Accoppiamento in grafi bipartiti e non

Lorenzo Cioni

Sommario

Il presente lavoro contiene un esame di alcuni algoritmi per la risoluzione di *problemi di accoppiamento* in grafi bipartiti e non. L'articolo si apre con alcuni esempi sia di grafi bipartiti sia di grafi non bipartiti, in modo da inquadrare il problema nei suoi vari aspetti, vengono poi date alcune definizioni e viene introdotta la notazione utilizzata nel testo per poi passare alla presentazione dei vari algoritmi, alcuni scritti in *pseudocodice*, e alla loro discussione.

1 Introduzione

Con il termine *accoppiamento* si definisce una relazione $\Re$ binaria simmetrica fra elementi di un insieme $\mathcal{A}$ (con $|\mathcal{A}| = n,\ n \in \mathcal{N}$) caratterizzata dal fatto che le coppie di elementi in relazione $\Re$ fra di loro sono coppie distinte ovvero vale la proprietà seguente:

$$dati\ a,\ b,\ c,\ d\ \in \mathcal{A} \mid a\Re b\ \&\ c\Re d\ \Rightarrow a \neq c,\ a \neq d,\ b \neq c,\ b \neq d \tag{1}$$

Se la relazione $\Re$ definisce un insieme di coppie $\mathcal{C}$ tale che $|\mathcal{C}| = \frac{n}{2}$ (e la cosa è possibile se e solo se n è pari) si parla di *accoppiamento perfetto* o *assegnamento*.
Nel caso in esame l'insieme $\mathcal{A}$ è l'insieme N dei nodi di un grafo $G = (N, A)$ non orientato. Verranno esaminati due casi.
Il primo è quello in cui l'insieme dei nodi N può essere partizionato in due sottoinsiemi O e D (tali che $O \cup D = N e O \cap D = \emptyset$) per cui un accoppiamento (eventualmente perfetto) coinvolge nodi $i \in O$ e nodi $j \in D$. In questo caso il grafo è detto *bipartito*.
Il secondo è quello in cui ciascun nodo $i \in N$ può essere accoppiato con ogni altro nodo $j \in N,\ i \neq j$. In questo caso il grafo verrà detto *non bipartito*.

2 Esempi

2.1 Grafi bipartiti

Un esempio di *grafo bipartito* è illustrato dalla figura 1 ([Ore76]). Il grafo

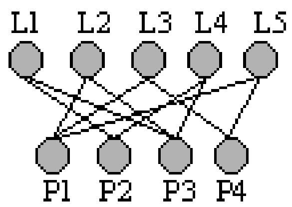

Figura 1: *Esempio di grafo bipartito*

di figura 1, $G = (N, A)$, è caratterizzato da un partizionamento dell'insieme N in due sottoinsiemi L (lavori assegnabili) e P (persone disponibili) e rappresenta la situazione: "quali persone sono qualificate per quali lavori".
In generale, un grafo G è detto essere bipartito se gli elementi associati ai nodi

non sono omogenei e possono essere suddivisi in due categorie distinte in modo che i corrispondenti nodi possano essere partizionati in due sottoinsiemi disgiunti. Dato che in figura 1 si ha:

$$|L| = 5 \tag{2}$$

$$|P| = 4 \tag{3}$$

l'accoppiamento non può essere perfetto. Condizione necessaria perchè un accoppiamento sia perfetto è che sia $|O| = |D|$.
Come sarà meglio chiarito in seguito, se con M si definisce un accoppiamento fra elementi di L e elementi di P si ha:

$$|M|_{max} = min\{|P|, |L|\} \tag{4}$$

Sempre con riferimento alla figura 1, dato un elemento $p \in P$ ed un elemento $l \in L$, se il grafo contiene un arco (p, l) vuol dire che la persona p è qualificata per il lavoro l e, viceversa, che il lavoro l può essere svolto dalla persona p.
In generale il grafo G non è completo (ovvero $|A| < \frac{|N| \times (|N| - 1)}{2}$) per cui non sempre è possibile stabilire un accoppiamento fra un elemento $p \in P$ ed uno $l \in L$. Una condizione necessaria e sufficiente perchè sia possibile accoppiare gli elementi di P ciascuno con un elemento di L è che risulti verificata la seguente *condizione di diversità* ([Ore76]):

> dato un gruppo di k persone, $k \in [1, \ldots, n]$, dove $n = |P|$, ci devono essere almeno k lavori per i quali le k persone nell'insieme sono qualificate.

Abbiamo visto cosa significa un *accoppiamento* fra due insiemi P ed L: si parla di accoppiamento quando comunque si prenda un elemento $p \in P$ esiste al più un arco (p, l) incidente in un elemento distinto $l \in L$. La figura 2 contiene due esempi di accoppiamento relativi al grafo bipartito della figura 1. Se gli insiemi P ed L sono tali che $|P| = |L|$ ed è soddisfatta la *condizione*

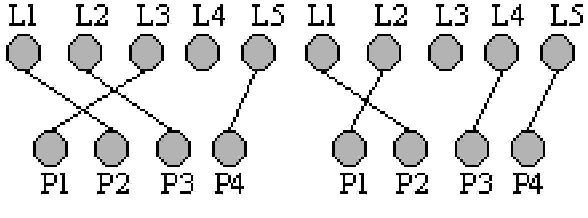

Figura 2: *Esempi di accoppiamenti in grafo bipartito*

di diversità ([Ore76]) allora è possibile stabilire una corrispondenza biunivoca fra elementi di P ed elementi di L e, in questo caso, si parla di *accoppiamento perfetto* o di *assegnamento* o anche di *problema dei matrimoni* (cfr. la figura 3).

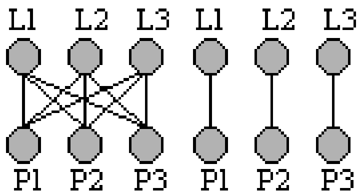

Figura 3: *Esempio di grafo bipartito completo con uno dei possibili assegnamenti*

2.2 Grafi non bipartiti

Esempi di situazioni descrivibili mediante grafi $G = (N, A)$ non bipartiti sono tutti quelli in cui gli elementi associati ai nodi sono omogenei. Due esempi sono i seguenti (cfr. la figura 4, [Ore76]):

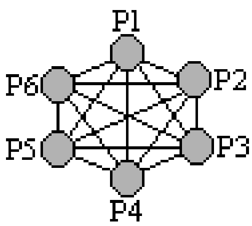

Figura 4: *Esempio di grafo non bipartito completo*

1. torneo circolare con n partecipanti,

2. creazione di coppie di piloti in un insieme di n piloti.

Nel primo caso il grafo descrittivo della situazione è:

- non bipartito,

- completo, dato che ogni giocatore deve giocare con tutti gli altri, mentre nel secondo caso è

- non bipartito,
- in genere non completo, dato che non tutti i piloti possono, in genere, formare una coppia.

Nel caso 1 ([Ore76]), dati $n = |N|$ partecipanti ciascuno di essi si deve incontrare con i rimanenti $n-1$. Se n è dispari si può introdurre un giocatore fittizio f in modo che le coppie (i, f) con $i \neq f$ possono essere interpretate come un turno di riposo per il giocatore i. Nel seguito, pertanto, supporremo, per il caso 1, che n sia pari.
Nel caso di un torneo circolare, il grafo è completo e un modo per ricavare gli $n-1$ accoppiamenti ciascuno di cardinalità $\frac{n}{2}$ è quello di determinare la cosiddetta *tabella degli accoppiamenti*.
La tabella degli accoppiamenti (cfr. la figura 5) è una matrice quadrata $n \times n$ in cui la prima riga e la prima colonna contengono i numeri da 1 a n mentre i rimanenti $(n-1) \times (n-1)$ sono costruiti con le regole seguenti:

1. le $\frac{n}{2}$ righe successive alla prima iniziano (a colonna 2) ciascuna con un numero pari crescente da 2 a n seguito dai numeri decrescenti fino a 2 e poi di nuovo decrescenti da n al valore iniziale escluso (tale ordinamento è detto ordine circolare decrescente),
2. le altre $\frac{n}{2}-1$ righe sono costruite in modo analogo solo che a colonna 2 hanno numeri dispari crescenti da 3 a (n-1) ciascuno seguito dagli altri numeri in ordine circolare decrescente,
3. come ultimo passo gli elementi della diagonale principale sono posti a 1 ciò perchè ciascun giocatore non gioca mai contro se stesso.

1	2	3	4	5	6
2	1	6	5	4	3
3	4	1	2	6	5
4	6	5	1	3	2
5	3	2	6	1	4
6	5	4	3	2	1

Figura 5: *Tabella dei turni per il grafo di figura 4*

Una volta costriuta la tabella, ognuna delle righe fornisce gli accoppiamenti fra gli n giocatori ad ogni turno per gli n turni. Nella tabella l'elemento della

riga i e della colonna j indica il giocatore avversario del giocatore j al turno i.
La tabella gode delle seguenti proprietà ([Ore76]):

1. se i giocatori sono n si hanno n turni e in ciascun turno si hanno $\frac{n}{2}$ accoppiamenti,

2. ogni giocatore incontra una ed una sola volta tutti gli altri giocatori,

3. ad ogni turno ogni giocatore incontra un giocatore diverso,

4. se ad un certo turno il giocatore k incontra il giocatore j allora dalla tabella risulta anche che il giocatore j incontra il giocatore k.

Nel caso 2 ([AMRJ96] si presentano in genere due tipi di problemi:

1. di *cardinalità massima*,

2. di *costo minimo*.

Nel primo caso si tratta di determinare il massimo numero possibile di coppie di piloti mentre nel secondo caso si tratta di formare un dato numero di equipaggi con in più il vincolo di minimizzare il costo totale di un insieme di coppie (a ciascuna delle quali viene attribuito un costo che tiene conto della maggiore o minore compatibilità fra i due piloti): in quest'ultimo caso il grafo è completo (o può essere sempre esteso in modo che lo sia) e pesato. Si fa notare che, una volta che sia stato definito un insieme di coppie di piloti, l'associazione fra le coppie dell'insieme e l'insieme degli aereomobili è di nuovo un problema di accoppiamento su un grafo bipartito.
Vedremo nel seguito come sia possibile affronatre e risolvere tali problemi.

3 Definizioni, notazione e algoritmi

Prima di presentare gli algoritmi che consentono di determinare accoppiamenti che soddisfano certi requisiti di costo e/o cardinalità in grafi bipartiti e non, sulla base di [GPS$^+$02] e [AMRJ96], si introducono alcune definizioni e si precisa la notazione che sarà usata nelle successive sezioni.

3.1 Definizioni e notazione per grafi bipartiti (e non)

Dato un grafo non orientato $G = (N, A)$ lo si dice *bipartito* se i suoi nodi possono essere pensati raggruppati in due categorie distinte in modo da suddividere l'insieme N in due insiemi disgiunti che diremo O (nodi origine, $|O| = n$) e D (nodi destinazione, $|D| = d$) [1] in modo che si possa scrivere $G = (O \cup D, A)$ dove $A \subseteq O \times D$ è l'insieme degli archi[2] del grafo, $|A| = m$.
Con il termine 'accoppiamento' M su un grafo G non orientato si definisce un insieme di spigoli non adiacenti ovvero privi di nodi in comune.
Dato il grafo $G = (N, A)$ per caratterizzare un accoppiamento si danno alcune definizioni a livello dei nodi e degli archi.
Per i nodi si usano le dizioni di *nodo esposto* per indicare un nodo in cui non incide nessun arco di un accoppiamento e di *nodo accoppiato* per indicare un nodo in cui incide un arco di un accoppiamento.
Gli archi a loro volta sono detti *interni* se appartengono ad un accoppiamento M altrimenti sono detti *esterni*.
Tali definizioni consentono di partizionare l'insieme N in due sottoinsiemi contenenti, rispettivamente, i *nodi esposti* e i *nodi accoppiati* e l'insieme degli archi A in due sottoinsiemi di archi accoppiati o interni $A_I = M$ e di archi liberi o esterni $A_E = A \setminus M$.
Dato che gli algoritmi che saranno descritti fanno uso di concetti quali *cammino alternante* e *cammino aumentante* si danno le seguenti definizioni.
Dato un grafo G, un cammino P è, per definizione, una successione di archi a due a due adiacenti che uniscono un nodo origine i_0 ad un nodo destinazione i_n.
Un ciclo è un cammino in cui il nodo iniziale e quello finale coincidono.
Un cammino (detto anche *catena*) è detto essere *semplice* se non non con-

[1] I due insiemi O e D sono una partizione di N nel senso che valgono le seguenti equazioni:

$$N = O \cup D \tag{5}$$

$$O \cap D = \emptyset \tag{6}$$

[2] Detti anche spigoli, essendo il grafo non orientato.

tiene cicli come sottocammini propri.
Un cammino non semplice è un cammino in cui si ha ripetizione di nodi con o senza ripetizione di archi
Un cammino, infine, è detto essere elementare se sono distinti anche i vertici che lo compongono.
Un *cammino alternante* è una catena elementare di elementi di G i cui elementi appartengono alternativamente a $A \setminus M$ e ad M, un *cammino aumentante* è un cammino alternante i cui nodi iniziale e finale sono esposti.
Dato un cammino aumentante P rispetto ad un accoppiamento M, si ha:

1. $|P| = 2 \times k + 1, \ k \in \mathcal{N}$,
2. gli spigoli di P in posizione dispari sono liberi (o esterni) mentre quelli in posizione pari sono accoppiati (o interni),
3. un modo per determinare un nuovo accoppiamento M' è quello di applicare la seguente operazione di composizione:

$$M' = M \setminus P \cup P \setminus M \tag{7}$$

   cioè si tolgono da M gli archi accoppiati di P e si aggiungono quelli liberi. Una formulazione alternativa è la seguente:

$$M' = M \setminus P_I \cup P_E \tag{8}$$

in cui P_I è l'insieme degli archi di P che sono in M mentre P_E è l'insieme degli archi di P che non sono in M per cui $P = P_E \cup P_I$ e $P_E \cap P_I = \emptyset$.
Dato un cammino aumentante P per costruzione si può verificare che vale la relazione seguente:

$$|P_E| = |P_I| + 1 \tag{9}$$

mentre dato un cammino aumentante P rispetto ad un accoppiamento M si ha che il nuovo accoppiamento ottenuto mediante la (8) gode della proprietà seguente:

$$|M'| = |M| - |P_I| + |P_E| = |M| + 1 \tag{10}$$

Al proposito vale il seguente Teorema

Teorema di Berge

Un accoppiamento M su un grafo G è di cardinalità massima se e solo se non ci sono in G accoppiamenti di cardinalità maggiore.

Dimostrazione

La condizione è necessaria e sufficiente per cui la dimostrazione la si fa in due passi. Dimostro il teorema dimostrando che se esiste un cammino aumentante P per M allora M non è un accoppiamento di massima cardinalità e che se M è un accoppiamento di massima cardinalità allora non esistono cammini aumentanti per M.

Condizione Necessaria

Dato un accoppiamento M di massima cardinalità, se per assurdo esistesse un cammino aumentante P per M allora potrei creare un accoppiamento M' usando la (7) per cui M non sarebbe un accoppiamento di massima cardinalità contro l'ipotesi.

Condizione Sufficiente

Supponiamo di avere un accoppiamento M rispetto a cui non si hanno cammini aumentanti e sia M^* un accoppiamento tale che:

$$|M^*| = |M| + 1 \tag{11}$$

Se calcolo la differenza simmetrica fra i due accoppiamenti come:

$$\Delta = M \setminus M^* \cup M^* \setminus M \tag{12}$$

ho che Δ contiene un numero dispari di spigoli e, inoltre, contiene un numero di spigoli di M^* maggiore del numero di spigoli di M (esattamente uno in più). In più, il sottografo indotto da Δ ha nodi di grado ≤ 2 (gli archi di Δ appartengono o a M o a M^* ma non ad entrambi per cui su ciascuno dei nodi del grafo indotto possono incidere al più due archi, uno per ciascun accoppiamento). Le componenti connesse del grafo indotto G_Δ sono pertanto:

1. nodi isolati, non ci interessano,
2. cicli con un numero pari di spigoli, tanti di M e tanti di M^*,
3. cammini

Poichè Δ ha cardinalità dispari esiste almeno un cammino con un numero dispari di elementi con un arco di M^* in più (per costruzione di M^*) per cui tale cammino (che inizia e termina con un arco in M^*) è un cammino aumentante per M contro l'ipotesi che M sia un accoppiamento rispetto a cui non esistono cammini aumentanti.

3.2 Problemi di cardinalità massima per grafi bipartiti

Dato un grafo non orientato ([AMRJ96]) $G = (O \cup D, A)$ con $|O| = n$ e $|D| = m$ un *accoppiamento* M di O in D si dice completo se $\forall i \in O\, \exists (i, j) \in M$.
Un accoppiamento completo di O in D si dice una *copertura* per O.
Se $m = n$ una copertura per O è una copertura anche per D. Se $m > n$ una copertura per O lascia $m - n$ nodi di D esposti. Il viceversa accade per $m < n$ considerando un accoppiamento di D in O.
Per poter avere un accoppiamento completo di O in D deve essere $|O| \leq |D|$. Definendo con W un sottoinsieme di O e con $G(W)$ l'insieme dei vertici di D adiacenti ad almeno uno dei vertici di W si ha il seguente Teorema.

Teorema di Hall
Dato un grafo bipartito $G = (O \cup D, A)$ esiste un accoppiamento completo di O in D se e solo se vale la seguente relazione:

$$\forall W \subseteq O\ |W| \leq |G(W)| \tag{13}$$

Dimostrazione
La condizione è necesaria e sufficiente per cui la dimostrazione la si fa in due passi.

Condizione necessaria
Se esiste un sottoinsieme $W \subseteq O$ tale che $|W| \geq |G(W)|$ non si può avere un arco incidente che collega vertici distinti di W a vertici distinti di $G(W)$ ovvero di D.

Condizione sufficiente
Si lavora per induzione su $m = |O|$ sfruttando la condizione necessaria prima dimostrata.
Se $m = 1$ la condizione è banalmente vera.
Suppongo che la condizione sia vera per tutti i grafi G con $|O| < r$ e prendo un grafo con $|O| = r$. Si hanno due sottocasi.
1)

$$\forall W \subset O \textit{ con } |W| = k\ (k < r) \textit{ si abbia } |G(W)| \geq k + 1 \tag{14}$$

Se considero un elemento $o \in O$ e lo accoppio con un elemento di D ottengo un insieme O' con $r - 1$ elementi in cui vale ancora la condizione iniziale $|W| \leq |G(W)|$ e, utilizzando l'ipotesi induttiva, si accoppiano gli elementi di O' con i restanti elementi di D.

2)

$$\exists W \subset O \; con \; |W| = \; k \; (k < r) \; e \; si \; abbia \; |G(W)| \; = \; k \tag{15}$$

Per l'ipotesi induttiva, posso accoppiare gli elementi di W con quelli di $G(W)$. Devo verificare che lo stesso accada per gli elementi di $O \setminus W$.
Definisco $W' \; = \; O \setminus W$ con

$$h = |W'| = |O| - |W| = r - k \tag{16}$$

dato che $|O| \; = \; r$ e $|W| \; = \; k$. Devo dimostrare che $|G(W') \setminus G(W)| \; \geq h$ in modo da applicare l'ipotesi induttiva, essendo $h \; < \; r$. Lo si dimostra per assurdo supponendo che sia $|G(W') \setminus G(W)| \; < h$.
Si ha:

$$|G(O)| = |G(W) \cup (G(W') \setminus G(W))| \leq |G(W)| + |G(W') \setminus G(W)| < k + h = r = |O| \tag{17}$$

in modo da avere $|G(O)| \; < \; |O|$ contro la condizione necessaria prima dimostrata.
Da qui discende che l'ipotesi fatta è falsa e quindi che è $|G(W') \setminus G(W)| \geq h$ $= |W'|$.
Essendo $h \; < \; r$, per l'ipotesi induttiva, si possono accoppiare gli elementi di W' con elementi distinti di D ovvero con gli elementi di $G(W') \setminus G(W)$ e quindi gli elementi di O con gli elementi di D.

Si ha quindi:

1. accoppiamento M completo di O in D se

$$\forall i \in O \, \exists (i,j) \in M \tag{18}$$

2. accoppiamento M completo di D in O se

$$\forall j \in D \, \exists (i,j) \in M \tag{19}$$

3. se $|O| = |D|$ si può avere un accoppiamento completo tra O e D detto *accoppiamento perfetto* o *assegnamento*.

Al proposito si ha il seguente corollario che dà una condizione sufficente per l'esistenza di un accoppiamento perfetto fra O e D.

Corollario

Se ogni vertice di O è adiacente a k ($k \geq 1$) vertici di D e viceversa allora esiste un accoppiamento perfetto fra O e D.

Dimostrazione

Si dovrebbe dimostrare che esistono:

1. un accoppiamento di O in D,
2. un accoppiamento di D in O,

ma, data la simmetria del problema, si dimostra l'esistenza solo del primo. Sfruttando il teorema di Hall è sufficiente dimostrare che:

$$\forall W \subseteq O \; |W| \leq |G(W)| \tag{20}$$

Se considero $W \subseteq O$ e definisco con F l'insieme degli archi incidenti in W e con F' l'insieme degli archi incidenti in $G(W)$ ho che valgono le relazioni seguenti:

$$|F| = k|W| \tag{21}$$

$$|F'| = k|G(W)| \tag{22}$$

ed essendo $F \subseteq F'$ (dato che il secondo contiene anche archi provenienti da nodi in $O \setminus W$) si ha $|F| \leq |F'|$ per cui, sostituendo le (21) e (22), si ha:

$$k|W| \leq k|G(W)| \rightarrow |W| \leq |G(W)| \tag{23}$$

ovvero la tesi.

La ricerca di accoppiamenti completi ha come caso particolare la ricerca, in un grafo bipartito, di un accoppiamento di cardinalità massima.
Il problema di accoppiamento di massima cardinalità ha una formulazione come problema di Programmazione Lineare Intera in cui, usando le variabili booleane $x_{ij} \in \{0, 1\}$, si introducono i vincoli seguenti:

$$\sum_{j \in D \,:(i,j) \in A} x_{ij} \leq 1 \;\; \forall i \in O \tag{24}$$

$$\sum_{i \in O \,:(i,j) \in A} x_{ij} \leq 1 \;\; \forall j \in D \tag{25}$$

con la funzione obbiettivo

$$max \sum_{(i,j)\in A} x_{ij} \tag{26}$$

in cui $x_{ij} = 1$ se $(i, j) \in M$ e $x_{ij} = 0$ se $(i, j) \notin M$. In pratica si fa uso di algoritmi che sfruttano la topologia del problema e fanno uso dei concetti di cammini alternanti e aumentanti (cfr. la prossima sezione).

3.3 Un algoritmo per l'accoppiamento di massima cardinalità

L'algoritmo parte da un accoppiamento M (in genere si parte da $M = \emptyset$) per costruire un albero aumentante T_r di radice r (un nodo esposto) fino a trovare un altro nodo esposto e determinare un cammino aumentante con cui creare un nuovo accoppiamento, migliore del precedente, oppure fino ad arrivare a stabilre che non esiste un cammino aumentante a partire da r.
La procedura viene ripetuta per ogni nodo esposto e si arresta quando sono stati esaminati tutti i vertici esposti e nessun nodo può più essere accoppiato.
L'albero T_r è formato da più cammini alternanti con in comune la radice r.
I soli cammini che terminano in un nodo esposto sono cammini aumentanti.
I singoli cammini sono accresciuti per aggregazione di nodi e di archi.
Partendo da un nodo r esposto gli spigoli aggiunti sono alternativamente esterni e interni all'accoppiamento corrente M. Se aggiungendo archi (e i nodi in cui questi incidono) si arriva ad un nodo accoppiato si ha un cammino alternante mentre, se il nodo raggiunto è esposto, si ha un cammino aumentante. I nodi, man mano che sono aggiunti all'albero T_r, sono etichettati e classificati, in base sia all'accoppiamento corrente M sia all'albero corrente T_r, come:

1. *outer* (O): lo sono r e i nodi di posto dispari in cammini alternanti,
2. *inner* (I): lo sono i nodi di posto pari in cammini alternanti.

Orientando gli archi di T_r a partire da r i nodi *inner* sono i nodi esposti oppure i nodi iniziali di archi dell'accoppiamento M mentre i nodi *outer* sono la radice r ed i vertici finali di spigoli dell'accoppiamento M.
L'algoritmo (cfr. la figura 6) si compone essenzialmente di cinque passi alcuni dei quali sono ripetuti fino a che non è soddisfatta una condizione di arresto.
I passi dell'algoritmo, che verranno dettagliati più oltre, sono i seguenti:

1. inizializzazione delle strutture dati
2. verifica della condizione d'arresto e inizializzazione dell'albero aumentante T_r

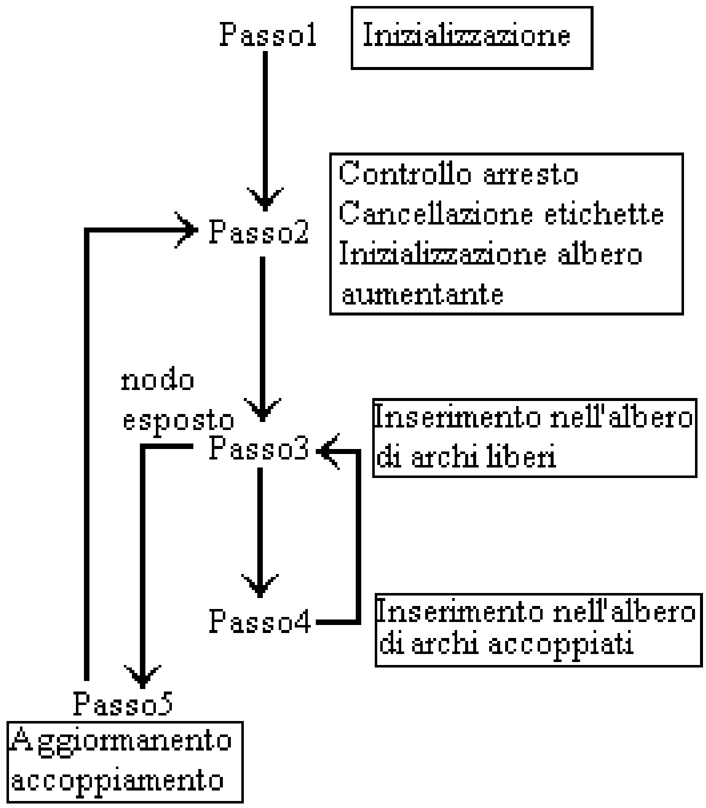

Figura 6: *Schema di algoritmo*

3. inserimento in T_r degli archi esterni,
4. inserimento in T_r degli archi interni,
5. aumento dell'accoppiamento.

Al passo numero (1) si definisce un accoppiamento M iniziale (eventualmente $M = \emptyset$) e si etichetta ogni nodo $i \in O \cup D$ come "non esplorato" (ad esempio con un array di flag booleani $esplorato(i) := false;$).

Al passo (2) si cancellano tutte le etichette O (*outer*) e I (*inner*) assegnate ai nodi (al proposito si può usare un array di char e mettere *etichetta(i):="";* per $i \in O \cup D$).
Inoltre si definisce l'insieme R dei nodi esposti e non ancora esplorati cioè

$$R = \{i \in O \cup D | esplorato(i) = \text{“}false''\& esposto(i) = \text{“}true''\} \tag{27}$$

Fatto questo si hanno i passi seguenti:

1. if $|R| \leq 1$ then return M;
2. select&remove(r,R);
3. *etichetta(r):="O"*;
4. i:=1;
5. A_i:={r};
6. T_r:=A_i;
7. esplorato(r):=true;

Al passo (3) si inseriscono in T_r gli archi liberi ovvero non contenuti nell'accoppiamento corrente M e incidenti in nodi di T_r. A tale scopo si hanno i passi seguenti:

1. i:=i+1; (i assume un valore pari)
2. si definisce A_i come l'insieme dei vertici adiacenti ai vertici di A_{i-1} e che non sono presenti in T_r. Se $A_i = \emptyset$ allora si torna al passo 2 altrimenti si inserisce in T_r sia ogni nodo $q \in A_i$ sia un arco (p, q) con $p \in A_{i-1}$.
3. si etichetta ogni vertice di A_i come I (*inner*);
4. se si ha $q \in A_i$ esposto si prosegue al passo (5).

Al passo (4) si inseriscono in T_r gli archi accoppiati ovvero contenuti nell'accoppiamento corrente M. A tale scopo si hanno i passi seguenti:

1. i:=i+1; (i assume un valore dispari)
2. si definisce A_i come l'insieme dei vertici adiacenti ai vertici di A_{i-1} mediante archi di M e incidenti in nodi di T_r. Si accresce T_r inserendo sia ogni nodo $q \in A_i$ sia gli archi dell'accoppiamento;
3. si etichetta ogni vertice di A_i come O (*outer*);
4. si prosegue al passo (3).

Al passo (5) si aggiorna l'accoppiamento ovvero, dato il cammino aumentante P_A, si esegue l'operazione di composizione fra il cammino P_A e l'accoppiamento corrente M determinando il nuovo accoppiamento ovvero si esegue la seguente istruzione:

$$M := M \oplus P_A = M \setminus P_A \cup P_A \setminus M \tag{28}$$

e si torna al passo (2).

Dato un grafo bipartito $G = (O \cup D, A)$ se il nodo r e i nodi *outer* sono in O allora i nodi *inner* sono in V e viceversa. Un cammino aumentante collega sempre un vertice esposto $o \in O$ con un vertice esposto $d \in D$ per cui, per risolvere il problema, è sufficiente ricavare gli alberi aumentanti con le radici r in O (oppure in D se si guarda il grafo dall'altro lato). Anche se fra due nodi esposti possono esssere presenti più cammini aumentanti la procedura, per come li costruisce, ne determina solo uno.

Complessità
Se il grafo ha $2n$ vertici, l'algoritmo ha complessità $O(n^3)$. Supponendo di partire da $M = \emptyset$ si devono costruire $O(n)$ alberi aumentanti e la costruzione di ciascuno costa $O(n^2)$

3.4 Accoppiamenti di cardinalità massima in grafi bipartiti e problemi di flusso

Un problema di accoppiamento di massima cardinalità in un grafo bipartito $G = (O \cup D, A)$ può essere ricondotto ad un problema di flusso massimo su un grafo esteso $\overline{G}$ (cfr. la sezione 4.1). In tal caso il problema da risolvere ha la formulazione seguente come problema di Programmazione Lineare ma lo si risolve sfruttando la topologia del grafo e utilizzando concetti di cammini alternanti e aumantanti (cfr. la sezione 4.1).

$$x_{si} - \sum_{j \in D \,:(i,j) \in A} x_{ij} = 0 \;\; \forall i \in O \tag{29}$$

$$\sum_{i \in O \,:(i,j) \in A} x_{ij} - x_{jt} = 0 \;\; \forall j \in D \tag{30}$$

con la funzione obbiettivo

$$max \sum_{i \in O} x_{si} \tag{31}$$

in cui s e t sono, rispettivamente, la *sorgente* e il *pozzo* del grafo esteso $\overline{G}$ e per tutti gli archi di $\overline{G}$ è $0 \leq x_{hk} \leq 1$. Se il problema primale è un problema di flusso massimo, il problema *duale* è quello di determinare una sezione di capacità minima ovvero un sottoinsieme di nodi del grafo originario sufficiente a coprire tutti gli archi.

4 Algoritmi "canonici" per grafi bipartiti

Alcuni dei problemi "canonici" che possono essere affrontati e risolti nel caso di grafi bipartiti ([GPS+02]) sono i seguenti.

1. Accoppiamento di *cardinalità massima*: si vuole determinare su G un accoppiamento M la cui cardinalità (indicata con $|M|$) sia massima ovvero in cui sia massimo il numero di archi dell'accoppiamento. Il problema, dati n e d, è superiormente limitato da $min\{n, d\}$.

2. Accoppiamento perfetto (ovvero *assegnamento*) di *costo minimo*. Un assegnamento è un accoppiamento in cui nessun nodo è esposto. In questo caso si assume
$$n = d \tag{32}$$
e si definisce il costo dell'accoppiamento come
$$C(M) = \sum c_{i,j} \forall (i, j) \in M, seM = \emptyset \Rightarrow C(M) = 0 \tag{33}$$
dove $c_{i,j}$ è il costo di ciascun arco del grafo G.

3. Assegnamento *bottleneck* ovvero, sotto l'ipotesi (32), si vuole un assegnamento in cui sia minimo il massimo costo degli archi dell'accoppiamento ovvero in cui sia minimo il valore bottleneck dell'accoppiamento. Il valore *bottleneck* di un accoppiamento M è il valore
$$V(M) = max\{c_{i,j} : (i, j) \in M\} \tag{34}$$

4. Accoppiamento di *massima cardinalità* e *costo minimo*.

5. Accoppiamento di *massima cardinalità* e *valore bottleneck minimo*.

I problemi 4 e 5 sono delle generalizzazioni, rispettivamente, del secondo e del terzo e permettono di fare a meno dell'ipotesi (32) e di supporre $d > n$. Vediamo ora brevemente le tecniche di risoluzione dei problemi suddetti, facendo riferimento a [GPS+02].

4.1 Accoppiamento di massima cardinalità

La ricerca di un accoppiamento di massima cardinalità su un grafo non orientato $G = (N, A)$ può essere ricondotta alla ricerca del flusso massimo in un grafo orientato esteso $\overline{G} = (\overline{N}, \overline{A})$ ottenuto dal grafo G aggiungendo un nodo sorgente s (collegato a tutti i nodi $i \in O$ da archi orientati (s, i)), un nodo pozzo t (collegato a tutti i nodi $j \in D$ da archi orientati (j, t)) e orientando gli archi $(i, j) \in A$ dai nodi di O ai nodi di D (gli archi orientati formano l'insieme A'). Si ha quindi:

$$\overline{N} = O \cup D \cup \{s, t\} \tag{35}$$

$$\overline{A} = A' \cup \{(s, i) \mid i \in O\} \cup \{(j, t) \mid j \in D\} \tag{36}$$

Gli archi di $\overline{A}$ hanno capacità pari a 1 (ovvero è $u_{ij} = 1 \, \forall \, (i, j) \in \overline{A}$).
Dato un flusso x a valori interi uscente da s, l'insieme degli archi saturi di A' in $\overline{G}$ forma un accoppiamento M su G la cui cardinalità coincide con il valore v del flusso. Dato un accoppiamento M su G si costruisce un flusso di valore $v = |M|$ inviando sugli archi corrispondenti a quelli dell'accoppiamento un flusso $x_{ij} = 1$ e tenendo scarichi gli altri, per cui si ha un flusso $v = |M|$.
Dato il grafo G si può costruire il grafo esteso $\overline{G}$ su cui risolvere un problema di flusso massimo che permette di determinare un accoppiamento di cardinalità massima sul grafo di partenza.
In realtà per determinare un accoppiamento di cardinalità massima si sfruttano le caratteristiche topologiche della struttura grafo bipartito arrivando ad una versione modificata dell'algoritmo per cammini aumentanti.
L'algoritmo che si usa parte da un accoppiamento vuoto (ovvero $M = \emptyset$) e cerca un cammino aumentante su $\overline{G}$ su cui spedire una unità di flusso in più in modo da aumentare di una unità la cardinalità dell'accoppiamento M su G. Se trova un cammino aumentante P lo compone con il vecchio accoppiamento M per trovarne uno nuovo M' contenente un arco in più, se non lo trova termina e l'accoppiamento trovato è quello di cardinalità massima.
L'algoritmo cicla fino a che riesce a trovare un cammino aumentante P.
Se sul grafo $\overline{G}$ si ha un flusso x (che corrisponde ad un accoppiamento M sul grafo G) si ha che valgono le seguenti relazioni fra gli archi dei due grafi:

$$x_{ij} = 1 \, per \, (i, j) \in A' \, sse \, (i, j) \in M \tag{37}$$

$$x_{ij} = 0 \, per \, (i, j) \in A' \, sse \, (i, j) \in A \setminus M \tag{38}$$

I nodi di un cammino su $\overline{G}$ devono appartenere alternativamente a O e a D mentre gli archi del cammino possono essere usati in senso concorde al loro verso se non sono saturi e in verso discorde se non sono scarichi. Un cammino aumentante su $\overline{G}$ (rispetto ad un flusso x) corrisponde ad un cammino alternante su G. Il cammino su G però deve essere alternante e aumentante, ovvero deve avere come estremi due nodi esposti in G (cfr. [GPS+02], pag. 112).
Si ha, quindi, una corrispondenza biunivoca fra cammini aumentanti su $\overline{G}$ e cammini alternanti aumentanti su G.

Un cammino alternante P su G rispetto ad un accoppiamento M è un cammino composto alternativamente da archi esterni ad M (archi in $P_E = P \backslash M$) e da archi interni ad M (archi in $P_I = M \cap P$). Un tale cammino è detto essere (alternante) aumentante se inizia e termina con un arco in P_E ovvero se $|P_E| = |P_I| + 1$. Su $\overline{G}$ si ha una operazione di composizione fra flussi indicata come:

$$x' = x \oplus P' \tag{39}$$

in cui P' è un cammino aumentante su cui faccio passare una unità di flusso. Il flusso x' ha valore $v' = v + 1$ se x ha valore v.
Su G si ha una operazione di composizione fra un accoppiamento e un cammino alternante aumentante P indicata come:

$$M' = M \oplus P = M \setminus P_I \cup P_E \tag{40}$$

con

$$|M'| = |M \oplus P| = |M \setminus P_I| + |P_E| = |M| + 1 \tag{41}$$

essendo $|P_E| = |P_I| + 1$.
La procedura che si utilizza ha la struttura seguente:

Procedure Accoppiamento-MaxCard(O,D,A,M)
 begin
  $M = \emptyset$;
  while Cammino-Aumentante(O,D,A,M,P) **do**
   Cambia-Accoppiamento(M,P);
 end

Partendo da un accoppiamento vuoto (ovvero da un flusso nullo) si aplica la *Cammino − Aumentante* in modo da determinare se esiste su G un cammino alternante aumentante. Se lo si trova con esso si aggiorna l'accoppiameto determinandone uno di cardinalità maggiore, se non lo si trova la procedura restituisce "false" e l'algoritmo termina.
La *Cammino − Aumentante* determina un cammino alternante aumentante facendo una visita del grafo G a partire dai nodi esposti di O e percorrendo alternativamente un arco esterno ed un arco interno. Se, durante la visita, si trova un nodo esposto di D si è trovato il cammino cercato e lo si restituisce. Se, a fine visita, non si è trovato nessun nodo di tale tipo allora non esistono cammini alternati aumentanti e la procedura termina restituendo un accoppiamento di massima cardinalità. Se si è trovato il cammino alternante aumentante P si chiama la *Cambia − Accoppiamento* che realizza l'operazione di composizione e determina il nuovo accoppiamento.

Complessità

La procedura termina dopo al più n iterazioni, una per ciascun nodo di O, e ciascuna iterazione costa $O(m)$ perchè richiede una visita del grafo ovvero di al più m archi. Pertanto la complessità è $O(mn)$.

4.2 Assegnamento di costo minimo

In questo caso il problema può essere ricondotto ad un problema equivalente di flusso di costo minimo su un grafo esteso $\overline{G} = (\overline{N}, \overline{A})$ in cui:

$$\overline{N} = O \cup D \cup \{s,t\} \tag{42}$$

$$\overline{A} = A' \cup \{(s,i) \mid i \in O\} \cup \{(j,t) \mid j \in D\} \cup \{(s,t)\} \tag{43}$$

in cui gli archi di A' sono gli archi di A orientati dai nodi di O ai nodi di D.
I nodi hanno bilanci nulli (ovvero sono nodi di transito) tranne s e t: il primo è una sorgente di n (con $n = |O|$) unità di flusso ($b_s = -n$) mentre il secondo è un pozzo di n (con $n = |D|$) unità di flusso ($b_t = n$).
I costi degli archi di A' concidono con il costo degli archi di A mentre i costi degli archi aggiunti sono nulli tranne il costo di (s,t) che viene posto pari a $M = nC + 1$ (dove C è il massimo dei costi non negativi degli archi). Ciò garantisce che nella ricerca di cicli aumentanti di costo negativo si trovi sempre un ciclo di costo negativo qualora si percorra l'arco (s,t) in senso discorde all'orientamento in modo da poter inviare su tale ciclo un flusso $\theta = 1$ (si ha in questo modo un ciclo aumentante di costo negativo).
Le capacità degli archi sono poste uguali a 1 tranne quella dell'arco (s,t) che vale n perchè tale arco deve poter portare tutto il flusso uscente da s a t in due occasioni: se non ci sono cammini aumentanti da s a t oppure nella fase iniziale di un algoritmo per cicli aumentanti di costo negativo.
In pratica si usa, al posto del grafo $\overline{G}$, un grafo ausiliario calcolato partendo dal grafo G su cui si ha un accoppiamento corrente M.
Il grafo ausiliario è $G_M = (N_M, A_M)$ in cui:

$$N_M = O \cup D \cup \{s\} \tag{44}$$

$$A_M = \{(s,j) \mid j \in O_M\} \cup \{(j,i) \mid (i,j) \in M\} \cup \{(i,j) \mid (i,j) \in A \setminus M\} \tag{45}$$

Il grafo G_M contiene gli archi da s ai nodi esposti di O (O_M) con costo nullo ($c_{sj} = 0$), gli archi dell'accoppiamento corrente orientati da D ad O e con costi di segno cambiato ($c_{ji} = -c_{ij}$) e gli archi liberi, ovvero non nell'accoppiamento corrente, con costi di segno invariato.

Il grafo G_M dipende dall'accoppiamento M per cui, ogni volta che si determina un nuovo accoppiamento componendo l'accoppiamento corrente con un cammino aumentante P, lo si deve rideterminare. Lo scopo è quello di determinare un cammino da s a t che coincide con un cammino da s ad un nodo esposto di D per cui nel grafo G_M non compaiono nè il nodo t nè l'arco (s,t), il cui costo è superiore a quello di qualunque cammino (semplice) da s a t che non lo contiene.
Sul grafo G_M si può determinare un cammino aumentante da s a un nodo esposto $d \in D_M$, detto P_{sd}, che corrisponde ad un cammino aumentante P ottenuto rimuovendo l'arco iniziale da P_{sd}. Per come si è costruito il grafo G_M i costi dei due cammini coincidono e il loro valore è ottenibile dalla differenza fra la somma dei costi degli archi esterni (ovvero non appartenenti all'accoppiamento M) del cammino P e la somma degli archi interni (ovvero appartenenti all'accoppiamento M) del cammino P.
L'operazione di invio di una unità di flusso su un cammino aumentante corrisponde alla definizione di un nuovo accoppiamento mediante l'applicazione della operazione di composizione:

$$M' = M \oplus P \tag{46}$$

il cui costo è dato dal costo dell'accoppiamento M più il costo del cammino P.
La procedura utilizzata ha la seguente struttura:

Procedure Accoppiamento-MinCost(O,D,A,c,M)
  begin
    Inizializza(M,O_M,D_M,G_M);
    while SPT(G_M,P) do
      Cambia-Accoppiamento(M,P,O_M,D_M,G_M);
  end

La procedura *Inizializza(M,O_M,D_M,G_M)* determina un accoppiamento iniziale di costo minimo fra quelli di una data cardinalità e, sulla base dell'accoppiamento M, determina sia gli insiemi dei nodi esposti (O_M e D_M) sia il grafo ausiliario iniziale G_M.
Un modo di implementare la Inizializza(M,O_M,D_M,G_M) può essere il seguente:

1. si ordinano per costo crescente gli archi di A,
2. preso $(i,j) \in A$ lo si marca come esplorato e lo si inserisce in M se non viola la definizione di accoppiamento,

3. si continua ad esaminare gli archi fino a che non si sono eplorati tutti oppure non si è arrivati a $|M| = n$.

La procedura SPT(G_M,P) determina nel grafo ausiliario corrente un albero dei cammini minimi T di radice s per i nodi del quale determina le etichette $d(i)$. Un modo per determinare un cammino aumentante di costo minimo P è quello di prendere il nodo $j \in D_M$ con il valore minimo dell'etichetta e rimuovere l'arco iniziale uscente da s. Se la SPT non trova tale cammino, ovvero non trova un nodo $j \in D_M$ che abbia una etichetta di valore finito, allora resituisce "false" e l'algoritmo termina. Ciò accade di sicuro se in O non si hanno più nodi esposti (ovvero se $O_M = \emptyset$). Se la SPT trova un nodo esposto in D_M la procedura restituisce un cammino aumentante P che la $Cambia - Accoppiamento$ compone con l'accoppiamento corrente M in modo da determinare un accoppiamento con un arco in più. Fatto questo la procedura determina i nuovi insiemi dei nodi esposti in O e in D e il nuovo grafo ausiliario G_M su cui la SPT cercherà un nuovo cammino aumentante. Se $|O| = |D|$ e l'algoritmo si arresta senza che in O vi siano più nodi esposti allora l'algoritmo restituisce un assegnamento di costo minimo altrimenti, se in O o in D rimangono nodi esposti, si parla di accoppiamento di massima cardinalità e costo minimo.

Terminazione

L'algoritmo termina in un numero finito di passi perchè l'insieme dei nodi O esaminati per la ricerca dei cammini aumentanti ha cardinalitè finita e perchè la SPT termina in un numero finito di passi e lo stesso vale per la $Cambia - Accoppiamento$.

Complessità

Il ciclo *while* viene ripetuto al più n volte (una per ogni nodo dell'insieme O) e ogni iterazione costa $O(mn)$ (è il costo della SPT se l'insieme Q è implementato come una fila) per cui la complessità è $O(mn^2)$.

4.3 Assegnamento bottleneck

Dato un grafo bipartito $G = (O \cup D, A)$ in cui agli archi sono associati i costi c_{ij} il valore bottleneck $V(M)$ di un un accoppiamento o di un assegnamento M è il massimo valore dei costi di M ovvero $V(M) = max\{c_{ij} | (i,j) \in M\}$. Un accoppiamento bottleneck è un accoppiamento M il cui valore bottleneck è minimo mentre un assegnamento bottleneck è un accoppiamento bottleneck

perfetto ovvero tale che se $|O| = |D| = n$ allora è $|M| = n$. La ricerca di un accoppiamento/assegamanto bottleneck (nel seguito AB) si basa su un algoritmo iterativo che fa uso, ad ogni passo, di un grafo parziale ricavato incrementalmente dal grafo originario G sulla base del valore di un parametro v.
Supponendo di conoscere il valore bottleneck ottimo z si prende un valore di v in modo da ricavare da G un grafo ridotto G_v privo di accoppiamenti perfetti. Per essere sicuri di ciò basta prendere $v < z$ (infatti z è il costo dell'AB di costo minimo) e lo si può fare prendendo

$$v = min\{c_{ij}|(i,j) \in A\} \tag{47}$$

Fissato v si ricava il grafo ridotto $G_v = (O \cup D, A_v)$ in cui

$$A_v = \{(i,j) \in A\,|c_{ij} \leq v\} \tag{48}$$

Ottenuto il grafo ridotto, su di esso si determina un accoppiamento di massima cardinalità M. Se si ha $|M_v| = n$ oppure $A_v = A$ allora l'algoritmo si arresta e restituisce l'accoppiamento M_v che, a seconda dei casi, è un assegnamento oppure una accoppiamento di massima cardinalità.
Se nessuna delle due condizioni di arresto è soddisfatta si prende un nuovo valore di v maggiore del precedente usando la relazione seguente:

$$v = min\{c_{ij}|(i,j) \notin A\} \tag{49}$$

e sulla base del nuovo valore si determina un nuovo grafo ridotto $G_v = (O \cup D, A_v)$ che contiene gli archi del vecchio grafo più alcuni archi i cui costi sono uguali al valore v e per il quale, partendo dall'accopiamento trovato M_v se ne trova un'altro con un arco in più.
Lo schema della procedura in pseudocodice è il seguente:

Procedure Accoppiamento-Bottleneck(O,D,A,c,M_v,v)
    begin
        Inizializza(v,A_v,M_v);
        while $A_v \neq A$ **and** $|M_v| < n$
        begin
            v:=$min\{c_{ij} \mid (i,j) \notin A_v\}$;
            $A_v := A_v \cup \{(i,j) \notin A_v \mid c_{ij} = v\}$;
            Accoppiamento-MaxCard(O,D,A_v, M_v);
        end
    end

La procedura $Inizializza$ determina un valore di v dall'esame del grafo G e, sulla base di questo, produce un accoppiamento iniziale M_v di massima cardinalità nel grafo G_v.
La procedura $Inizializza$ può avere la struttura seguente:

Procedure Inizializza(v,A_v,M_v)
begin
foreach $i \in O$ $v[i] = min\{c_{ij} | (i,j) \in FS(i)\}$;
v:=max $v[i]$;
$A_v := \{(i,j) \in A \mid c_{ij} \leq v\}$;
Accoppiamento-MaxCard(O,D,A_v, M_v);
end

Nella procedura $Accoppiamento - Bottleneck$ si usa inizialmente l'accoppiamento M_v determinato dalla procedure $Inizializza$ e, ad ogni iterazione, si fa uso dell'accoppiamento trovato nella iterazione precedente che è un accoppiamento valido dato che gli archi presenti in esso ad una iterazione sono presenti anche alla iterazione successiva.
Ordinando i costi degli archi in senso crescente, ad ogni iterazione si riesce facilmente a determinare il nuovo valore di v in base al quale determinare il nuovo grafo ridotto G_v su cui cercare un nuovo accoppiamento di massima cardinalità.
L'algoritmo termina in due casi:

1. se $A_v = A$,
2. se $|M_v| = n$.

Nel primo caso non si hanno più archi da aggiungere mentre nel secondo caso si ha un accoppiamento perfetto ovvero un assegnamento: l'algoritmo permette pertanto di determinare un assegnamento bottleneck, se esiste, oppure un accoppiamento bottleneck di massima cardinalità con un approccio alla $Kruskal$, basato sull'aggiunta incrementale di archi in ordine crescente di costo.
L'uso dell'accoppiamento M_v determinato ad una iterazione come valore iniziale della successiva è giustificato dal fatto che se un arco appartiene ad un insieme A_v ad una iterazione appartiene anche all'insieme A_v della iterazione successiva, determinato da un nuovo valore di v maggiore del precedente. In più tale utilizzo consente di ridurre i numero dei cammini aumentanti a n e garantisce la correttezza dell'algoritmo. La procedura $Accoppiamento - MaxCard$, infatti, non conosce i costi degli archi e potrebbe, ripartendo tutte le volte dall'accoppiamento $M = \emptyset$, determinare un accoppiamento di pari cardinalità ma costo maggiore, nel caso in cui non

esistono accoppiamenti perfetti. Tale problema viene evitato utilizzando, ad ogni iterazione, l'accoppiamento determinato alla iterazione precedente.

Complessità.

La complessità è $O(mn)$

5 Un algoritmo per grafi non bipartiti

5.1 Definizioni e notazione per grafi non bipartiti

Nel caso dei grafi non bipartiti $G = (N, A)$ i problemi che si tovano nella applicazione degli algoritmi sviluppati per i grafi bipartiti e che obbligano ad apportare a questi delle modifiche sono legati alla presenza nei grafi G di cicli contenenti un numero dispari di archi.
Dato un grafo G in esso possono essere presenti:

1. cicli con un numero pari di archi e di nodi,
2. cicli con un numero dispari di archi e di nodi.

Mentre i cicli del primo tipo portano ad una etichettatura univoca dei nodi indipendentemente dal verso di percorrenza del ciclo, quelli del secondo tipo non godono di tale proprietà e, quindi, richiedono un trattamento speciale.

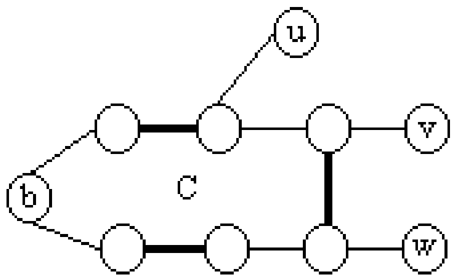

Figura 7: *Esempio di ciclo con numero dispari di elementi*

Con riferimento alla figura 7 ([AMRJ96]) e con riferimento ad un accoppiamento M (i cui archi compaiono in grassetto nella figura 7) si ha che se un ciclo ha $2k + 1$ archi allora k sono in M e $k + 1$ sono liberi per cui si hanno almeno due archi adiacenti che non fanno parte di M (ad esempio, in figura 7, i due archi incidenti nel nodo b): nel ciclo C si ha allora un nodo in cui non incidono archi di M e che diremo *esposto in* C (il nodo b in figura 7).
In C si ha un cammino aumentante dal nodo b ai nodi adiacenti a nodi del ciclo ed esposti quali u, w e v e tali cammini aumentanti percorrono gli archi del ciclo in direzioni orarie o antiorarie e, se hanno archi a comune, li percorrono in direzioni opposte, cosa che non accade in cicli di un numero pari di elementi in cui cammini che percorrono il ciclo in direzioni opposte sono disgiunti.
I circuiti con un numero dispari di elementi creano problemi perchè o non

consentono di etichettare correttamente un nodo in modo da stabilire se è accoppiato o esposto oppure permettono di individuare cammini che sembrano essere aumentanti e invece non lo sono. Una soluzione è la seguente.
Se in un grafo non bipartito G si trova un circuito Γ con un numero di archi pari a $2k + 1$ (di cui k appartengono all'accoppiamento corrente M) e con due archi non accoppiati che individuano un nodo esposto nel circuito (o ciclo) C ad esso si può sostituire uno *pseudovertice* g collegato a tutti i nodi di G adiacenti ai nodi di C.
Dato un grafo G e un accoppiamento M, si dice *germoglio* o $inflorescenza$ Γ un circuito C con $2k + 1$ spigoli tale che (cfr. la figura 8, [AMRJ96]) :

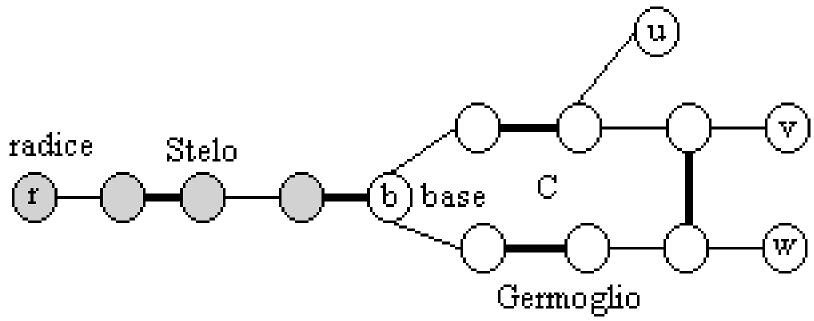

Figura 8: *Esempio di germoglio*

1. Γ contiene k spigoli accoppiati e un solo vertice b esposto nel ciclo, ovvero non accoppiato con altri vertici del ciclo, detto *base* del germoglio,

2. esiste un cammino alternante P da un vertice r (detto *radice* del germoglio), esposto rispetto ad M, al vertice b, detto *stelo* del germoglio (cfr. i nodi in colore della figura 8).

Inoltre si ha:

1. lo stelo contiene un numero pari di archi perchè inizia con un arco esterno e termina con uno interno,

2. la radice coincide con la base del germoglio se lo stelo non contiene archi,

3. dalla radice r ho un cammino alternante a qualunque vertice di G adiacente a un qualunque vertice del germoglio Γ (percorrendo gli archi del germoglio in senso orario o antiorario a seconda dei casi).

Una volta definito il concetto di germoglio Γ si ha bisogno di un criterio per *individuare* un germoglio durante la costruzione di un albero aumentante T_r e di un metodo per *sostituire* ad un germoglio un oggetto che non abbia gli stessi problemi e che non introduca alterazioni nella ricerca dei cammini aumentanti.
Si individua un germoglio quando, durante la costruzione di T_r, si scopre che un nodo già inserito in T_r ed etichettato (come *outer* o come *inner*) dovrebbe essere aggiunto di nuovo all'albero aumentante con una etichetta opposta ha quella che a già avuto in una etichettatura precedente.
L'elemento che si sostituisce ad un germoglio Γ, come già visto, è uno pseudoverice g collegato a tutti i nodi di G adiacenti ai vertici del circuito la cui contrazione produce lo psudovertice g.
Per giustificare il metodo di contrazione dei germogli si hanno i seguenti teoremi.

Teorema

Dato un grafo G e un accoppiamento M se durante la ricerca di un cammino aumentante P a partire da un nodo esposto r trovo un germoglio Γ allora ho un cammino alternante da r ad ogni vertice $i \in \Gamma$ che termina con un arco di M. Da ciò discende che da r esiste un cammino aumentante ad ogni nodo esposto di G che sia adiacente ad un nodo di Γ: ciò permette di contrarre il germoglio Γ senza che ciò alteri i cammini aumentanti esistenti.

Teorema

Si supponga che durante la ricerca di un cammino aumentante dal nodo r si trova un germoglio Γ in modo da passare, grazie alla contrazione del germoglio, dal grafo G al grafo contratto G^*.
Si ha un cammino P aumentante da r in G se e solo se ne esiste uno in G^*.

Dimostrazione

Suppongo di avere un cammino aumentante P^* su G^* e dimostro che esiste un cammino aumentante P su G.
Dato P^* ho due casi possibili:

1. P^* non contiene uno pseudovertice g ottenuto dalla contrazione di un germoglio Γ,

2. P^* contiene uno pseudovertice g ottenuto dalla contrazione di un germoglio Γ.

Nel primo caso basta prendere $P = P^*$. Nel secondo caso come cammino P considero il cammino che parte dal nodo esposto r arriva al nodo base del germoglio prosegue lungo gli archi del cammino di ordine pari del germoglio e arriva al nodo esposto destinazione d. Tale cammino esiste per il teorema precedente per cui ho trovato il cammino aumentante P.
Suppongo ora di avere un cammino aumentante P su G e dimostro che esiste un cammino aumentante P^* su G^*.
Dato P ho due casi possibili:

1. P non include nessun germoglio Γ,

2. P include un germoglio Γ.

Nel primo caso basta prendere $P^* = P$. Nel secondo caso come cammino P^* considero il cammino aumentante ottenuto dalla contrazione di un germoglio Γ in uno pseudovertice g e sfrutto ancora il teorema precedente.

Teorema

Si suppone che in un grafo G non vi sono cammini aumentanti a partire da un nodo u rispetto ad un accoppiamento M. Se si trova un cammino aumentante P rispetto a M tra due vertici v e w è possibile determinare un nuovo accoppiamento $M' = M \oplus P$ ma, di nuovo, non si hanno cammini aumentanti dal nodo u neanche rispetto al nuovo accoppiamento M'.

Si nota, infine, come un germoglio Γ possa contenere al suo interno uno pseudovertice g ottenuto dalla precedente contrazione di un altro germoglio. Una volta che si è ottenuto un accoppiamento M^* di cardinalità massima per il grafo G^* si ottiene l'accoppiamento M di massima cardinalità per G riespandendo gli pseudovertici contrazione dei vari germogli individuati nel grafo G durante la costruzione dei cammini aumentanti. La riespansione deve essere fatta in modo opportuno, rispettando le regole degli accoppimenti in base alla struttura interna del germoglio e del tipo degli archi in esso incidenti.

5.2 Un algoritmo per grafi non bipartiti

L'algoritmo (cfr. la figura 8) è caratterizzato dai passi seguenti:

1. Inizializzazione
2. Criterio di arresto
3. Esplorazione della stella di un nodo u
4. Accrescimento dell'albero T_r
5. Contrazione di un germoglio
6. Aggiornamento dell'accoppiamento

Durante la fase di $Inizializzazione$ si determina un accoppiamento iniziale qualunque M (eventualmente $M = \emptyset$) e in più si pone:

1. $G^* := G$
2. $M^* := M$
3. $foreach\ i \in N\ nodoesplorato[i]\ :=\ false;$
4. $foreach\ h \mid a_h \in A\ arcoesplorato[h]\ :=\ false;$

Il passo (2) contiene i seguenti passi:

1. $foreach\ i \in N\ etichetta[i]\ :=$ “”;
2. determino R, l'insieme dei vertici esposti non esplorati
3. if $|R| \leq 1$ then return M^* else begin scelta di $r \in R$; $etichetta[r] := O$; $T_r := \{r\}$; end

Al passo (3) si sceglie un nodo u etichettato come O fra quelli presenti in T_r e che abbia almeno un arco uscente non esplorato. Se un tale nodo non esiste si marca il nodo r come esplorato, si marcano gli archi come non esplorati e si torna al passo (2) ovvero:
$nodoesplorato[r]\ :=\ true;$
$foreach\ h \mid a_h \in A\ arcoesplorato[h]\ :=\ false;$
go to (2);
Se il nodo ha un arco uscente non esplorato si va al passo (4).

Al passo (4) si arriva solo se il nodo u scelto in T_r di etichetta O ha almeno un arco uscente non esplorato (u, v). Si hanno i tre casi per v: v ha

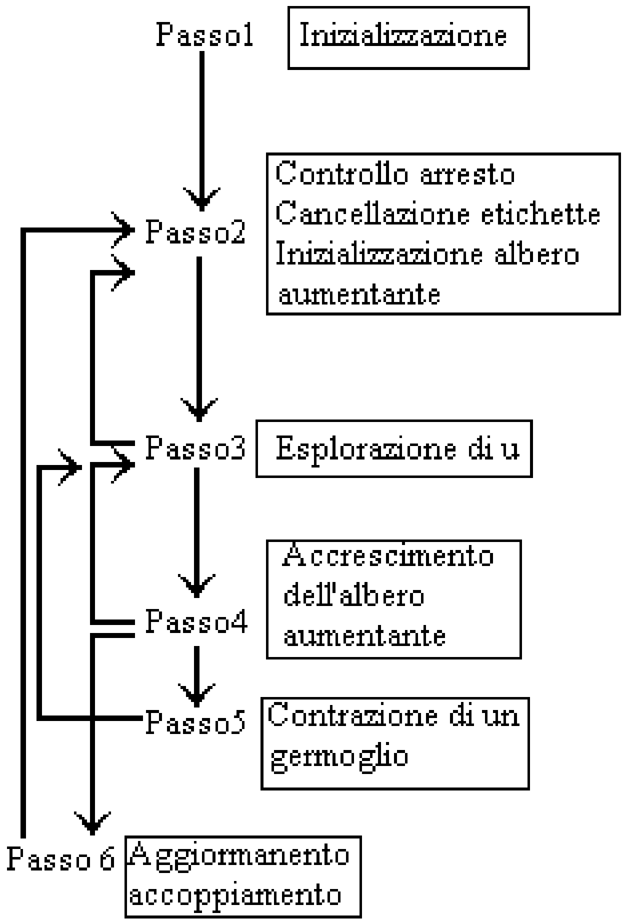

Figura 9: *Schema dell'algoritmo*

etichetta I per cui si sceglie un altro nodo (o anche lo stesso) e un altro arco non esplorato; v ha etichetta O come u e si ha un germoglio oppure v non ha etichetta e può essere o esposto (si è trovato un cammino aumentante) oppure accoppiato (e si inserisce in T_r anche l'arco accoppiato.
Il passo (4) contiene i seguenti passi:

1. $nodoesplorato[u]$:= $true$
2. if(etichetta(v)="I") then go to (3);
3. if(etichetta(v)="O") then go to (5);
4. if(etichetta(v)="") then
   begin

insert(v,(u,v),T_r);
etichetta(v):=“I”;
if(esposto(v)=true) then goto (6)
else
begin
add(T_r, $(v,w)\ \in\ M^*$);
etichetta(w)=“O”;
go to (3);
end;
end;

Il passo (5) contiene i seguenti passi:

1. contrazione del germoglio Γ in uno pseudovertice g_i;

2. aggiornamento di G^*, M^* e T_r;

3. etichetta[g_i]:=“O”; go to (3);

Il passo (6) contiene i seguenti passi:

1. $M^* := M^* \oplus P_A$;

2. $foreach\, h \mid a_h \in A\, esplorato[h]\ =\ false$;

Lo pseudovertice g_i è etichettato come *outer* perchè in g_i incide al più un arco dell'accoppiamento.Tale etichettatura ci consente di utilizzare g_i alla iterazione successiva come radice dell'albero aumentante.

Complessità

La complessità, per un grafo G non bipartito con n nodi, è $O(n^4)$ ma può essere ridotta a $O(n^3)$ come nel caso di un grafo bipartito.

6 La copertura di cardinalità minima

Dato un grafo non orientato $G = (N, A)$ per risolvere un problema di copertura si deve trovare un sottoinsieme degli archi ($C \in A$ tale che ogni vertice di N sia coperto da almeno un arco ovvero tale che in ogni nodo $i \in N$ incida almeno un arco di C. Gli archi di C, contrariamente a quello che accade nel caso di un accoppiamento, possono anche risultare adiacenti. Nel caso dei problemi di copertura si hanno:

1. coperture di cardinalità minima

2. coperture di costo minimo

Utilizzando le variabili booleane x_{ij} assocaite ai singoli spigoli (con la convenzione che sia $x_{ij} = 1$ se $(i, j) \in C$ e $x_{ij} = 0$ altrimenti) il problema della copertura di cardinalità minima può essere espresso come segue:

$$min \sum_{(i,j)\in A} x_{ij} \tag{50}$$

$$\sum_{j:(i,j)\in A} x_{ij} \geq 1 \quad \forall i \in N \tag{51}$$

Tale probelma lo si può risolmere a partire da un accoppiamento di cardinalità massima per lo stesso grafo G. A tale proposito si anno i seguenti teoremi.

Teorema
Se M è un accoppiamento di cardinalità massima per un grafo G e a ciascun vertice i esposto in G (ovvero su cui non incide nesssun arco di M) si aggiunge uno spigolo incidente in i si ottiene una copertura di cardinalità minima.

Teorema
Se C è una copertura di cardinalità minima per un grfao G e per ogni vertice i si cancellano tutti gli spigoli in esso incidenti meno 1 si ottiene un accoppiamento di cardinalità massima.

Il problema della copertura di costo minimo ha una formulazione analoga alla precedente in cui cambia solo la funzione obbiettivo che risulta essere

$$min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{52}$$

in cui c_{ij} sono i costi degli archi $(i, j) \in A$.

Riferimenti bibliografici

[AMRJ96] Giovanni Andreatta, Francesco Mason, and Giorgio Romanin-Jacur. *Ottimizzazione su reti.* Edizioni Libreria Progetto Padova, 1996.

[GPS+02] Giorgio Gallo, Stefano Pallottino, Maria Grazia Scutellà, Antonio Frangioni, and Giancarlo Bigi. *Appunti di Ricerca Operativa.* Diaprtimento di Informatica, Univerità di Pisa, 2002.

[Ore76] Oystein Ore. *I Grafi e le loro applicazioni.* Zanichelli, 1976.