Capitolo 2

Reti di Clos riconfigurabili in ambiente classical switching

2.1 Introduzione

In questo capitolo presentiamo i principali risultati sulle problematiche legate alle reti di Clos riconfigurabili in ambiente classical switching.
I problemi che prenderemo in esame sono due:

1. Fissati i paramentri n ed r, vogliamo determinare il minimo numero m di switch centrali affinchè la rete di Clos C(n,m,r) sia riconfigurabile (*Problema di riconfigurabilità*).
2. Data una C(n,m,r) riconfigurabile, determinare un algoritmo che per ogni frame F di $\Phi(n,r)$ costruisca una configurazione R che realizzi F e che sia compatibile con C(n,m,r) (*Problema di routing*).

Nel paragrafo 2.4 risolveremo il primo problema, mentre nel 2.5 faremo una rassegna dei principali algoritmi di routing per reti di Clos riconfigurabili.
Ora seguono una serie di definizioni e proprietà matematiche che ci consentiranno nel paragrafo 2.3, di definire i due problemi sopra esposti in termini di teoria dei grafi ed algebra lineare.

2.2 Definizioni, proprietà e notazioni matematiche

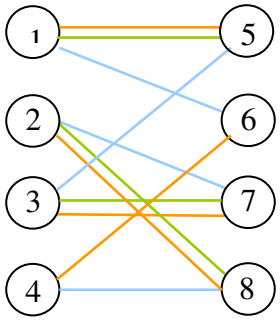
Nel paragrafo 1.2 abbiamo visto alcune nozioni sui grafi. Nel prossimo le estenderemo introducendo i concetti di *edge-coloring* e *matching*.
Nel paragrafo 2.2.2 invece daremo definizioni e proprietà riguardanti l'algebra lineare ed il calcolo combinatorio.

2.2.1 Matching ed edge-coloring

Sia G=(V,E) un multigrafo. Un *matching* di G è un insieme M di archi, in cui non esiste una coppia di archi incidenti su un medesimo vertice. Sia $C \subseteq V$ l'insieme

dei vertici su cui incidono gli archi del matching M, si dice che C è *coperto* da M oppure che M è *C-containing*.
Un *matching massimo* M è un matching di G che ha cardinalità massima. Nel caso in cui M copra tutti i vertici V (V-containing) allora diciamo che M è un *matching perfetto*.

Una *colorazione degli archi* o *edge-coloring* (*EC*) di un multigrafo G, è un assegnamento di un colore ad ogni arco, in modo che su ogni vertice incidano archi di colori diversi. Una *colorazione parziale*, è un edge-coloring in cui alcuni archi non sono colorati.



L'*indice cromatico* di G, $\chi(G)$, è il minimo numero di colori utilizzabile in un edge coloring di G mentre una colorazione degli archi di G che utilzza $\chi(G)$ colori è detta *colorazione minima* o *edge coloring minimo*.
Uno dei principali risultati riguardanti l'edge coloring è il seguente lemma :

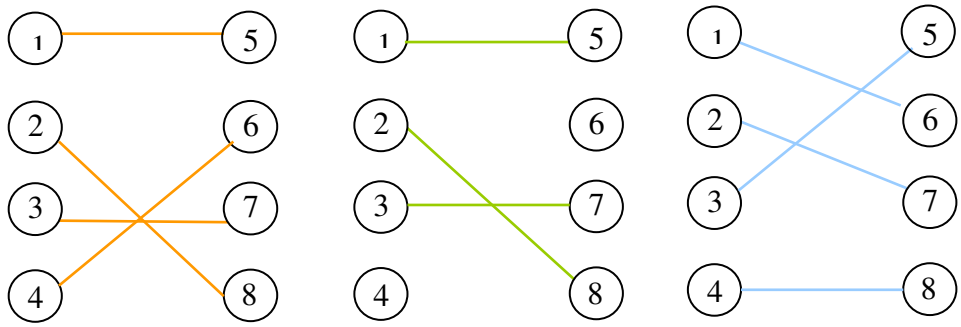
Lemma 2.1(König 1916) [CSKoni16]
Sia G un multigrafo bipartito, allora vale:

$$\chi(G) = deg(G)$$

Ovvero, il numero minimo di colori che si possono utilizzare per un edge coloring di un multigrafo bipartito, coincide con il grado del multigrafo.

Dalla definizione di matching, segue che in un edge coloring, tutti gli archi di uno stesso colore formano un matching e che il multinsieme E può essere partizionato in non meno di $\chi(G)$ matching, infatti se fosse possibile trovare una partizione di

E in $\chi(G)$-1 matching, allora esisterebbe un EC di G con $\chi(G)$-1 colori, il che è impossibile per definizione di $\chi(G)$.



Un edge coloring di un multigrafo G fornisce quindi una partizione di E in tanti matching quanti sono i colori utilizzati.
Possiamo riassumere il rapporto esistente tra matching ed edge coloring con la seguente proprietà:

Proprietà 2.1
Un multigrafo $G=(V,E)$ ammette un edge coloring con m colori se e solo se E può essere partizionato in m matching.

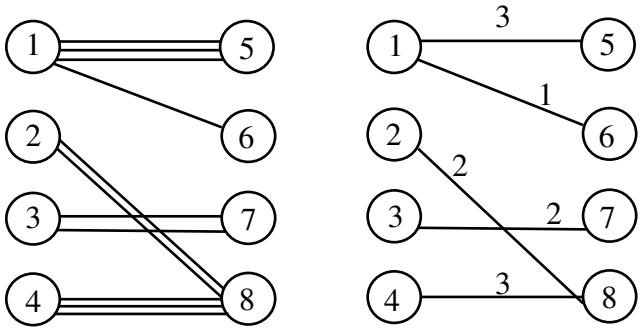
Se indichiamo con $\alpha(G)$ la cardinalità minima di una partizione di E in matching, allora dalla proprietà precedente segue immediatamente che $\chi(G)=\alpha(G)$.

Un altro importante risultato che riguarda i multigrafi bipartiti è il seguente:

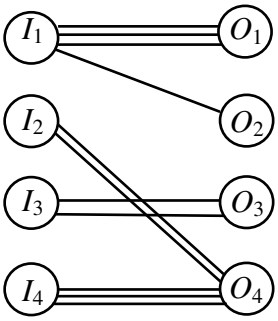
Lemma 2.2 (Mendelsohn & Dulmage) [ECMeDu58]
Sia dato G=(I,O,E), esiste sempre un matching che copre ogni vertice di grado massimo contenuto in $I \bigcup O$.

Sia $G=(I,O,E)$ un multigrafo bipartito. Chiamiamo *grafo sotteso di G*, il grafo bipartito $G_S=(I,O,E_S)$ definito nel modo seguente: per ogni arco $(u,v)\in E$ con molteplicità k, in G_S costruiamo un solo arco (u,v) etichettato con k.

Esempio



Sia $G=(I,O,E)$ un multigrafo bipartito con $I=\{I_1,...,I_r\}$ e $O=\{O_1,...,O_r\}$. Definiamo *matrice d'incidenza di G*, quella matrice A di dimensione $r\times r$ in cui ogni suo elemento a_{ij} è la molteplicità dell'arco (I_i,O_j).



$$A=\begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

Matrice d'incidenza

Osserviamo che la somma degli elementi della riga i (colonna j) di A, corrisponde al grado del vertice I_i (O_j) di G. Dunque se G è d-regolare la somma degli elementi di ogni linea di A, vale d.

Proprietà 2.2

Sia $G=(I,O,E)$ un multigrafo bipartito e $G_S=(I,O,E_S)$ quello sotteso.

M è un matching di G se e solo se M è un matching di G_S.

Dimostrazione

Trascuriamo le etichette degli archi di G_S. Se $M\subseteq E$ allora per definizione di G_S per ogni (u,v) di M esiste un arco (u,v) in E_S e $M\subseteq E_S$. Essendo M un matching di

G, segue che gli archi (u,v) non hanno vertici in comune e quindi M è un matching anche per G_S. L'implicazione inversa si dimostra allo stesso modo.

Dalla proprietà segue inoltre che se M è un matching perfetto per G allora lo è anche per G_S e viceversa.

2.2.2 Algebra Lineare

Sia E una matrice d'interi non negativi e di dimensione $r \times r$. Se in ogni riga e colonna esiste al più un elemento uguale ad uno e tutti gli altri sono zero, diciamo che E è *elementare*.

Esempio:

$$E=\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Sia P una matrice elementare di dimensione $r \times r$. Se P contiene esattamente r elementi non nulli, diciamo che P è una matrice di *permutazione*.

Sia T una matrice d'interi non negativi di dimensione $r \times r$. Chiamiamo *decomposizione* di T una sequenza S di matrici elementari $\{E_1,\ldots,E_m\}$ tali che:

$$T = E_1 + \ldots\ldots\ldots\ldots + E_m$$

m è chiamata *lunghezza* o *dimensione* della decomposizione.
Inoltre, indichiamo con $\rho(T)$ la lunghezza *minima* della decomposizione di T.
Osserviamo che almeno una decomposizione di T esiste sempre. Ad esempio si può costruire come segue: per ogni elemento t_{ij} di T si costruiscono t_{ij} matrici elementari in cui tutti gli elementi, tranne quello di posizione (i,j), sono nulli.

Se una matrice T è tale che ogni sua ogni linea (riga o colonna) ha la stessa somma d, diremo che T è *d-regolare*.

Sia S un insieme di elementi ed $I=\{1,2,....,n\}$ un insieme finito di indici.

Per ogni $i \in I$, sia S_i un sottoinsieme di S. Un *Sistema di Rappresentanti Distinti* (SDR) degli insiemi S_i è un insieme $\{x_1,....,x_n\}$ di n elementi tale che:

- $x_i \in S_i \qquad i=1,..,n$
- $x_i \neq x_j \qquad \forall\ i \neq j$

Enunciamo un teorema che dà una condizione necessaria e sufficiente affinché esista un SDR.

Teorema di Hall [CSHall35]

Sia S un insieme di elementi ed $I=\{1,2,....,n\}$ un insieme finito di indici. Per ogni $i \in I$, sia S_i un sottoinsieme di S.

Un SDR degli S_i esiste se e solo se, per ogni $k=1,...,n$ e per ogni scelta di k indici distinti $i_1,....,i_k$, i sottoinsiemi $S_{i_1},....,S_{i_k}$, nel complesso, contengono almeno k elementi distinti.

Un'applicazione del teorema di Hall ci permette di esprimere una proprietà delle matrici d-regolari.

Corollario 2.1[CSHall67]

Sia T una matrice di numeri reali non negativi, di dimensione $r \times r$. Indichiamo con $S=\{1,2,....,r\}$ l'insieme degli indici di colonna di T. Siano $S_1,...,S_r$ r sottoinsiemi di S dove il generico S_i è costituito dagli indici di colonna tali che $t_{ij}>0$. Se ogni linea di T ha la stessa somma, allora $S_1,...,S_r$ ammettono un SDR.

Dimostrazione

Supponiamo per assurdo che la tesi non sia vera. Per il teorema di Hall esistono k sottoinsiemi S_i che nel complesso al più contengono (k-1) elementi distinti. Consideriamo la matrice Q composta dalle k righe di T corrispondenti agli insiemi S_i. Possiamo affermare che esiste almeno una colonna x di Q che è tutta nulla. Infatti, abbiamo detto che gli S_i in tutto ammettono al massimo k-1 elementi distinti quindi esiste un indice x di colonna che non compare in nessun S_i. Per definizione degli S_i segue che tutte le righe di Q hanno l'elemento in posizione x nullo.

Facciamo vedere che se eseguiamo la somma di tutti gli elementi di Q in modi differenti, otteniamo risultati diversi e quindi una contraddizione.

Sommiamo tutti gli elementi di Q per riga. Per ipotesi ogni linea di T ha stessa somma d, quindi avendo ogni riga di Q ancora questa proprietà, la somma vale dk. Facendo invece la somma di tutti gli elementi di Q per colonna, al più otteniamo $(k\text{-}1)d$. Infatti, ogni colonna j di Q contiene solo una porzione degli elementi della corrispondente colonna di T, quindi, per l'ipotesi del teorema, ogni colonna di Q ha somma al più d. Poiché esiste almeno una colonna tutta nulla, segue che la somma di tutti gli elementi di Q è al più $(k\text{-}1)d$.
Abbiamo così ottenuto una contraddizione. □

2.3 Modelli matematici

Le problematiche inerenti la riconfigurabilità delle reti di Clos, possono essere formulate con tre modelli matematici diversi.
I primi due modelli che vedremo sono espressi in termini di teoria dei grafi mentre il terzo utilizzerà alcuni concetti legati alla decomposizione di una matrice.

2.3.1 Riconfigurabilità ed edge coloring

Riassumiamo in tre semplici proprietà il rapporto tra riconfigurabilità di una rete di Clos C(n,m,r) e l'edge coloring di multigrafi bipartiti.
Proprietà 2.3 *Rete di Clos riconfigurabile*
Una C(n,m,r) è riconfigurabile se e solo se, per ogni frame F, il grafo delle connessioni G_F ammette un edge coloring con m colori.

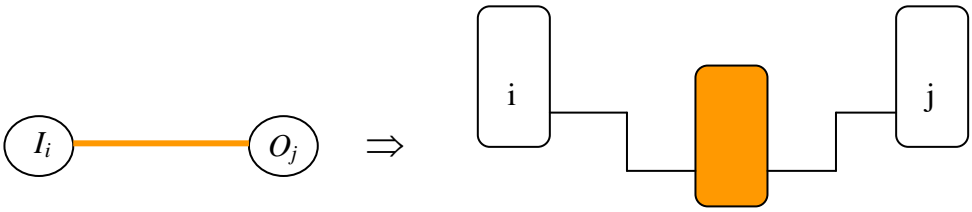
Dimostrazione

$\chi(G_F) \leq m \ \forall F \in \Phi(n,r) \Rightarrow$ C(n,m,r) *è riconfigurabile*

Quello che vogliamo dimostrare è che per ogni frame F a partire da una colorazione di G_F, si può ottenere una configurazione R che realizza F e che è compatibile con C(n,m,r).
Una configurazione R per il frame F compatibile con la rete, può essere costruita nel seguente modo:

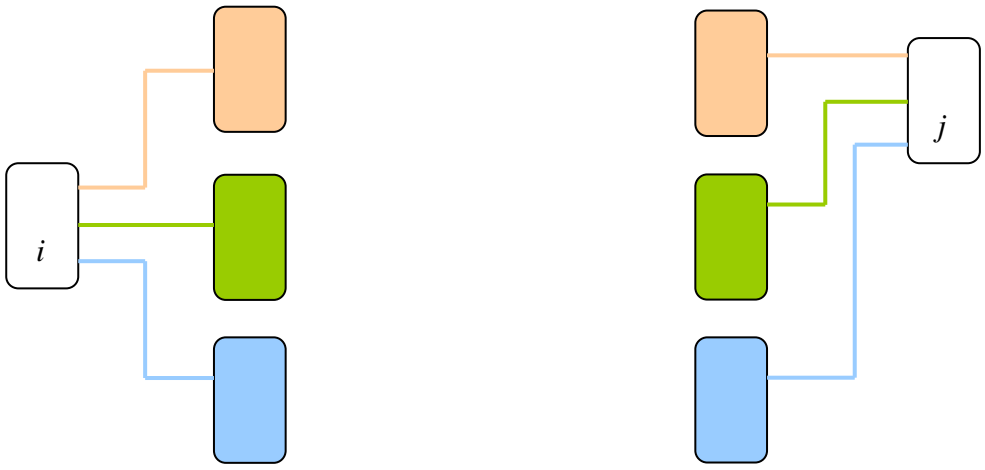
- Associamo ad ogni switch centrale uno degli m colori usati nell'EC minimo
- Per ogni connessione $(i,j) \in F$ costruiamo la rotta (i,c,j) con c il colore dell'arco (I_i,O_j) di G_F.

Esempio



$(i,j) \in F$ rotta $(i,arancio,j)$

Per costruzione R realizza F quindi non rimane che verificare che R sia anche compatibile con la rete. Poiché da ogni switch d'ingresso (uscita) partono (arrivano) rotte tutte di colore diverso, ciò significa che ognuna attraversa uno switch intermedio diverso, quindi la configurazione è compatibile.



generico switch d'ingresso i generico switch d'uscita j

$C(n,m,r)$ *è riconfigurabile* $\Rightarrow \chi(G_F) \leq m \ \forall F \in \Phi(n,r)$

Si procede come nel caso precedente. □

Da questa dimostrazione segue immediatamente la seguente proprietà:

Proprietà 2.4 *Algoritmo di routing*

Sia A_{EC} un algoritmo che determina edge coloring con *m* colori per un multigrafo bipartito, allora A_{EC} è anche un algoritmo di routing per una C(*n,m,r*) riconfigurabile.

Anche il problema della riconfigurabilità può essere visto in termini di colorazione di archi di un multigrafo bipartito:

Proprietà 2.5 *Problema di riconfigurabilità*

Siano fissati due interi positivi *n* ed *r*. Indichiamo con $\chi_{\min}(n,r)$ il numero minimo di colori affinché, per ogni frame *F* di $\Phi(n,r)$, il multigrafo G_F ammetta un edge color con $\chi_{\min}(n,r)$ colori, cioè in simboli $\chi_{\min}(n,r) = \max_{F\in\Phi(n,r)} \chi(G_F)$.

Vale la relazione:

$$\chi_{\min}(n,r) = \mathrm{M}(n,r).$$

Ricordiamo che M(*n,r*) è il minimo numero di switch centrali in una C(*n*,*m*,*r*) affinché la rete sia riconfigurabile.

Dimostrazione

Per definizione della funzione *max* segue che esiste $\overline{F} \in \Phi(n,r)$ tale che $\chi(G_{\overline{F}}) = \chi_{\min}(n,r)$ e $\chi(G_F) \le \chi(G_{\overline{F}}), \forall F \in \Phi(n,r)$ da cui, per la proprietà 2.3, segue che C(*n,m,r*) è riconfigurabile se e solo se $m \ge \chi(G_{\overline{F}})$ e dunque $\chi(G_{\overline{F}}) = \mathrm{M}(n,r)$, da cui segue la tesi $\chi_{\min}(n,r) = \mathrm{M}(n,r)$. □

2.3.2 Riconfigurabilità e matching

Consideriamo un edge coloring con *m* colori di un multigrafo bipartito *G*=(*I*,*O*,*E*). Come abbiamo già visto nel paragrafo 2.2, tutti gli archi di un certo colore formano un matching e gli *m* matching corrispondenti ai colori usati nella colorazione di *G*, costituiscono una partizione di *E*. Questo ci consente di esprimere le tre proprietà del paragrafo 2.3.1 in termini di matching ovvero di

poter mettere in relazione le problematiche riguardanti la riconfigurabilità di una rete di Clos con la teoria dei matching:

Proprietà 2.6 *Rete di Clos riconfigurabile*
Una C(*n,m,r*) è riconfigurabile se e solo se, per ogni frame *F*, il multinsieme *E* del multigrafo delle connessioni G_F =(*I,O,E*) ammette una partizione con *m* matching.

Proprietà 2.7 *Algoritmo di Routing*
Sia C(*n,m,r*) una rete di Clos riconfigurabile e sia G_F =(*I,O,E*) il grafo delle connessioni relativo ad un generico frame *F*.
Un algoritmo A_M che determina una partizione di *E* in *m* matching, è anche un algoritmo di routing per la rete C(*n,m,r*).

Proprietà 2.8 *Problema di riconfigurabilità*
Siano fissati due interi positivi *n* ed *r*.
Indichiamo con $\mu_{\min}(n,r)$ il numero minimo di matching con cui si può partizionare ogni multigrafo delle connessioni G_F con $F \in \Phi(n,r)$, in simboli: $\mu_{\min}(n,r) = \max_{F \in \Phi(n,r)} \alpha(G)$.

Vale la relazione:

$$\mu_{\min}(n,r) = \mathrm{M}(n,r)$$

(ricordiamo che $\alpha(G)$ è la minima cardinalità di una partizione di *E* in matching).
Le dimostrazioni non sono necessarie poiché si ricavano direttamente dalle tre proprietà del paragrafo precedente e dalla proprietà 2.1.

2.3.3 Riconfigurabilità ed algebra lineare

L'ultimo modello matematico che prendiamo in considerazione è quello che esprime le problematiche legate alle reti di Clos riconfigurabili in termini di

algebra lineare. Le tre seguenti proprietà sono l'equivalente di quelle viste nei due paragrafi precedenti e sintetizzano il modello adottato.

Proprietà 2.9 *Rete di Clos riconfigurabile*

Una C(*n,m,r*) è riconfigurabile se e solo se, per ogni frame $F \in \Phi(n,r)$, la matrice di traffico T_F ammette una decomposizione di lunghezza *m* tale che:

$$T_F = \sum_{i=1}^{m} E_i \quad (\forall i = 1,...,m \,.E_i \text{ è una matrice elementare})$$

Proprietà 2.10 *Algoritmo di Routing*

Sia C(*n,m,r*) una rete di Clos riconfigurabile e sia T_F la matrice di traffico relativa ad un frame $F \in \Phi(n,r)$.

Un algoritmo A_{DEC} che, $\forall F \in \Phi(n,r)$ determina una decomposizione di T_F con lunghezza al più *m*, è anche un algoritmo di routing per la rete C(*n,m,r*).

Proprietà 2.11 *Problema di riconfigurabilità*

Siano fissati due interi positivi *n* ed *r*. Indichiamo con $\rho_{\min}(n,r)$ la lunghezza minima con cui si può decomporre ogni matrice T_F, con $F \in \Phi(n,r)$. In simboli, $\rho_{\min}(n,r) = \max_{F \in \Phi(n,r)} \rho(T_F)$.

Vale la relazione:

$$\rho_{\min}(n,r) = \mathrm{M}(n,r).$$

Ricordiamo che $\rho(T_F)$ è la minima lunghezza che può avere una decomposizione di T_F.

Diamo la dimostrazione solo della prima proprietà, poiché da questa si ottengono immediatamente le altre due.

Dimostrazione proprietà 2.9

Dimostriamo che: $T_F = \sum_{i=1}^{m} E_i \Rightarrow$ C(*n*,*m*,*r*) *è riconfigurabile*

Sia *F* un qualsiasi frame di $\Phi(n,r)$ e T_F la matrice di traffico associata *F*.

Vediamo come dalla decomposizione $T_F = \sum_{i=1}^{m} E_i$ si possa risalire ad una configurazione *R* che realizza *F* e che sia compatibile con la rete C(*n*,*m*,*r*).

Associamo ad ogni switch centrale *x* della rete, la matrice elementare E_x. Costruiamo *R* come segue:

- Sia e_{ij}^x il generico elemento di E_x. Se $e_{ij}^x = 1$ allora aggiungiamo ad *R* il cammino (*i*,*x*,*j*). Ripetiamo il procedimento per ognuna delle *m* matrici della decomposizione.

Verifichiamo che la *R* così ottenuta è compatibile con C(*n*,*m*,*r*) e realizza il frame *F* di $\Phi(n,r)$.

Supponiamo per assurdo che *R* non sia compatibile, allora esisterebbero almeno due cammini (*i*,*x*,*j*) e (*u*,*y*,*v*) che "attraversano" uno stesso link. Questo può accadere in uno dei seguenti tre casi:

1. $i=u,\ x=y,\ j=v$
2. $i=u,\ x=y,\ j \neq v$
3. $i \neq u,\ x=y,\ j=v$

Il primo caso è impossibile perché, per costruzione nel caso CS, *R* è un insieme, quindi è costituito solo da cammini diversi l'uno dall'altro.

Il secondo caso comporterebbe l'esistenza di una matrice E_x che sulla riga *i* ha due elementi uguali ad uno: $e_{ij}^x = e_{iv}^x = 1$. Questo è impossibile per definizione di matrice elementare. Analogamente si dimostra che la terza situazione non può presentarsi.

Ora verifichiamo che *R* realizza *F*.

Per definizione di matrice di traffico, l'elemento t_{ij} di T_F rappresenta il numero di richieste del "tipo" (*i*,*j*) appartenenti ad *F*. Nella decomposizione di T_F devono quindi esistere esattamente t_{ij} matrici elementari con l'elemento di coordinate (*i*,*j*) uguale ad uno.

Esempio:

$$T_F = \begin{pmatrix} .. & .. & .. \\ .. & .. & 3 \\ .. & .. & .. \end{pmatrix} = ... + \begin{pmatrix} .. & .. & 0 \\ 0 & 0 & 1 \\ .. & .. & 0 \end{pmatrix} + \begin{pmatrix} .. & .. & 0 \\ 0 & 0 & 1 \\ .. & .. & 0 \end{pmatrix} + \begin{pmatrix} .. & .. & 0 \\ 0 & 0 & 1 \\ .. & .. & 0 \end{pmatrix} + ...$$

Per costruzione, in R ci sono t_{ij} rotte che realizzano le t_{ij} connessioni del tipo (i,j).

In conclusione R è compatibile con la rete C(n,m,r) e realizza il frame F. Inoltre, avendo scelto F a caso, segue che C(n,m,r) è riconfigurabile.

Dimostriamo che, C(n,m,r) *è riconfigurabile* $\Rightarrow \forall F \in \Phi(n,r).\ \exists (E_1,...,E_m)$

$t.c. T_F = \sum_{i=1}^{m} E_i$

Sia R una configurazione compatibile con C(n,m,r) e che realizza un frame F appartenente a $\Phi(n,r)$.

Vogliamo dimostrare che esistono m matrici elementari tali che

$$T_F = \sum_{i=1}^{m} E_i \quad \text{(con } T_F \text{ matrice di traffico associata ad } F\text{)}$$

Sia $R_x = \{(i,x,j) \in R \mid i,j=1,..,r\ \}$ l'insieme di tutti i cammini passanti per lo switch centrale x. Costruiamo la matrice E_x nel modo seguente:

$$e_{ij}^{x} = \begin{cases} 1 \text{ se } (i,x,j) \in R_x \\ 0 \text{ altrimenti} \end{cases} \quad (1)$$

Essendo R compatibile per ipotesi, segue che E_x è elementare. Infatti, in R_x non possono esistere più cammini con stesso switch d'ingresso e/o uscita.

Ripetendo il procedimento per ogni switch centrale, otteniamo m matrici elementari $\{E_1,.....,E_m\}$.

Non rimane che verificare che la sequenza $S=\{E_1,.....,E_m\}$ costituisce una decomposizione di T_F.

Consideriamo la sommatoria $\sum_{x=1}^{m} e_{ij}^{x}$. Per la (1) segue che,

$$\sum_{x=1}^{m} e_{ij}^{x} = |R_{ij}| \quad \forall i,j = 1,..,r \qquad (2)$$

dove R_{ij} è l'insieme dei cammini in R che vanno dallo switch d'ingresso i a quello di uscita j. Sia t_{ij} il generico elemento di T_F, per ipotesi R realizza il frame F, dunque $|R_{ij}|= t_{ij} \ \forall i, j = 1,..,r$. Dalla (2) segue,

$$\sum_{x=1}^{m} e_{ij}^{x} = |R_{ij}| = t_{ij} \quad \forall i, j = 1,..,r$$

quindi $\{E_1,\ldots\ldots,E_m\}$ costituiscono una decomposizione di T_F. □

2.4 Problema di riconfigurabilità

La soluzione al problema di riconfigurabilità di una rete di Clos C(n,m,r) è attribuita in [CSHwan83] a Slepian [CSSlep52] e Duguid [CSDugu59] che negli anni cinquanta hanno dimostrato il teorema:

Teorema 2.1

Una C(n,m,r) è riconfigurabile se e solo se $m \geq n$, vale a dire M(n,r)=n.

La dimostrazione originale è stata fatta per induzione su n, però grazie alla proprietà 2.5 possiamo darne una alternativa molto semplice.

Dimostrazione

Dalla proprietà 2.5, per dimostrare che M(n,r)=n, basta dimostrare che $n= \chi_{\min}(n,r)$.

Per la proprietà 1.2 $\forall F \in \Phi(n,r)$, $\deg(G_F) \leq n$.

Per il lemma di Konig segue:

$\forall F \in \Phi(n,r)$, $\chi(G_F) \leq n$

Essendo facile costruire un frame $\overline{F} \in \Phi(n,r)$ per cui $\deg(G_{\overline{F}}) = n$, la tesi è dimostrata:

$\chi_{\min}(n,r) = \max_{F \in \Phi(n,r)} \chi(G_F) = n$ □

2.5 Algoritmi di routing

Nei tre paragrafi che seguono vediamo come le proprietà 2.4, 2.7. e 2.10 si possono utilizzare per costruire algoritmi di routing per reti di Clos riconfigurabili. Nel paragrafo 2.5.4 invece presenteremo un algoritmo che non è

direttamente “inquadrabile” in nessuno dei tre modelli matematici discussi in precedenza.

2.5.1 Algoritmi di routing basati su edge coloring

Descriviamo ora due modi di ottenere un edge coloring di un multigrafo bipartito. Il primo utilizza una tecnica chiamata *cammini alternati* (o *aumentati*) l'altro impiega una strategia di divide et impera ed è chiamato *colorazione euleriana*

2.5.1.1 Cammini alternati

Un approccio molto noto al problema della determinazione di una colorazione minima di un multigrafo bipartito, è la tecnica dei *cammini alternati*. Tale tecnica risale al 1962 [ECOre62] e, come vedremo nel capitolo 3 è stata utilizzata anche nel primo algoritmo di routing per reti di Clos riconfigurabili in ambiente multirate proposto in [MRMeTu89] nel 1989.

Prima di vedere nel dettaglio l'algoritmo, diamo qualche definizione.
Un *cammino* *P* in un grafo *G*=(*V*,*E*), è una sequenza di archi adiacenti $(v_1, v_2)(v_2, v_3).....(v_{k-2}, v_{k-1})(v_{k-1}, v_k)$ dove i nodi v_1 e v_k sono detti *estremità* del cammino *P*. Se $v_1 \neq v_k$, allora *P* è detto *cammino aperto*, altrimenti *cammino chiuso*.

Sia *G*=(*V,E*) un grafo bipartito *parzialmente colorato* e *C* l'insieme dei colori da utilizzare. Diciamo che un vertice $v \in V$ è $\alpha - missing$ (con $\alpha \in C$) se nessun arco incidente su *v* è colorato con α. Sia (*u,v*) un arco di *E* non colorato, diremo che (*u,v*) è $\alpha\beta - colorabile$ se *u* è $\alpha - missing$ e v è $\beta - missing$.

Sia (*u,v*) $\alpha\beta - colorabile$,definiamo *massimo cammino alternato per* (*u,v*) $\left(\text{indicato con } P_{\alpha\beta}(u,v)\right)$ un cammino di *G* tale che:

- inizia in *u* o *v*
- è composto di archi colorati alternativamente con α e β.
- abbia lunghezza massima

Nel caso particolare in cui $\alpha = \beta$ allora il cammino ha lunghezza nulla e significa che sia in *u* sia in *v* non ci sono archi incidenti di colore α e che l'arco (*u,v*) può quindi essere colorato con α.

Vediamo l'algoritmo:

```
Algoritmo Edge_Coloring_Cammini_Alternati(G)

begin

   1. D=deg(G);
   2. C={1,...,D};
   3. for each (u,v)∈E non-colorato do
         colora_arco((u,v),C,G);
end;
```

La procedura ***colora_arco,*** dato un arco (*u,v*) $\alpha\beta - colorabile$ ed un multigrafo *G* parzialmente colorato con colori *C*, colora (*u,v*) con α o β ottenendo una nuova colorazione parziale del grafo *G*:

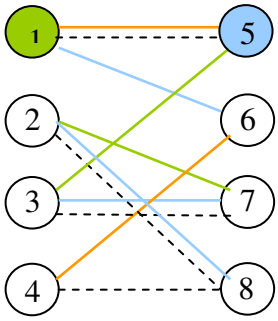
```
Perocedura colora_arco((u,v),C,G)

begin

   1. (α,β)= due colori t.c. (u,v) sia αβ-colorabile;
   2. if α=β then colora (u,v) con α;
               else
                begin
                     2.1  determina P_αβ(u,v);
                     2.2  Inverti i colori di P_αβ(u,v);
                     2.3  if P_αβ(u,v) inizia da v
                               then colora (u,v) con α;
                               else colora (u,v) con β;
                end;
end;
```

Vediamo ora un esempio per chiarire come funziona la procedura ***colora_arco***.

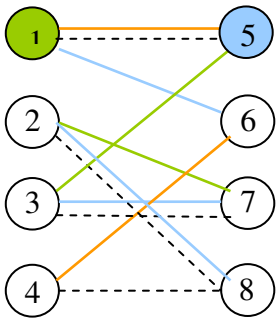
Consideriamo il multigarfo *G* parzialmente colorato rappresentato nella figura seguente.



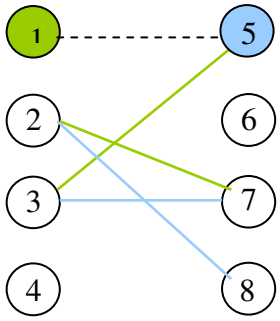
Il grado di *G* è 3 ed i colori utilizzati sono: celeste, verde, arancio. Applichiamo la procedura all'arco (1,5).

Abbiamo colorato i nodi 1 e 5 con i colori che ancora possono essere utilizzati per la colorazione dell'arco (1,5) ed indicato con una linea tratteggiata gli archi ancora da colorare.

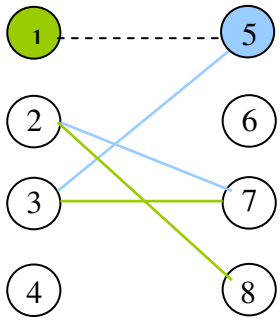
Passo 1: α=verde, β=celeste



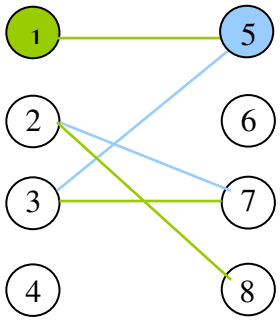
Passo 2.1: determinazione del massimo cammino alternato



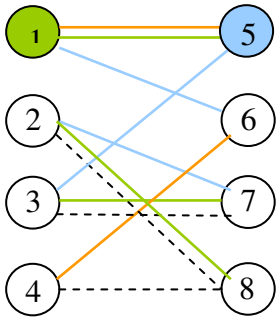
Passo 2.2: inversione dei colori del cammino



Passo 2.2: colora (1,5) con α=verde



Al termine della procedura abbiamo la seguente colorazione parziale di *G*:



Complessità

In [ECGaKa82] si dimostra che con un'opportuna struttura dati, l'algoritmo ha complessità $O(|V||E|)$ in tempo e $O(|V|+|E|)$ in spazio. In particolare è dimostrato che la procedura **`colora_arco`** ha complessità $O(|V|)$ *-tempo.*

2.5.1.2 Euler color

Nel 1976 Gabow [ECGabo76] propone un algoritmo di colorazione minima per multigrafi bipartiti basato su una tecnica di divide et impera. Negli anni successivi l'algoritmo è stato migliorato in termini di complessità da altri autori mantenendone inalterata la struttura ([ECCoHo82],[ECSchr98][ECCole01]). Nel prossimo paragrafo vediamo l'algoritmo di Gabow originale che presenta una complessità di $O\left(\sqrt{|V|}\,|E|\log(D)\right)$, mentre nel paragrafo 2.5.1.4 illustriamo una variante (la più recente che siamo riusciti a trovare) che ha complessità $O(|E|\log D)$ ed è stata presentata in [ECCole01] nel 2001.

Definizioni e notazioni

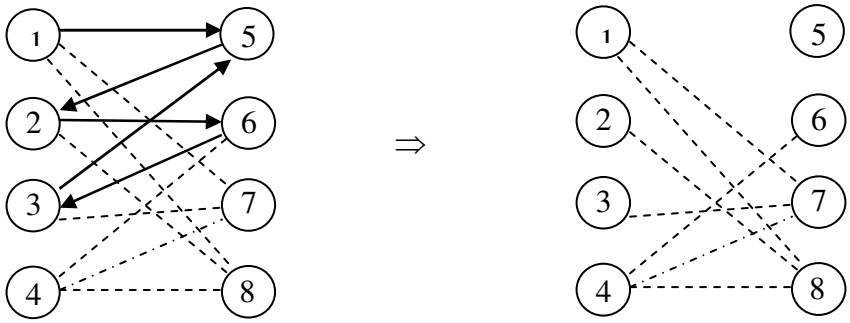
Consideriamo un multigrafo bipartito $G=(I,O,E)$ di grado $D=deg(G)$.

Una *partizione euleriana P*, è una partizione di *E* in cammini chiusi e aperti, in modo che ogni vertice di grado dispari sia al termine di un cammino aperto ed ogni vertice di grado pari, sia alla fine di un cammino chiuso.

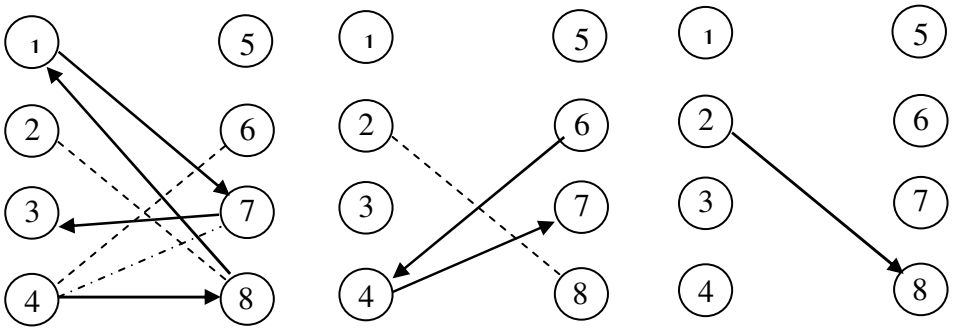
Si può dimostrare che ogni grafo ammette almeno una partizione euleriana. In particolare per un grafo bipartito in [ECGabo76] sì da una procedura per determinarne una in $O(|E|)$. Diamo una breve descrizione di tale procedura:

1. Si sceglie un vertice *u* di grado dispari (se non c'è, uno qualsiasi con grado non nullo). Se ci sono solo nodi con grado zero, allora la procedura è finita.
2. Si attraversa un arco da *u* a al vertice *v* e si elimina l'arco (*u*,*v*).
3. Si ripete il passo 2 fino a quando ci troviamo in un nodo di grado 0. Gli archi che abbiamo eliminato costituiscono un cammino.
4. Si ricomincia dal passo 1.

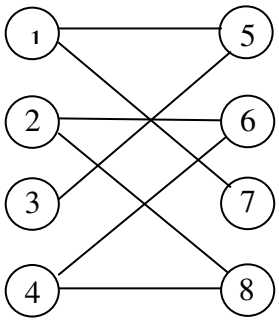
Vediamo un esempio:
Il primo cammino è determinato a partire dal nodo 1.



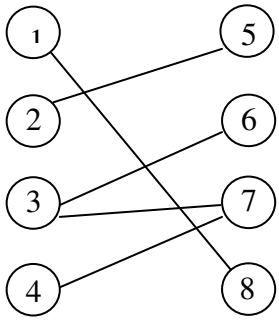
In figura sono rappresentati il primo cammino aperto determinato ed il grafo "rimanente" privato di tale cammino. Abbiamo indicato con una linea tratteggiata gli archi ancora da percorrere, mentre la freccia indica il verso che abbiamo seguito per costruire il cammino. Riportiamo di seguito tutti i cammini che sono costruiti:



Sia P una partizione euleriana di E, uno *split-euleriano* di $G=(I,O,E)$, è una coppia di multigrafi bipartiti $G_1 = (I, O, E_1)$ e $G_2 = (I, O, E_2)$ dove $E_1 \text{ed } E_2$ costituiscono una partizione di E costruita percorrendo ogni cammino di P e ponendo, in modo alternato in E_1 ed E_2 ogni arco attraversato.
Nella figura seguente riportiamo lo split euleriano relativo all'esempio precedente.



deg (G_1)=2



deg (G_2)=2

Si può dimostrare che vale la seguente proprietà:

Proprietà 2.12 [ECGabo76]
Sia u un vertice del multigrafo $G=(I,O,E)$ indichiamo con $\deg_1(u)$ e $\deg_2(u)$ il grado che u assume in G_1 e G_2 rispettivamente. Valgono le relazioni:

- $\deg_1(u), \deg_2(u) \leq \lceil \deg(u)/2 \rceil$
- Se $\deg(u)$ è pari allora $\deg_1(u)=\deg_2(u)=\deg(u)/2$
- Se $\deg(u)$ è dispari allora $|\deg_1(u)-\deg_2(u)| =1$

Dalla proprietà segue immediatamente che se $\deg(G)$ è pari allora G_1 e G_2, hanno ognuno grado $\deg(G)/2$.

2.5.1.3 Algoritmo di Gabow

Presentiamo prima un algoritmo che è in grado di determinare una colorazione minima di G solo se il grado di G è del tipo $\deg(G)=2^k$ con k intero positivo. Successivamente vedremo come tale algoritmo potrà essere corretto per ottenere una colorazione minima per ogni grafo bipartito in $O\left(\sqrt{|V|}\,|E|\log(D)\right)$ con $D=deg(G)$.

La tecnica usata è quella del *divide et impera*: il multigrafo G viene diviso in due sottografi G_1 e G_2 utilizzando uno split-euleriano. A questo punto viene applicata ricorsivamente sui due sottografi la procedura di colorazione.
Riportiamo l' algoritmo :

```
Algoritmo colorazione_euleriana(G)
begin

   1. D=deg(G);
                if D>1 then
                          begin
                            2.1 (G1,G2)=split_euleriano(G);
                            2.2 colorazione_euleriana(G1);
                            2.3 colorazione_euleriana(G2);
                          end;
                        else colora archi di G
                             utilizzando nuovo colore;
end;
```

Vediamo ora il motivo per cui l'algoritmo funziona solo per grafi con $\deg(G)=2^k$ dove k è un intero positivo. Ricordiamo che, essendo G bipartito, per il lemma di Konig il numero di colori utilizzati in una colorazione minima coincide con $\deg(G)$.
Supponiamo che G abbia grado dispari. Per la proprietà 2.12 la decomposizione ottenuta al passo 2.1 con uno split-euleriano produce i multigrafi G_1 e G_2 ognuno con grado non maggiore di $\lceil \deg(G)/2 \rceil$. Dunque anche se i passi 2.2 e 2.3 fornissero per i due sottografi una colorazione minima, al termine dell' algoritmo

avremmo una colorazione di G con $\deg(G_1)+\deg(G_2) \leq 1+ \deg(G)$ colori che in generale non è minima.

Se invece $\deg(G)$ è pari, l' inconveniente appena visto non si presenta poichè $\lceil \deg(G)/2 \rceil = \deg(G)/2$. Affinchè quindi l'algoritmo fornisca una colorazione minima, ad ogni passo della ricorsione il grado del grafo deve essere pari, cioè è necessario che $\deg(G)=2^k$.

L'algoritmo ***colorazione-euleriana*** può essere modificato in modo da ottenere una colorazione minima per qualsiasi multigrafo bipartito G. La strategia di tale modifica è di diminuire di uno il grado di G nel caso sia dispari.

```
Algoritmo Gabow(G)

begin
   1. D=deg(G);
   2. if D>1 then
                 begin
                   2.1 if dispari(D) then
                           begin
                              2.1.1 M=Max_matching(G);
                              2.1.2 colora M con nuovo
                                    colore;
                              2.1.3 E=E-M;
                            end;
                    2.2   (G1,G2)=split_euleriano(G);
                    2.3   colorazione_euleriana_minima(G1);
                    2.4   colorazione_euleriana_minima(G2);
                   end;
              else colora G con nuovo colore;

end;
```

Dove, il passo ***Max_matching***(G) calcola un matching M di G che copre tutti i vertici di grado massimo $deg(G)$, ovvero determina un matching Ω-containing.

Verifichiamo cosa accade se il multigrafo G ha grado $D=deg(G)$ dispari.

Eliminando da G tutti gli archi di M (passo 2.1.3) il grafo che ne risulta ha grado D-1 e quindi è pari. Il passo 2.2 calcola uno split euleriano di G producendo i multigrafi G_1 e G_2 ognuno con grado $\lceil (D-1)/2 \rceil$=(D-1)/2. Con quest'accorgimento è stato dunque eliminato il "difetto" dell'algoritmo precedente.

Complessità dell'algoritmo di Gabow

Dimostriamo che l'algoritmo di Gabow ha complessità $O\left(\sqrt{|V|}\,|E|\log(D)\right)$.

Facciamo l'ipotesi che ogni vertice di G=(V,E) abbia grado non nullo, nel caso questo non fosse vero, prima di applicare l'algoritmo a G, basterà eliminare i nodi di grado zero con un costo O(|V|).

Da quest'ipotesi segue che $|V| \le 2|E|$, quindi |V|=O(|E|).

Indichiamo con $T(|E|,D)$ il numero di passi eseguiti dall'algoritmo. Se D=1, colorare gli archi costa O(|E|), quindi $T(|E|,D) \le c|E|$, con c costante positiva opportuna. Se D>1 ed è dispari, viene calcolato il matching (passo 2.1.1) con un costo di $O\left(\sqrt{|V|}(|E|+|V|)\right)$ che per l'ipotesi iniziale equivale a $O\left(\sqrt{|V|}\,|E|\right)$.

L'operazione di *split* di G (passo 2.2) ha complessità O(|E|) e produce i due multigrafi G_1 e G_2 tali che $|E_1|+|E_2|=E$ e $V=V_1=V_2$ e $\deg(G_1)=\deg(G_2)=\deg(G)/2$.

Le due chiamate ricorsive (passi 2.3 e 2.4) effettuate su G_1 e G_2 costano rispettivamente $T(|E_1|,\lfloor D/2 \rfloor)$ e $T(|E_2|,\lfloor D/2 \rfloor)$. Il numero totale dei passi $T(|E|,D)$ eseguiti dall'algoritmo può essere maggiorato nel modo seguente:

$$T(|E|,D) \le \begin{cases} k|E|+q\sqrt{|V|}\,|E|+T(|E_1|,\lfloor D/2 \rfloor)+T(|E_2|,\lfloor D/2 \rfloor) & \text{se } D>1 \text{ e } D \text{ dispari} \\ k|E|+T(|E_1|,D/2)+T(|E_2|,D/2) & \text{se } D>1 \text{ e } D \text{ pari} \\ c|E| & \text{se } D=1 \end{cases}$$

dove $k|E|$ e $q\sqrt{|V|}\,|E|$ sono rispettivamente le complessità delle procedure ***split_euleriano*** e ***Max_matching*** (con k,q costanti positive opportune)

Si può verificare facilmente che dalla disuguaglianza precedente segue che $T(|E|,D) \leq s\sqrt{|V|}\,|E|\log D$ con s costante positiva opportuna, quindi l'algoritmo ha complessità $O\left(\sqrt{|V|}\,|E|\log(D)\right)$.

2.5.1.4 Algoritmo di R.Cole, K.Host e S.Schirra

In [ECCole01] R.Cole, K.Host e S.Schirra forniscono un algoritmo di complessità $O(|E|logD)$ di colorazione minima per multigrafi bipartiti. L'algoritmo si basa sulla tecnica di divide et impera utilizzata da Gabow e su una procedura di complessità $O(|E|)$ per la determinazione di un matching perfetto.

Gli autori dimostrano il seguente teorema:

Teorema 2.2
Sia $G=(I,O,E)$ un multigrafo bipartito regolare. Un matching perfetto in G può essere determinato in tempo $O(|E|)$.

Nella dimostrazione si fornisce la descrizione della procedura, che chiameremo *Matching_Perfetto*, in grado di ottenere tale risultato.
Applicando il teorema e la tecnica di colorazione di Gabow segue il corollario:

Corollario 2.2
Una colorazione minima di un multigrafo $G=(I,O,E)$ bipartito k-regolare, può essere determinata in tempo $O(|E|\log k)$.

Dimostrazione
Sia $G=(I,O,E)$ k-regolare. Osserviamo che se k è pari, per la proprietà 2.12, uno split euleriano di G è una coppia di multigrafi bipartiti $k/2$-regolari. Nell'algoritmo di Gabow si può dunque sostituire la procedura di calcolo del matching Ω-containing (*Max_matching*) con quella per il calcolo del matching perfetto del teorema 2.2. Il costo dell'algoritmo di edge coloring segue immediatamente da quello visto per l'algoritmo di Gabow e si ottiene risolvendo la seguente disequazione ricorsiva:

$$T(|E|,k) \le \begin{cases} k|E| + |E| + T(|E_1|,\lfloor k/2 \rfloor) + T(|E_2|,\lfloor k/2 \rfloor) & \text{se k>1 e k dispari} \\ k|E| + T(|E_1|,k/2) + T(|E_2|,k/2) & \text{se k>1 e k pari} \\ c|E| & \text{se k=1} \end{cases}$$

Si può verificare che tale disequazione ha soluzione $T(|E|,k) \le s|E|\log k$, con s costante positiva opportuna, ovvero l'algoritmo ha complessità O(|*E*|log*k*). □

Per applicare l'algoritmo di colorazione a qualsiasi *G*=(*I*,*O*,*E*) è sufficiente rendere *G* *D*-regolare. Schirver in [ECSchr98] descrive un modo per farlo in *O(|E|)* passi. La trasformazione di *G* ottenuta è tale che se una colorazione è minima per il multigrafo modificato lo è anche per quello originale. Segue il corollario:

Corollario 2.3
Una colorazione minima di un multigrafo *G*=(*I,O,E*) bipartito di grado *D*, può essere determinata in tempo O(|*E*|log*D*).

Vediamo l'algoritmo nel complesso:

```
Algoritmo Cole_Ost_Schirra(G)
begin
   1. regolarizza(G);
   2. colorazione_euleriana(G);
end;
```

dove `regolarizza` rende il multigrafo *D*-regolare mentre la procedura `colorazione_euleriana` non è altro che l'algoritmo di Gabow in cui il matching Ω–containing è calcolato con la nuova procedura *Matching_Perfetto* di complessità O(|*E*|) presentata in [ECCole01].

Di seguito riportiamo la procedura `colorazione_euleriana` e la descrizione di `regolarizza` data da Schrijver in [ECSchr98].

```
Procedura colorazione_euleriana(G)
begin
 3.  D=deg(G);
 4.  if D>1 then
          begin
```

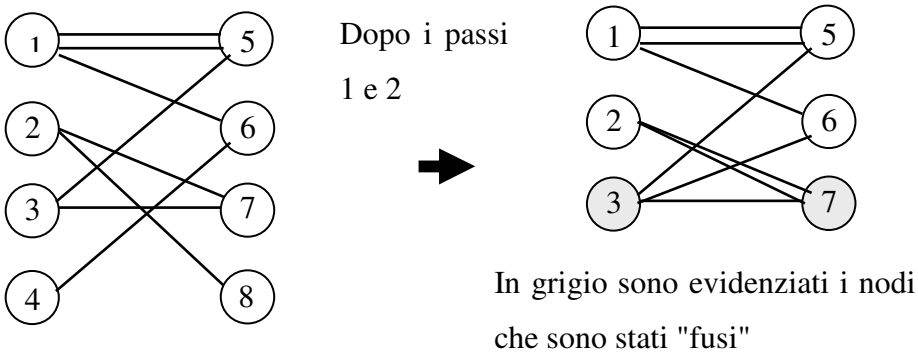
```
      2.1 if dispari(D) then
            begin
              4.1.1 M=Matching_Perfetto(G);
              4.1.2 colora M con nuovo colore;
              4.1.3 E=E-M;
            end;
         4.2   (G1,G2)=split_euleriano(G);
         4.3   colorazione_euleriana_minima(G1);
         4.4   colorazione_euleriana_minima(G2);
        end;
      else colora G con nuovo colore;

end;
```

Diamo la descrizione di `regolarizza`:

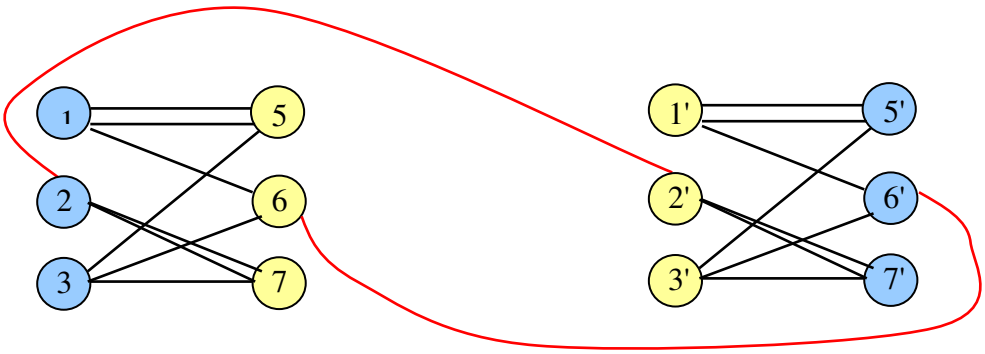
1. Sì "fondono" i nodi di I in modo che ognuno abbia grado strettamente maggiore di $\lfloor \deg(G)/2 \rfloor$ e minore o uguale a $\deg(G)$.
2. Si effettua la stessa operazione per i nodi di O. A questo punto al più esistono due vertici di G con grado minore o uguale a $\lfloor \deg(G)/2 \rfloor$.
3. Si fa una copia di G che indichiamo con $G'=(I',O',E')$.
4. Si costruisce il multigrafo bipartito $H=(I \cup O', O \cup I', E \cup E')=(I_H, O_H, E_H)$.
5. Sia u un vertice di G di grado $\deg(u)$ e u' la sua copia in G'. Aggiungiamo $(\deg(G)-\deg(u))$ archi del tipo (u,u'). Eseguiamo quest'operazione per ogni vertice di G. Ora tutti i nodi di H hanno grado $\deg(G)$ e dunque H è regolare.

Vediamo un esempio:



In grigio sono evidenziati i nodi che sono stati "fusi"

Dopo i passi 3, 4 e 5 otteniamo il multigrafo bipartito H (nella figura seguente i nodi di I_H sono stati colorati in azzurro, mentre quelli di O_H in giallo).



Osserviamo che una colorazione minima di H ne induce una sul multigrafo originale G: basta togliere da H gli archi che sono stati aggiunti al passo 5 e "ricordare" la corrispondenza tra gli archi originali e quelli successivi alla fusione dei vertici.

2.5.2 Algoritmi di routing basati sul calcolo di matching

Sia C(n,m,r) una rete di Clos, riconfigurabile. La proprietà 2.7 ci dice che come algoritmo di routing, se ne può utilizzare uno che determina la decomposizione di un multigrafo bipartito in m matching. Nel seguito presentiamo due algoritmi proposti da F. Hawang che sfruttano tale proprietà [CSHwan83].
Il primo algoritmo è basato sul calcolo di un matching che copre tutti i vertici di grado massimo (Ω-containing), l'altro utilizza il calcolo del matching massimo

2.5.2.1 Algoritmo Routing Ω-containing

Dal lemma 2.2 segue il semplice algoritmo:

```
Algoritmo Routing_Ω-Containing(G)
//M[] è un vettore che memorizza i matching calcolati
//dall'algoritmo
begin
1. i=1;
2. while (deg(G)>0) do
 begin
  2.1. M[i]=Calcola_Ω-containing(G);
  2.2. E=E-M;//il grado di G diminuisce di 1
  2.3. i=i+1;
 end;
end;
```

Dove la procedura ***Calcola_Ω-containing***(*G*) determina un matching Ω-containing sul grafo *G*.
Al termine dell'algoritmo il vettore *M[]* conterrà *deg(G)* matching: infatti ad ogni iterazione (passo 2.2) il grado di *G* diminuisce di uno.

La complessità dell'algoritmo è il prodotto tra quella della procedura ***Calcola_Ω-containing*** e il grado di *G* (cioè il numero d'iterazioni).
In [CSHwan83] per il calcolo del matching Ω-containing è utilizzato un algoritmo di Cole ed Hopcroft [ECCoHo82] che ha complessità $O(|E|log|V|)$ dunque ***Routing_Ω-Containing*** è $O(D|E|log|V|)$.

2.5.2.2 Algoritmo Routing matching massimo

Consideriamo il caso in cui *G* sia *D*-regolare. Dal lemma 2.2 ne consegue che in *G* esiste sempre un matching massimo. Infatti, essendo il multigrafo *D*-regolare, ogni nodo ha grado *D* e quindi per il lemma 2.2 esiste un matching *M* che copre tutti i vertici di *V*, dunque *M* è massimo.

Vediamo l'algoritmo che determina una decomposizione di *G* in *deg(G)* matching:

```
Algoritmo Routing_Matching_Massimo(G)
//M[] è un vettore che memorizza i matching calcolati
//dall'algoritmo
begin

1. regolarizza(G);
```

```
2. i=1;
3. while (deg(G)>0) do
 begin
  3.1. M[i]=Calcola_Matching_Massimo(G);
  3.2. E=E-M;//il grado di G diminuisce di 1
  3.3. i=i+1;
 end;
end;
```

dove la procedura ***regolarizza*** serve a rendere il multigrafo *D*-regolare.

Diamo una breve descrizione di come funziona l'algoritmo.

Sia *D* il grado del multigrafo iniziale. Alla prima iterazione, poiché *G* è regolare e *M*[1] è massimo, al passo 3.2 ogni nodo viene privato di un arco, quindi il grafo *G*=(*V*,*E*-*M*[1]) è ancora regolare ed ha grado diminuito di uno rispetto a quello iniziale. Il ciclo dunque è ripetuto *D* volte ed al suo termine, nel vettore *M*[] abbiamo ottenuto i *D* matching che volevamo.

Complessità dell'algoritmo matching massimo

Per quanto riguarda il passo per rendere regolare *G*=(*I*,*O*,*E*), Hwang non dice come realizzarlo. In ogni caso possiamo utilizzare quello proposto da Schrijver visto nel paragrafo 2.5.1.4 che ha complessità $O(|E|)$ [ECSchr98].

Dalla proprietà 2.2 sappiamo che un matching massimo M_{Max} per un multigrafo *G*=(*I*,*O*,*E*), può essere determinato sul grafo G_S=(*I*,*O*,E_S) sotteso a *G*. L'algoritmo per il calcolo di M_{Max}, utilizzato in [CSHwan83] è quello di Hopcroft e Karp (1973) [ECHoKa73] che ha complessità in tempo $O(|E_S||V|^{0,5})$. Poiché in un grafo, gli archi sono al più il quadrato del numero dei suoi vertici, segue che $O(|E_S||V|^{0,5})$=$O(|V|^{2,5})$.

L'algoritmo di routing nel suo complesso presenta dunque una complessità in tempo pari a $O(D|V|^{2,5})$.

Varianti

Osserviamo che l'algoritmo che determina un matching Ω-containing, può essere utilizzato anche in ***Routing_Matching_Massimo***, poiché essendo il multigrafo regolare, il matching Ω-containing copre tutti i vertici e quindi è anche massimo. Più precisamente, il matching determinato è un matching *perfetto*. Questo ci consente di sfruttare l'algoritmo per il calcolo di un matching

perfetto di R.Cole, K.Ost e S.Schirra di cui abbiamo già parlato nel paragrafo 2.5.1.4 e che ha complessità $O(|E|)$. Segue che ***Routing_Matching_Massimo*** ha complessità $O(D|E|)$.

2.5.3 Algoritmi di routing basati sulla decomposizione di matrici

Dalla proprietà 2.10 segue che un algoritmo che determina una decomposizione di una matrice, puo essere utilizzato come algoritmo di routing per una rete di Clos riconfigurabile.

Nei paragrafi 2.5.3.1 ed 2.5.3.2 presenteremo due algoritmi di decomposizione tratti da due ambiti diversi: quello relativo alle reti di Clos e quello riguardante le comunicazioni satellitari.

Ora illustriamo alcuni risultati inerenti la teoria del calcolo combinatorio che utilizzeremo per giustificare la correttezza dei due algoritmi di decomposizione.

Teorema 2.3 [CSHall67]

Sia T una matrice di numeri reali non negativi, di dimensione $r \times r$. Se ogni linea di T ha la stessa somma d, allora

$$T=\sum_{i=1}^{m} \mu_i P_i \quad (1)$$

con $\mu_i \geq 0$ e P_i matrice di permutazione.

Dimostrazione

Dimostriamo per induzione sul numero q degli elementi t_{ij} di T diversi da zero, che vale la (1).

Se $T \neq \varnothing$, per l'ipotesi del teorema, segue che $q \geq r$. Se $q=r$, è facile verificare che esiste una matrice di permutazione P tale che $T=dP$.

Trattiamo ora il caso in cui $q>r$.

Sia $S=\{1,2,...,r\}$. Consideriamo r sottoinsiemi $S_1,.....,S_r$ di S dove il generico S_i è costituito dagli indici di colonna tali che $t_{ij}>0$. Per il corollario 2.1 esiste un SDR di $S_1,.....,S_r$ che indichiamo con $C=\{j_1,j_2,..,j_i,..,j_r\}$. Definiamo la matrice P_1 di

dimensione $r \times r$ in modo che ogni suo elemento p_{ij_i} (i=1,..,r) valga uno e tutti gli altri siano nulli. Poiché C è un SDR, $j_1 \neq j_2 \neq ... \neq j_r$, quindi P_1 è una matrice di permutazione. Inoltre per definizione degli S_i, $j_1 \in S_1$,..., $j_r \in S_r$ segue che gli elementi t_{ij_i}, corrispondenti ai $p_{ij_i} = 1$, sono strettamente maggiori di zero.

Sia $\mu_1 = \min\{ t_{ij_i} \mid i=1,..,r \text{ e } j_i \in C \}$. Per quanto appena osservato, $T_1 = T - \mu_1 P_1$ è una matrice con elementi non negativi in cui ogni riga e colonna ha somma $d - \mu_1$. Inoltre, T_1 avrà un numero q_1 di elementi maggiori di zero minore di q, quindi per ipotesi induttiva vale la relazione: $T_1 = \sum_{i=2}^{m} \mu_i P_i$. Per definizione di T_1 segue la tesi del teorema:

$$T - \mu_1 P_1 = \sum_{i=2}^{m} \mu_i P_i \Rightarrow T = \sum_{i=1}^{m} \mu_i P_i \qquad \square$$

Consideriamo una matrice T $r \times r$ che sia d-regolare. La dimostrazione precedente, suggerisce il seguente schema per un algoritmo di decomposizione per T.

Schema Algoritmo di decomposizione

begin

i=1;

while T $\neq \varnothing$ do

begin

Determina un SDR $C=\{j_1,j_2,..,j_i,..,j_r\}$ per gli insiemi S_1,.....,S_r.

Costruisci la matrice di permutazione P_i in base a C;

$\mu_i = \min\{ t_{ij_i} \mid i=1,..,r \text{ e } j_i \in C \}$;

$T_i = \mu_i P_i$;

$T = T - T_i$;

$i = i+1$;

end ;

end ;

Rimangono da risolvere ancora due problemi:

- come determinare un SDR degli insiemi S_1,.....,S_r
- poter applicare l'algoritmo anche a quelle matrici le cui linee non hanno tutte la stessa somma.

Gli algoritmi di decomposizione che illustreremo nei prossimi paragrafi, seguono lo schema visto e risolvono i due problemi lasciati irrisolti..

2.5.3.1 Algoritmo di Neiman

Il primo algoritmo di routing per reti di Clos riconfigurabili basato sulla decomposizione di matrici, è stato proposto da V.I. Neiman nel 1969 in [CSNeim69].

```
Algoritmo Routing_Neiman(T_F)
begin
1. Aggiungi_traffico_fittizio(T_F);
2. i=1;L=∅;
3. while T≠∅ do
 begin
  3.1. C=Determina_un_SDR(T_F);
  3.2. P_i=Costruisci_matrice_di permutazione(C);
  3.3. L=L⋃P_i;    //L contiene la sequenza delle P_i
  3.4. T_F=T−P_i;
  3.5. i=i+1;
 end;
4. Togli_il_traffico_fittizio(L).
end;
```

Diamo una breve descrizione di come funziona l'algoritmo.

Fissati i parametri n ed r, sia F un qualsiasi frame di $\Phi(n,r)$ e T_F la matrice di traffico associata.

All'inizio, l'algoritmo aggiunge ad F delle connessioni "fittizie" in modo che ogni linea di T_F abbia stessa somma n. All'ultimo passo il traffico aggiunto sarà eliminato dalle matrici di permutazione. Questo consente di poter determinare una decomposizione di qualsiasi T_F anche se ogni riga e colonna della matrice non ha la stessa somma.

L'algoritmo procede sostanzialmente come descritto nello schema suggerito dalla dimostrazione del teorema 2.3. C'è però una differenza che lo rende meno efficiente. Supponiamo che la matrice di permutazione P_i nella decomposizione compaia $\mu_i > 1$ volte. Neiman calcola tutte le μ_i matrici P_i mentre nello schema questo è evitato utilizzando l'accorgimento $T = T - \mu_i P_i$.

Per quanto riguarda la determinazione dell' SDR, diamo solo qualche cenno poiché la procedura adottata è un po' macchinosa.
Siano $S_1,...,S_r$ gli r sottoinsiemi definiti nella dimostrazione del teorema 2.3.
Ogni riga di T è scandita sequenzialmente marcando un suo elemento nel modo seguente:

- Sia i la generica riga presa in considerazione. Se $t_{ij} > 0$ e nessun altro elemento sulla stessa colonna j nelle altre righe già scandite è stato marcato, allora marcalo.

Al termine si ottengono al più r elementi $t_{ij} > 0$ che formano l'SDR di $S_1,...,S_r$.
In realtà lo stesso Neiman dimostra che può accadere che solo $r/2$ elementi siano marcati. Per correggere questo comportamento, una seconda parte della procedura abbastanza complicata, "completa" l'SDR in modo che abbia cardinalità r.

Neiman non dà i dettagli per l'implementazione del suo algoritmo. In [CSJajs85] comunque, si sostiene che possa essere realizzato in modo che abbia una complessità in tempo $O(n|V|^4)$ (V è l'insieme dei vertici del multigrafo G_F che ha T_F come matrice d'incidenza).

In [CSNels74] è stato ripreso l'algoritmo di Neiman modificando la prima parte della procedura per la determinazione di un SDR. Questo ha fatto sì di rendere meno probabile la seconda fase di "completamento". La modifica, in termini di complessità, presenta miglioramenti solo nel caso medio mentre in quello pessimo non cambia.

Sempre nell'ambito delle reti di Clos, in [CSJajs85] e [CSRama73] sono stati proposti altri algoritmi di decomposizione. In seguito, in [CSCarp93], è stato dimostrato che non sono corretti.

Nel prossimo paragrafo proponiamo un algoritmo di decomposizione tratto dallo studio sulle comunicazioni satellitari. Per opportuni valori dei parametri r ed n, vedremo che l'algoritmo risulterà essere più efficiente di tutti quelli visti fino ad ora, ma "stranamente" nella letteratura sulle reti di Clos non viene menzionato.

2.5.3.2 Algoritmo Routing TSA

Nel campo delle comunicazioni satellitari, la determinazione di una decomposizione di una matrice d'interi, è chiamato *Time Slot Assignment* (TSA). Del TSA, esistono diverse varianti e tra queste quella che a noi interessa è la seguente.

Definizione Time Slot Assignment di Base Simmetrico[TSAInuk79]
Sia T una matrice $r \times r$ d'interi positivi.Un TSA di T è una decomposizione di T .

Un TSA di una matrice T si dice *ottimo,* se la lunghezza della decomposizione è minima.

Sia C(n,m,r) una rete di Clos riconfigurabile. Poiché M(n,r)=n, dalla proprietà 2.11, segue che per ogni frame $F \in \Phi(n, r)$ un TSA ottimo per la matrice T_F, ha lunghezza al più n. Per la proprietà 2.10, un algoritmo che determina il TSA ottimo è sicuramente un algoritmo di routing per la C(n,m,r) riconfigurabile.

Di seguito presentiamo un algoritmo che in tempo $O(|V|^{4,5})$ calcola il TSA ottimo di una matrice T.

Algoritmo ***Routing_TSA***(T_F) [TSABoCo81]

begin
1. Aggiungi_traffico_fittizio(T_F);
2. i=1; A=L=∅;
3. **while** T≠∅ **do**

```
 begin
  3.1. C=Determina_un_SDR(T_F);
  3.2. P_i=Costruisci_matrice_di permutazione(C);
  3.3. μ_i= min{ t_{ij_i} | i=1,..r  e j_i∈C };
  3.4. T_i =μ_i P_i;
  3.5. T_F =T_F-T_i;
  3.6. L=L∪P_i;    //L contiene la sequenza delle P_i
  3.7. A=A∪ μ_i;    //A contiene la sequenza dei coefficienti μ_i
  3.8. i=i+1;
 end;
4. Togli_il_traffico_fittizio(L).
end;
```

Diamo la descrizione dei passi principali dell'algoritmo.

Sia d il valore massimo che assume la somma di una linea di T_F. Al passo 1 l'algoritmo aggiunge a T_F del traffico "fittizio" in modo che T_F sia d-regolare. All'ultimo passo il traffico aggiunto verrà eliminato dalle matrici di permutazione. Analogamente a quanto osservato nel precedente paragrafo, questo consente di poter determinare una decomposizione di qualsiasi T_F. Osserviamo che il primo passo rispetto a quello dell'algoritmo di Neiman è leggermente diverso. Infatti, *Routing_Neiman* fa sì che la somma di ogni linea di T_F sia n e non d. Per la proprietà 1.2, una matrice di traffico associata ad un frame $F \in \Phi(n,r)$ è tale che la somma di ogni sua linea è al più n, dunque $d \leq n$.

Il resto dell'algoritmo stavolta segue totalmente quanto suggerito nella dimostrazione del teorema 2.3. In particolare, a differenza di Neiman, grazie ai passi 3.4 e 3.5 non sono calcolate matrici di permutazioni "inutili".

Non rimane che capire come al passo 3.1 venga determinato l'SDR di $S_1,.....,S_r$. Ricordiamo che il generico S è l'insieme degli indici di colonna di T per cui $t_{ij}>0$. Gli autori in [TSABoCo81] utilizzano allo scopo un algoritmo per il calcolo di un matching massimo. Vediamo di seguito come questo sia possibile.

Sia T_F una matrice di traffico in cui ogni riga e colonna ha la stessa somma d. Osserviamo che il multigrafo delle connessioni $G_F=(I,O,E)$ ha come matrice d'incidenza T_F quindi G_F è d-regolare. Per il lemma di Mendelson e Dulmage (si veda paragrago 2.2) sicuramente esiste un matching perfetto $M \subseteq E$. Vediamo ora come da M si possa determinare un SDR degli insiemi $S_1,....,S_r$.

Ricordiamo che un matching perfetto M di un multigrafo, copre tutti i suoi vertici, dunque M è anche massimo e $M=\{(I_1, O_{j_1}), (I_2, O_{j_2}),....., (I_r, O_{j_r})\}$. Per definizione di matching sappiamo che tutti i suoi archi non hanno vertici in comune, quindi $j_1 \neq j_2 \neq .... \neq j_r$. Inoltre, poiché $M \subseteq E$, tutti gli elementi t_{ij_i} di T_F sono positivi segue che $j_1 \in S_1,..., j_r \in S_r$. Possiamo quindi concludere che $\{j_1,j_2,.....,j_r\}$ è un SDR di $S_1,....,S_r$.

Valutiamo ora la complessità del passo 3.1.

Sia $G_F=(I,O,E)$ il multigrafo delle connessioni relativo alla matrice di traffico T_F.

Dalla proprietà 2.2 sappiamo che il matching massimo può essere calcolato sul grafo $G_S=(I,O,E_S)$ sotteso a G_F. L'algoritmo per il calcolo del matching massimo utilizzato in [TSABoCo81] è quello presentato in [ECHoKa73] che sappiamo avere complessità in tempo $O(|E_S||V|^{0,5})$. Come gia osservato nel paragrafo 2.5.2, poiché in un grafo $|E| \leq |V|^2$, il passo 3.1 costa $O(|V|^{2,5})$.

2.5.3.3 Relazione tra edge coloring e decomposizione di una matrice

Sia T una matrice $r \times r$ d'interi positivi e $G=(I,O,E)$ il multigrafo di cui T è la matrice d'incidenza. Il modo con cui è stato ricavato l'SDR nell'algoritmo *Routing_TSA*, ci consente di mostrare come da una decomposizione di T si possa determinare un edge coloring di G.

Supponiamo di aver determinato una decomposizione di T in m matrici elementari:

$$T = \sum_{i=1}^{m} E_i \quad (1)$$

Sia E_x una generica matrice elementare della decomposizione. Costruiamo un matching M_x per G nel seguente modo:

- per ogni elemento e_{ij}^x di E_x uguale ad uno, aggiungiamo ad M_x l'arco (I_i,O_j).

Verifichiamo che M_x è un matching per G.

Se $(I_i,O_j) \in M_x$ allora $e_{ij}^x=1$. Dalla (1) segue che l'elemento t_{ij} di T è maggiore o uguale ad uno. Essendo T la matrice d'incidenza di G, l'arco (I_i,O_j), appartiene ad E, dunque $M_x \subseteq E$.

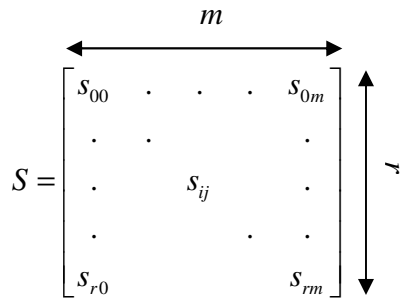
Per definizione di matrice elementare, se e_{ij}^{x}=1, tutti gli altri elementi di E_x sulla riga *i* e sulla colonna *j* sono nulli. Quindi M_x è un matching perché non esistono archi con vertici in comune.
Si può verificare facilmente che dall'unione degli *m* matching otteniamo tutti gli archi di *G.*
Abbiamo così ottenuto una decomposizione di *G* in *m* matching che sappiamo corrispondere ad un edge coloring di *G* con *m* colori.

2.5.4 Altri algoritmi: Algoritmo GS

Nel 1990 Gordon e Srikanthan [CSGoSr90] presentano un algoritmo di routing di complessità $O\left(D\,|V|^{2}\right)$ che nel 1996 è stato dimostrato essere non corretto da Hwang e Carpinelli in [CSHwan96].
Quest'ultimi hanno fornito una versione corretta dell'algoritmo ma che nelle sue parti essenziali ricalca l'originale. Di seguito vediamo la versione originale di Gordon e Srikanthan mettendo in evidenza il difetto che presenta, mentre faremo solo un accenno alla strategia che hanno seguito Carpinelli e Hwang per rimediare all'errore.

Algoritmo GS.
Sia C(*n,m,r*) una rete di Clos a tre stadi simmetrica ed *F* un suo frame compatibile.
Gordon e Srikanthan hanno introdotto l'utilizzo di una matrice $r \times m$ chiamata *Matrice di Specifica(S)* che "rappresenta" (vedremo come) una configurazione che realizza un frame *F* di connessioni. Diremo che *S* è *completa* se la configurazione che rappresenta è compatibile con la C(*n,m,r*), *incompleta* altrimenti.

La matrice di specifica *S,* ha tante righe quanti sono gli switch di ingresso e tante colonne quanti sono gli switch centrali. Il valore di un generico elemento di *S* è l'indice di uno switch di uscita. In termini di rotte diciamo che s_{ij}=*e*, rappresenta la rotta (*i,j,e*).

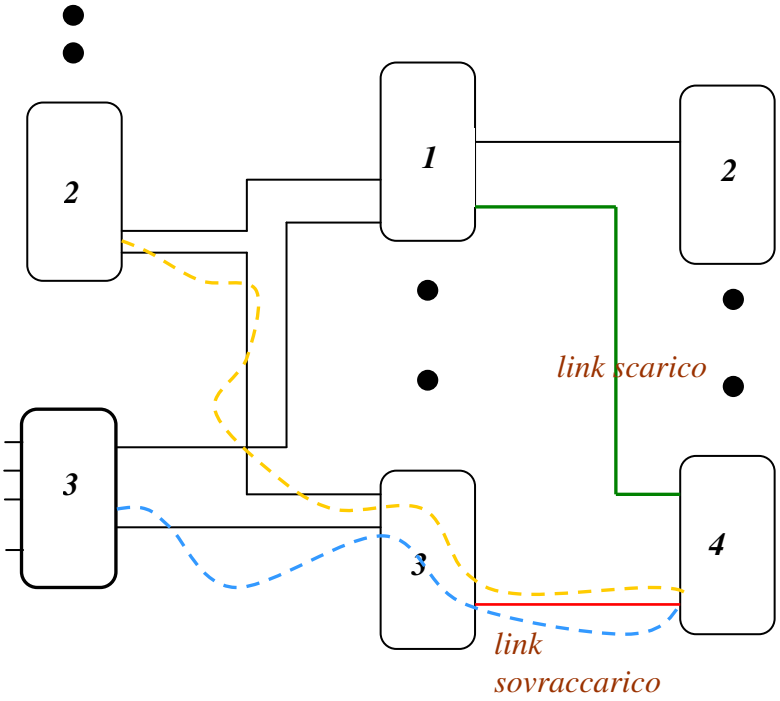


Osserviamo che affinché la matrice sia completa, in ogni colonna devono comparire tutti valori diversi. Infatti, siano s_{ij} e s_{uj} due elementi della stessa colonna *j*: se fosse $s_{ij} = s_{uj} = e$, significherebbe che per il link $(j,e)_2$ passano le due rotte (*i,j,e*) e (*u,j,e*) il ché non è ammissibile. In questa situazione diremo che la *colonna j* non è completa e che il link $(j,e)_2$ è *sovraccarico*.

Consideriamo il seguente esempio:

$$S = \begin{bmatrix} 1 & 4 & 2 \\ 2 & 3 & 4 \\ 3 & 1 & 4 \\ 2 & 1 & 3 \end{bmatrix}$$

Oserviamo che tutte le colonne sono non complete, ad esempio nella colonna 3 ci sono 2 elementi che valgono 4, ciò significa che il link $(3,4)_2$ tra secondo e terzo stadio, è attraversato dalle 2 rotte (2,3,4) e (4,3,4) quindi la configurazione non è compatibile con la rete. La figura seguente mostra la situazione:



L'idea di Gordon e Srikanthan è di "dirottare" uno dei due cammini che "sovraccaricano" il link $(3,4)_2$ verso lo switch centrale 1 in cui il link $(1,4)_2$ risulta "scarico".

Seguendo questa strategia, l'algoritmo GS è in grado di trasformare una matrice *S* non completa in una completa "scambiando" opportunamente gli elementidi una stessa riga.

Come vedremo tra breve, per compiere queste operazioni, l'algoritmo fa uso di un'ulteriore matrice $r \times m$ di interi detta *Matrice di Conteggio*(*C*), il cui generico elemento c_{ej} rappresenta il numero delle occorrenze del valore *e* nella colonna *j* di *S*. Riferendoci alla matrice *S* dell'esempio precedente, abbiamo:

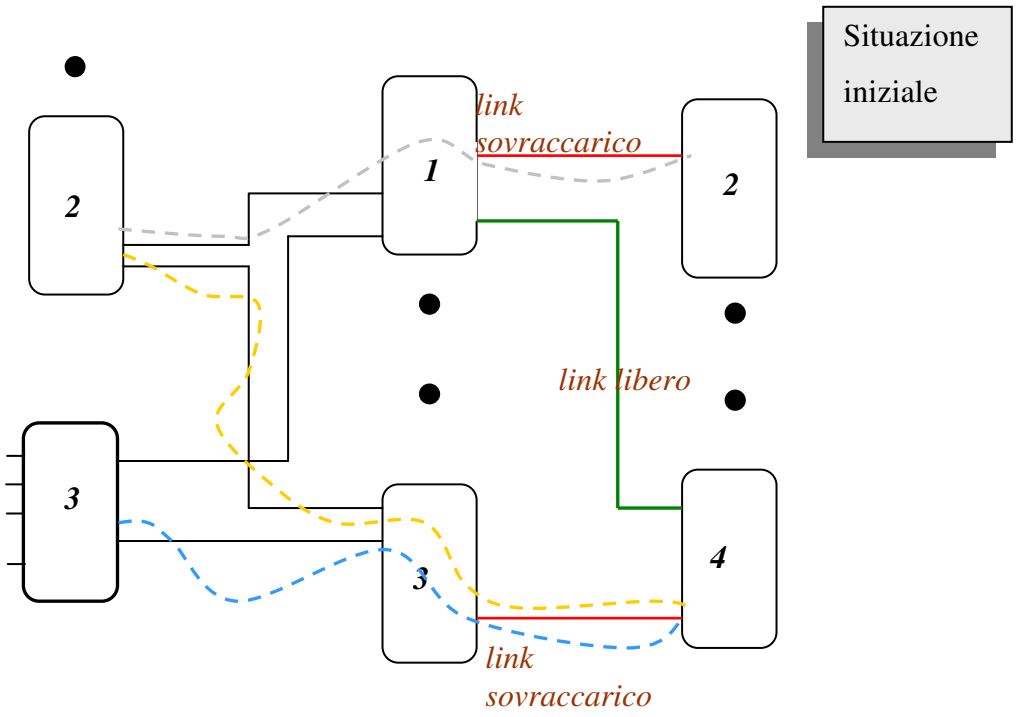
$$C = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

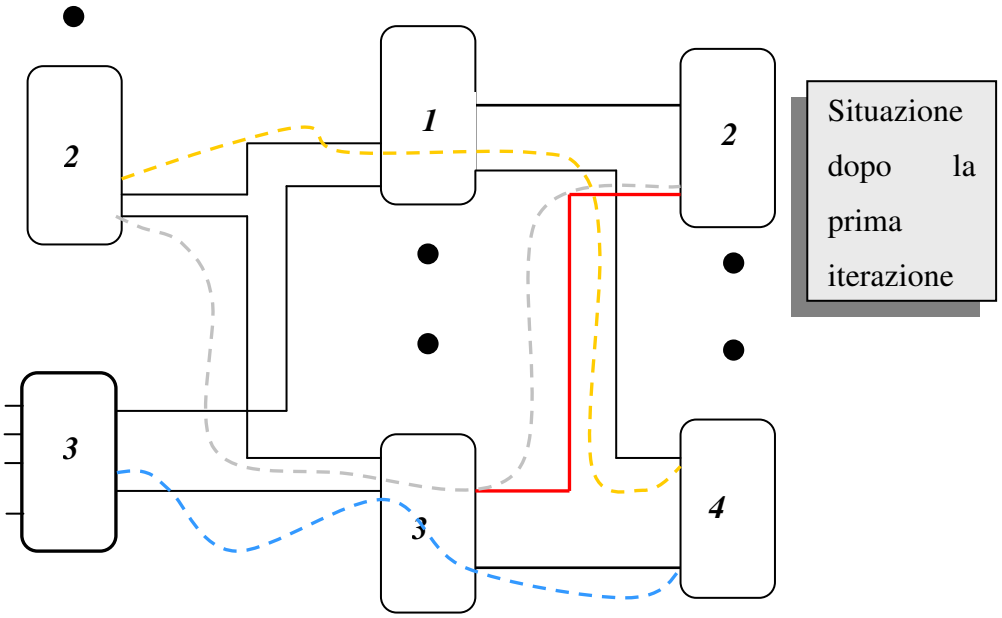
Diamo ora uno schema dell'algoritmo.

Agoritmo_GS

1. j=u=1;
2. Trova il minimo valore di j t.c., la colonna j-esima di S è incompleta. Se non esistono colonne incomplete, termina.
3. Trova una riga e di C t.c. c_{ej}=0.
4. Trova un elemento c_{ek} della riga e di C t.c. $c_{ek} \geq 2$. Determina una riga i di S seguendo l'ordine u, u+1,...., t.c. s_{ik} =e.
5. Scambia s_{ik} con s_{ij}.
6. u=i+1; torna al passo 2.

Per chiarire meglio come funziona l'algoritmo, eseguiamone un'iterazione sull'esempio dato in precedenza:
al *passo 2*, viene trovato che la colonna 1 è incompleta, quindi j=1.
Al *passo 3* il valore di e è 4. Si è trovato un link "scarico" ($(1,4)_2$) su cui sarà possibile far passare una rotta che sta "sovraccaricando" un altro link.
Al *passo 4*, k=3 e i=2. Abbiamo cioè determinato che il link $(3,4)_2$ è sovraccarico e che una delle rotte responsabili di tale condizione è la (2,3,4) che corrisponde all'elemento s_{23} di S.
Al *passo 5,* vengono scambiati gli elementi s_{21} ed s_{23} .Facendo riferimento alla figura successiva significa che abbiamo fatto passare la rotta grigia (2,1,2) attraverso lo switch centrale 3, mentre la rotta gialla (2,3,4) passerà attraverso lo switch centrale 1. Lo scambio è necessario poiché per far passare la rotta gialla dallo switch 1 abbiamo bisogno che il link $(2,1)_1$ sia libero.


Situazione iniziale
link sovraccarico
1
2
2
link libero
3
3
4
link sovraccarico


Situazione dopo la prima iterazione
2
1
2
3
3
4

La matrice *S* ottenuta dopo la prima iterazione è:

$$S = \begin{bmatrix} 1 & 4 & 2 \\ 4 & 3 & 2 \\ 3 & 1 & 4 \\ 2 & 1 & 3 \end{bmatrix}$$

come si può notare, il numero di colonne incomplete è passato da 3 a 2, ovvero, in termini di link, abbiamo "scaricato" i link $(1,2)_2$ e $(3,4)_2$ ma sovraccaricato il $(3,2)_2$.

Cosa non funziona nell'algoritmo.

In [CSHwan96] è dimostrato con un controesempio, che l'algoritmo GS, per opportune matrici *S*, non termina.
Il motivo per cui questo accade sta fondamentalmente nei passi 3 e 4 dove la scelta degli indici *k* ed *e* è lasciata libera nel caso ci siano più valori possibili. Vediamo un esempio in cui un' opportuna scelta dei valori di *k* ed *e*, comporta la non terminazione dell'algoritmo:

Esempio di non correttezza

Indichiamo con un asterisco, gli elementi di *S* che verranno scambiati.
Sia data *S*,

$$S = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 3 \\ 1 & 4 & 4 \\ 1 & 5 & 2 \\ 3 & 2 & 5 \end{bmatrix}$$

Prima iterazione: j=1, u=1, e=4, k=2, i=2

$$S = \begin{bmatrix} 1 & 3 & 5 \\ 2^* & 4^* & 3 \\ 1 & 4 & 4 \\ 1 & 5 & 2 \\ 3 & 2 & 5 \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Si scambiano gli elementi s_{21} e s_{23}.

Seconda iterazione: j=1, u=3, e=5, k=3, i=4

$$S = \begin{bmatrix} 1 & 3 & 5 \\ 4 & 2 & 3 \\ 1 & 4 & 4 \\ 1 & 5 & 2 \\ 3^* & 2 & 5^* \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Si scambiano gli elementi s_{51} e s_{53}.

Terza iterazione: j=1, u=1, e=2, k=2, i=2

$$S = \begin{bmatrix} 1 & 3 & 5 \\ 4^* & 2^* & 3 \\ 1 & 4 & 4 \\ 1 & 5 & 2 \\ 5 & 2 & 3 \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Si scambiano gli elementi s_{21} e s_{22}.

Quarta iterazione: j=1, u=3, e=3, k=3, i=5

$$S = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 3 \\ 1 & 4 & 4 \\ 1 & 5 & 2 \\ 5^* & 2 & 3^* \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Si scambiano gli elementi s_{51} e s_{53}. La matrice S che otteniamo è:

$$S = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 3 \\ 1 & 4 & 4 \\ 1 & 5 & 2 \\ 3 & 2 & 5 \end{bmatrix}$$

che è proprio la matrice di partenza. Se si effettuano le stesse scelte sui valori di *k* ed *e*, poiché *u*=1, segue che l'algoritmo ripeterà all'infinito le 4 iterazioni appena viste.

Nell'algoritmo proposto in [CSHwan96] (*Algoritmo_HC*) si corregge questo errore imponendo un'opportuna strategia alla scelta degli indici *k* ed *e*.

2.5.5 Confronto e sintesi dei risultati

Sia $G_F=(I,O,E)$ un generico grafo delle connessioni relativo alla rete di Clos C(*n*,*m*,*r*). Per definizione di G_F segue che $|I|=|O|=r$, $\deg(G_F)\le n$ e $|E|\le rn$. Nella tabella che segue riassumiamo le complessità degli algoritmi di routing espressi rispetto ai parametri *r* ed *n*. Abbiamo messo in evidenza anche i risultati che riguardano due reti di Clos particolari: la *rete di Benes* ($n=2$, $r=2^k$) e la *rete di Benes complementare* ($n=2^k$, $r=2$).

	Algoritmo	*Rete*	*Complessità*
Edge coloring	Cammini alternati *Ore, 1962*	Clos	$O(r^2 n)$
		Benes	$O(r^2)$
		Benes complementare	$O(n)$
	Euler Partition *Gabow 1976*	Clos	$O(r^{1.5} n\log n)$
		Benes	$O(r^{1.5})$
		Benes complementare	$O(n\log n)$
	Euler Partition *Cole, Ost e Schirra, 2001*	Clos	$O(rn\log n)$
		Benes	$O(r)$
		Benes complementare	$O(n\log n)$
Matching	Matching Massimo *Cole, Ost e Schirra, 2001*	Clos	$O(n^2 r)$
		Benes	$O(r)$
		Benes complementare	$O(n^2)$
	Matching Massimo *Hopcroft e Karp*	Clos	$O(n\, r^{2.5})$
		Benes	$O(r^{2.5})$
		Benes complementare	$O(n)$
	Matching Ω-containing	Clos	$O(n^2 r \log r)$
		Benes	$O(r \log r)$
		Benes complementare	$O(n^2)$
Decomposizione	Neiman *Neiman, 1969*	Clos	$O(nr^4)$
		Benes	$O(r^4)$
		Benes complementare	$O(n)$
	TSA *Bongiovanni, Coppersmith, 1981*	Clos	$O(r^{4.5})$
		Benes	$O(r^{2.5})$
		Benes complementare	$O(1)$
Altri	Algoritmo_HC (GS) *Carpinelli, Hwang e Lee, 1996*	Clos	$O(r^2 n)$
		Benes	$O(r^2)$
		Benes complementare	$O(n)$

Confrontando in termini di complessità i diversi algoritmi non possiamo dire che ne esista uno che si comporti meglio degli altri per qualsiasi rete di Clos. Infatti, la complessità dipende dai valori che assumono i parametri r ed n. Ad esempio se $r << (n\log n)^{2/7}$ allora, in generale, la migliore performance è data dall' algoritmo TSA. Al contrario se $r >> (n\log n)^{2/7}$ il miglior algoritmo è quello di edge coloring con partizioni euleriane di Cole, Ost e Schirra. Queste due condizioni sono esemplificate da due particolari reti di Clos, la rete di Benes e la rete complementare di Benes.

Nella pratica la complessità non è sufficiente per determinare quale sia il miglior algoritmo per una certa rete di Clos. Altri fattori influiscono: realizzabilità tecnologica, possibilità di rendere distribuito l'algoritmo, resistenza ai guasti (*fault tolerance*) ed altri aspetti che comunque esulano dai nostri scopi.
Ad esempio l' algoritmo GS è stato utilizzato nel dispositivo AT&T DACS IV-2000 switch-group anche se "peggiore", rispetto alla complessità, di altri [CSHwan96].

Per le reti di Clos generiche, va osservato che tutti gli algoritmi presentati, tranne *Routing_TSA*, sono *pseudopolinomiali*. Infatti, le "espressioni" delle loro complessità in tempo sono del tipo $O(n \cdot ...)$. Per n "sufficientemete grandi" gli algoritmi possono dunque avere un tempo di calcolo esponenziale (es. $n=2^r$).