente serán etiquetados como pares o impares según sea el caso. De esta forma, $state(v)$ puede tomar tres valores: *even* si v es par, *odd* si v es impar o *unreached* si v no ha sido agregado al bosque.

Otra etiqueta empleada para cada vértice v es $mate(v)$, que corresponde al vértice w tal que vw es una arista del apareamiento. Si existe tal vértice w, entonces se marca con una etiqueta $matched(v) = true$, en otro caso, se tendrá $matched(v) = false$, y $mate(v)$ quedará sin definir.

Para el recorrido de los árboles alternantes, se utiliza una etiqueta $parent(v)$, que indica cuál es el predecesor directo de v en el árbol alternante donde éste se encuentre. Esta etiqueta no se modifica cuando un vértice es reducido en otro.

Para la representación implícita de un *blossom* utilizamos conjuntos ajenos con unión por rango y compresión de trayectorias. Cada uno de los conjuntos corresponde a un *blossom* encontrado en la gráfica por el algoritmo.

Como sabemos, cada conjunto cuenta con un elemento canónico que representa a todos sus elementos y lo podemos obtener con la operación find. Sin embargo, lo que se busca es fijar a uno de los vértices del conjunto como la base del *blossom*, que no necesariamente tiene que ser el elemento canónico. Es por esto que recurrimos a otra etiqueta denominada $origin(v)$ para cada vértice. Así, podemos saber cuál es la base del *blossom* que contiene a v mediante la operación $origin(\mathsf{find}(v))$.

Sea G la gráfica original. Supongamos que G' es la gráfica actual obtenida a partir de G por medio de uno o varios procesos de reducción. La relación entre ambas puede expresarse de la siguiente manera:

Para cada $v \in V(G)$:

- si $origin(\mathsf{find}(v)) = v$ entonces $v \in V(G')$.
- si $origin(\mathsf{find}(v)) \neq v$, entonces v ha sido compactado en el vértice $origin(\mathsf{find}(v))$.

Por tanto, la gráfica actual G' se define como sigue:

$V(G') = \{origin(\mathsf{find}(v)) \mid v \in V(G)\}$

$E(G') = \{v'w' \mid v' = origin(\mathsf{find}(v)), w' = origin(\mathsf{find}(w))\}$
donde $v, w \in V(G)$.

Usaremos una etiqueta *visited* para los vértices pares que ya han sido agregados al bosque, para el caso en que sea revisada una arista con dos vértices pares u y v, pues puede tratarse de una trayectoria aumentante o un *blossom*. De esta forma, u y v tienen un antecesor común si y sólo si a lo largo de la trayectoria que termina en u hay un vértice ya visitado de la trayectoria que termina en v o viceversa. Es necesario determinar en cual de estas dos trayectorias aparece cada vértice que se revisa. Este proceso se hace con un recorrido sobre la gráfica actual usando las etiquetas *parent*.

Cuando una arista vw cierra un *blossom*, todos los vértices impares contenidos en él tienen la propiedad de que existe una trayectoria alternante de longitud par desde la raíz de su árbol alternante hacia ellos. Tales trayectorias incluyen la arista vw a la cual llamamos puente. Por tanto, para los vértices impares ubicados en la trayectoria en el árbol de v a la base del *blossom*, tenemos que indicar que (v, w) es el arco que representa la dirección que sigue el puente, mientras que para los que aparecen en la trayectoria en el árbol de w a la base del *blossom*, se indica que la dirección que sigue el puente está representada por el arco (w, v). La dirección que sigue la arista puente para cada vértice del *blossom* la identificaremos con la etiqueta *bridge*. De esta forma, podremos transformar una trayectoria aumentante en la gráfica actual a una en la gráfica original, sin la necesidad de conocer las gráficas intermedias que se usaron en las reducciones efectuadas.

Ahora detallaremos el algoritmo de Edmonds con las ideas dadas sobre la implementación. Asumimos que $v' = origin(\mathsf{find}(v))$ y $w' = origin(\mathsf{find}(w))$. Utilizaremos una cola Q para almacenar los vértices que se vayan etiquetando, los cuales en general son pares, salvo en el caso 3(a) que aquí explicaremos.

También definiremos la forma en la que se reconstruye una trayectoria aumentante en la gráfica original.

Sea $(v, w) \in E(G)$ no examinada tal que $state(v') = even$. Tenemos varios casos nuevamente:

1. Si $v' = w'$ no hacemos nada. Lo mismo ocurre si $state(w') = odd$.
2. Si $state(w') = unreached$, lo cual equivale a que $state(w) = unreached$, entonces podemos extender el árbol alternante definiendo las etiquetas como sigue: $state(w) = odd$, $parent(w) = v$, $state(mate(w)) = even$ y $parent(mate(w)) = w$. Agregamos finalmente $mate(w)$ a Q.
3. Si $state(w') = even$ entonces hay otros dos casos:
   (a) si v' y w' están en el mismo árbol alternante, entonces hemos encontrado un *blossom* cuya base b es el antecesor común más cercano de ambos y además, es par. Por cada vértice x dentro del *blossom* definimos $origin(\mathsf{find}(x)) = b$ y unimos los conjuntos ajenos que contienen a b y a x. Con esto aseguramos que para cualquier vértice x que esté en el *blossom* y sea elemento canónico, se tenga la información de que b es el vértice en el cual fue compactado. Si x es un vértice impar definimos $bridge(x) = (v, w)$ si está en la trayectoria de b a v y en cambio, ponemos $bridge(x) = (w, v)$ si está en la trayectoria de b a w. Este es el único caso donde agregamos vértices impares a Q, puesto que los vértices adyacentes a ellos en G, ahora son adyacentes a b en G'. Se repite el proceso.
   (b) si v' y w' están en distintos árboles alternantes, entonces hemos encontrado una trayectoria aumentante. Sea r_v la raíz del árbol que contiene a v' y r_w la del árbol que contiene a w'. Decíamos en la sección anterior que si unimos las trayectorias $[r_v, \ldots, v]$ y $[w, \ldots, r_w]$, tenemos una trayectoria aumentante. El detalle es que la primera de estas dos trayectorias no puede ser recuperada directamente con las etiquetas *parent*, pues éstas van en sentido inverso, por tanto necesitamos invertir el orden de los vértices que se obtienen de ella. Ahora definimos inductivamente un procedimiento que recupera la trayectoria aumentante entre dos vértices $x, y \in V(G)$:
      - Si $x = y$, la trayectoria consiste únicamente del vértice x.
      - Si $x \neq y$ y $state(x) = even$ entonces consideramos la trayectoria alternante $P' : parent(mate(x)), \ldots, y$. Por tanto, la trayectoria aumentante buscada consiste de x, $mate(x)$ y los vértices de P'.
      - Si $x \neq y$ y $state(x) = odd$ entonces recurrimos a la etiqueta $bridge(x) = (a, b)$. Consideramos las trayectorias alternantes $P_1 : b, \ldots, y$ y $P_2 : a, \ldots, mate(x)$. Por tanto la trayectoria aumentante buscada consiste de x, los vértices de P_2 en orden inverso y los vértices de P_1. Se repite el proceso con el apareamiento nuevo.

Apéndice C

Conjuntos Ajenos

Algunas aplicaciones involucran el agrupamiento de n objetos distintos en una colección de conjuntos ajenos. Dos operaciones fundamentales son determinar el conjunto al cual pertenece un elemento dado y unir dos conjuntos. En este capítulo revisamos métodos para el mantenimiento de una estructura de datos que soporte esas operaciones.

C.1 Especificación

En esta sección presentamos el tipo de datos abstracto para conjuntos ajenos, que como su nombre lo indica, mantiene una colección de conjuntos ajenos y dinámicos, $S = \{S_1, S_2, \ldots, S_k\}$. Cada conjunto S_i es identificado por un representante, también denominado **elemento canónico** del conjunto S_i.

Las operaciones especificadas para la manipulación de conjuntos ajenos son:

- makeSet(x). Crea un nuevo conjunto cuyo único miembro es el elemento x e incorpora el conjunto a la colección.
- union(x,y). Construye un nuevo conjunto que es la unión de los dos conjuntos cuyos elementos canónicos son x y y. Destruye los conjuntos antiguos y determina un representante para el nuevo conjunto. Se asume que $x \neq y$.
- find(x). Regresa el elemento canónico del conjunto que contiene al elemento x.

A continuación ilustramos con un ejemplo la manipulación de una colección de conjuntos ajenos con estas operaciones. El elemento canónico de cada conjunto es el que aparece en primer lugar del conjunto correspondiente.

Ejemplo.

S.makeSet(a) $\longrightarrow S = \{\{a\}\}$

$\mathsf{S.makeSet}(b) \longrightarrow S = \big\{\{a\},\{b\}\big\}$
$\mathsf{S.makeSet}(c) \longrightarrow S = \big\{\{a\},\{b\},\{c\}\big\}$
$\mathsf{S.makeSet}(d) \longrightarrow S = \big\{\{a\},\{b\},\{c\},\{d\}\big\}$
$\mathsf{S.makeSet}(e) \longrightarrow S = \big\{\{a\},\{b\},\{c\},\{d\},\{e\}\big\}$
$\mathsf{S.union}(a,b) \longrightarrow S = \big\{\{a,b\},\{c\},\{d\},\{e\}\big\}$
$\mathsf{S.union}(c,e) \longrightarrow S = \big\{\{a,b\},\{c,e\},\{d\}\big\}$
$\mathsf{S.find}(e) \longrightarrow c$

Para el análisis de los tiempos de ejecución de las operaciones definidas denotamos por n al número de elementos agrupados en la colección, es decir, el número de operaciones makeSet y por m al número total de operaciones makeSet, union y find. Como el número de operaciones makeSet está incluido en el número total de operaciones m, se tiene que $m \geq n$. Como los conjuntos son ajenos, cada vez que se ejecuta la operación find se reduce el número de conjuntos en una unidad, por lo cual, es posible ejecutar a lo más $n-1$ veces la operación find, ya que después de ello queda solamente un conjunto en la colección.

C.2 Representación Galler-Fischer

La estructura de datos propuesta por Galler y Fischer para el mantenimiento de conjuntos ajenos representa cada conjunto como un árbol enraizado cuyos nodos son los elementos del conjunto. El elemento canónico es la raíz del árbol. Cada nodo x tiene un apuntador $p(x)$ a su padre en el árbol y la raíz del árbol apunta a sí misma. Con esto las operaciones quedan determinadas de la siguiente forma:

- makeSet(x). Se define $p(x) = x$.
- union(x,y). Se define $p(y) = x$, es decir, el elemento canónico del nuevo conjunto es x.
- find(x). Se siguen los apuntadores padre desde x hasta el nodo raíz del árbol que lo contenga y se regresa la raíz.

Con esta representación y la forma en como se definieron las operaciones, resulta que los tiempos de ejecución de makeSet y union son constantes. Sin embargo, la definición de la operación union, denominada *unión simple*, provoca que el tiempo de ejecución de una operación find sea $O(n)$, pues puede ocurrir en algún momento que los n nodos se encuentren en el mismo árbol, formando una trayectoria dirigida.

Para mejorar los tiempos de ejecución de las operaciones con la estructura propuesta por Galler y Fischer, se manejan diversas estrategias; unas consideran ciertos criterios para la unión de los conjuntos y otras comprimen las trayectorias de búsqueda del elemento canónico de un conjunto, una vez que han sido determinadas.

C.3 Estrategias para mejorar el tiempo de ejecución

Compresión de trayectorias. Consiste en cambiar la estructura de los árboles cada vez que se lleva a cabo una operación find, moviendo algunos nodos más cerca de la raíz. Cuando se ejecuta la operación find(x), luego de identificar la trayectoria entre x y la raíz r del árbol que contiene a x, se obliga a que cada nodo de la trayectoria de x a r apunte directamente a r. De esta forma, la operación find recorre dos veces dicha trayectoria.

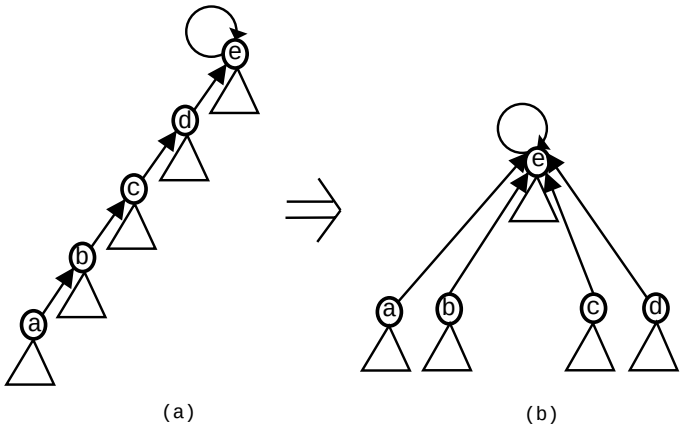

Figura C.1: Compresión de trayectorias

La Figura C.1 muestra un ejemplo del proceso de compresión de trayectorias después de ejecutar la operación find(a). En la parte (a) se muestra la trayectoria [a,b, c, d, e] y en la parte (b) se observa el efecto de su compresión. Los triángulos representan subárboles.

Unión por rango. Se mantiene para cada nodo x un entero no negativo llamado rango de x y denotado por $r(x)$, que representa una cota superior sobre la altura de x en el árbol donde se encuentra. La altura de x es el número de aristas en la trayectoria más larga entre x y cualquier nodo que no tenga descendientes.

Cuando se ejecuta una operación makeSet(x), se define $r(x) = 0$. Para unir dos conjuntos cuyos elementos canónicos son x y y se comparan los rangos de estos elementos y se procede como sigue:

- Si $r(x) > r(y)$ entonces se define $p(y) = x$
- Si $r(x) < r(y)$ entonces se define $p(x) = y$
- Si $r(x) = r(y)$ entonces se define $p(x) = y$ y se incrementa $r(y)$ en una unidad.

La Figura C.2(a) ilustra el proceso de unión cuando se tienen elementos canónicos con diferente rango: el nodo con rango r apunta al nodo con rango s; mientras que la Figura C.2(b) muestra la unión cuando se tienen elementos con el mismo rango: además de que el nodo con rango r apunta el nodo con rango s, éste toma un rango de valor $s + 1$.

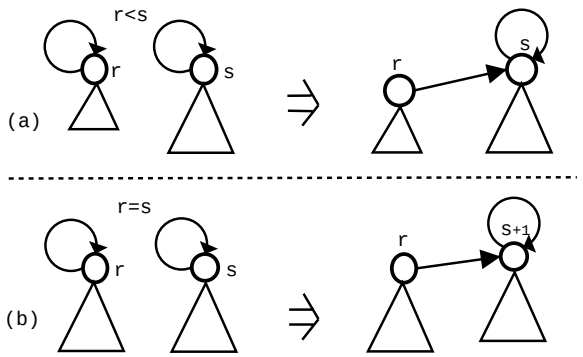

Figura C.2: Unión por rango

En el Listado 21 se presenta un pseudocódigo que recupera las ideas expuestas sobre la representación de conjuntos ajenos con unión por rango y compresión de trayectorias. Cada nodo x tiene una etiqueta $r(x)$ que representa el rango de x y una etiqueta $p(x)$ que representa el apuntador al padre de x en el árbol donde se encuentre. En el método que representa la operación find, la variable c_e representa el elemento canónico del conjunto de x.

Cada estrategia por separado mejora el tiempo de ejecución del total de operaciones realizadas. Hay una mejora más significativa si se aplican ambas estrategias. En este material no incluimos las demostraciones sobre ello, pero resulta que si usamos únicamente unión por rango, el tiempo de ejecución en el peor caso es $O(m \log n)$. Si empleamos nada más la compresión de trayectorias, necesitaremos conocer el número k de operaciones find que se ejecuten, pues mientras más grande sea k, habrá más compresiones y las operaciones subsecuentes se ejecutarán más rápido. Si $k < n$, se tiene un tiempo de ejecución en el peor caso de $\Theta(n + k \log n)$, pero si $k \geq n$, entonces el tiempo de ejecución es $\Theta(k \log_{(1+k/n)} n)$.

Cuando se aplican ambas estrategias, el peor caso es $O(m\alpha(m, n))$, donde $\alpha(m, n)$ es la inversa de la función de Ackermann, que definimos a continuación. Cabe señalar que para todos los propósitos prácticos donde se involucren los conjuntos ajenos, $\alpha(m, n) \leq 4$, por lo que se puede ver el tiempo total de ejecución como si fuera lineal en el número total de operaciones m, aunque $\alpha(m, n)$ no sea constante.

Listado 21 Conjuntos ajenos con unión por rango y compresión de trayectorias

```
procedure makeSet(x);
1: p(x) = x;
2: r(x) = 0;
procedure union(x,y);
1: if r(x) > r(y) then
2:    p(y) = x;
3: else
4:    p(x) = y;
5:    if r(x) = r(y) then
6:       r(y) = r(y) + 1;
7:    end if
8: end if
procedure find(x);
1: if p(x) = x then
2:    c_e = x;
3: else
4:    p(x) = find(p(x));
5:    c_e = p(x);
6: end if
```

C.4 La función de Ackermann y su inversa

Sean i y j dos números naturales. La función de Ackermann se define como sigue:

$$\begin{array}{ll} A(1,j) = 2^j & \text{para } j \geq 1, \\ A(i,1) = A(i-1,2) & \text{para } i \geq 2, \\ A(i,j) = A(i-1, A(i,j-1)) & \text{para } i,j \geq 2. \end{array}$$

Esta función tiene la propiedad de que crece rápidamente. De hecho es estrictamente creciente con cada argumento. En cambio su inversa es una función que crece lentamente.

Se define la función inversa de la función de Ackermann de la siguiente forma para $m \geq n \geq 1$,

$$\alpha(m,n) = min\{i \geq 1 \mid A(i, \lfloor m/n \rfloor) > \log n\}.$$

Para valores fijos de n, conforme el valor de m crece, $\lfloor m/n \rfloor$ toma valores más grandes mientras que la función $\alpha(m,n)$ toma valores cada vez menores. Esto va de acuerdo con lo que intuitivamente se mencionaba, ya que para un número dado n de elementos, al aumentar el número de operaciones, esperaríamos que las trayectorias de cada nodo a sus respectivas raíces tuvieran menor longitud

debido a la compresión de trayectorias.

Observamos que $\lfloor m/n \rfloor \geq 1$, pues $m \geq n$, por tanto $A(i, \lfloor m/n \rfloor) \geq A(i, 1)$ $\forall i \geq 1$.

En particular, $A(4, \lfloor m/n \rfloor) \geq A(4, 1) = A(3, 2) = A(2, 16)$, lo cual es mayor que el número estimado de átomos en el universo, aproximadamente 10^{80}. Sólo para valores muy grandes e imprácticos de n se tendría que $A(4, 1) \leq \log n$, por tanto, podemos decir que $\alpha(m, n) \leq 4$ para todos los propósitos prácticos.

Para mayores detalles teóricos sobre el análisis del tiempo de ejecución de los métodos revisados para conjuntos ajenos y la función de Ackermann, sugerimos la consulta del material presentado en [8, 20].

Índice de Figuras

Índice de Listados

Bibliografía

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison Wesley, 1983.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[4] S. Baase. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, second edition, 1988.

[5] N. Blum. Maximum matching in nonbipartite graphs without explicit consideration of blossoms. Research report, Universität Bonn, 1999.

[6] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, New York, 1976.

[7] G. Chartrand and O. R. Oellermann. *Applied and Algorithmic Graph Theory*. International Series in Pure and Applied Mathematics. McGraw-Hill, 1993.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill, 1990.

[9] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17, 1965.

[10] J. R. Evans and E. Minieka. *Optimization Algorithms for Networks and Graphs*. Marcel Dekker, second edition, 1992.

[11] H. N. Gabow. An efficient implementation of edmonds algorithm for maximum matching on graphs. *Journal of the ACM*, 23, 1976.

[12] M. T. Goodrich and R. Tamassia. *Data Structures and Algorithms in Java*. John Wiley & Sons, Inc., second edition, 2001.

[13] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer Verlag, 1999.

[14] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[15] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989.

[16] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[17] R. Möhring and M. Müller-Hannemann. Cardinality matching: Heuristic search for augmenting paths. Technical report, Technische Universität Berlin, 1995.

[18] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[19] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[20] R. E. Tarjan and J. van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2), 1984.

[21] M. A. Weiss. *Data Structures and Problem Solving Using Java*. Addison Wesley, 1998.

[22] M. Gasca Soto y F. Juárez Almaraz. *El análisis amortizado y sus aplicaciones*. Notas de Clase. Publicaciones del Departamento de Matemáticas, Facultad de ciencias, UNAM, 2004.