4. VISUALIZZAZIONE DI GRAFI: ALCUNI CASI PARTICOLARI

4.1. Disegno ortogonale 2D

Riferimenti: [KW98] pp. 121-122, 126-141, [Dal99] pp. 137-138.

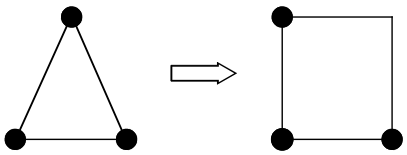
Esistono molti criteri per giudicare la qualità di una visualizzazione di un grafo. Da un punto di vista umano uno dei parametri più rilevanti è la leggibilità della visualizzazione, cioè la sua capacità di trasmettere in modo immediato il significato del grafo visualizzato. In questo senso, in particolare, alcune cose importanti da evitare sono: confondere quali siano gli estremi di un arco, rendere lungo ed intricato il percorso degli archi, sovrapporre tra loro due porzioni di archi, due nodi, o un nodo ed un arco.

Da un punto di vista algoritmico, è necessario catturare la qualità di una visualizzazione tramite delle funzioni obiettivo. Si tenta, quindi, di tradurre una regola umana tra quelle precedentemente accennate in una automatica; ad esempio, voler evitare di confondere gli estremi di un arco si può tradurre nel tentare di mantenere gli archi ben distanti. Tuttavia, questo può essere difficile soprattutto in prossimità dei nodi, dove molti archi si incontrano. Allora un'idea potrebbe essere quella di mettere in relazione l'angolo che gli archi formano tra loro quando incidono su di un nodo e la capacità ottica di distinguere gli archi. Questo induce una particolare funzione obiettivo: trovare il disegno tale che il minimo angolo tra due archi sia massimizzato. Formalmente, dato un grafo G, la sua risoluzione angolare è definita come il massimo, calcolato su tutte le sue rappresentazioni rettilinee, del più piccolo angolo tra archi adiacenti. Lo scopo è quello di determinare la rappresentazione in cui tale massimo è raggiunto.

Esiste un particolare tipo di disegno che cerca di rispondere a questo criterio, ed è il disegno ortogonale, in cui gli angoli sono forzati ad essere multipli di 90°; in tal caso gli archi risultano sempre paralleli agli assi cartesiani.

In tal modo però un disegno rettilineo potrebbe non essere più possibile, ed il prezzo da pagare è introdurre delle svolte (come esempio, si veda il triangolo, Figura 4.1). Per evitare confusione nell'ottenere degli archi che seguono percorsi troppo complicati, l'ideale è minimizzare il numero di svolte.

Figura 4.1.



Sfortunatamente, Formann et al. [Fal90] hanno dimostrato che capire se un dato grafo G ammetta una rappresentazione piana con risoluzione angolare di almeno 90° con un fissato numero k di svolte è un problema NP-completo, cioè risulta NP-arduo minimizzare il numero di svolte. Il punto cruciale della riduzione sta nel fatto che è difficile trovare il giusto ordinamento degli archi intorno a ciascun nodo. Tuttavia, per molti problemi, può essere sufficiente visualizzare un grafo in modo ortogonale quando una rappresentazione sia già fissata. Questo fa cadere l'NP-completezza ed esiste un algoritmo di Tamassia [T87] che rappresenta in modo ortogonale con il minor numero di svolte un grafo di cui è data la rappresentazione planare. Questo algoritmo si basa sulla costruzione di reti di flusso e la complessità è la stessa che serve per trovare un flusso di costo minimo su una rete; quando l'algoritmo è stato proposto, tale complessità era $O(n^2 \log n)$, ora è $O(n^{7/4} \log^{1/2} n)$.

Definizione: Un *disegno ortogonale su griglia* di un grafo $G=(V,E)$ è una funzione biunivoca che mappa nodi $v \in V$ in punti a coordinate intere $\Gamma(v)$ ed archi $(v,w) \in E$ in cammini che non si sovrappongono tali che le immagini dei loro estremi $\Gamma(v)$ e $\Gamma(w)$ siano connesse dai cammini corrispondenti. Tali cammini sono costituiti da segmenti orizzontali o verticali con le eventuali svolte a coordinate intere.

Un disegno ortogonale su griglia è detto *semplice* se nessun arco presenta svolte.

Si osservi che, poiché non è permessa sovrapposizione di archi, non è possibile rappresentare grafi con grado >4. Questa sezione si focalizzerà quindi su grafi con grado non superiore a 4, mentre nel seguito verranno proposti dei metodi per estendere gli stessi algoritmi a grafi con grado più alto.

4.1.1. Codifiche di rappresentazioni di grafi

Riferimenti: [KW98] pp. 126-129.

Prima di descrivere alcuni algoritmi per il disegno ortogonale su griglia, verrà descritto come codificare un grafo ed alcune sue rappresentazioni, in modo da usare queste codifiche per l'input e per l'output degli algoritmi.

Un *grafo* (planare o no) può essere semplicemente descritto tramite l'elenco dei suoi nodi ed archi. Una *rappresentazione piana* di un grafo planare contiene delle informazioni toplogiche addizionali: essendo gli archi rappresentati come curve di Jordan, vengono automaticamente individute delle regioni chiuse (ed una regione aperta) dette *facce*. Data una rappresentazione piana, la struttura delle facce è caratterizzata dai cicli che le delimitano, e quindi si può parlare di questi cicli come delle facce stesse. E' semplice vedere che le facce definiscono univocamente una data rappresentazione piana, poiché inducono un ordinamento univoco degli archi intorno ai nodi. E' quindi possibile descrivere una rappresentazione piana come una lista di nodi, una lista di archi, una lista di facce ed una lista di cicli che definiscono le varie facce. Il *verso di percorrenza* degli archi è, convenzionalmente, quello che consente di avere, percorrendo il perimetro di una faccia, l'interno della stessa sulla destra; da questo segue che per le facce interne il verso di percorrenza è orario, per la faccia esterna è antiorario (v. Figura 4.2).

Figura 4.2.

$F = \{ f_1, \ldots, f_5 \}$;

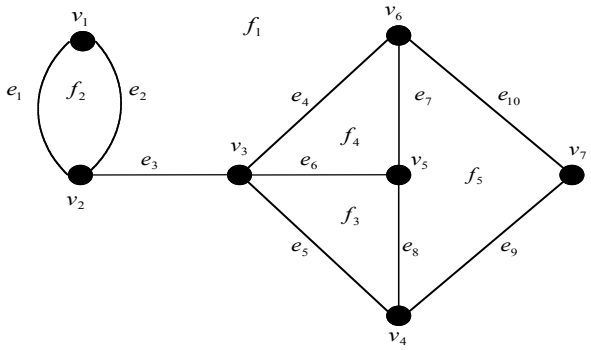
$P(f_1) = (e_1, e_3, e_5, e_9, e_{10}, e_4, e_3, e_2)$;

$P(f_2) = (e_2, e_1)$;

$P(f_3) = (e_5, e_6, e_8)$;

$P(f_4) = (e_6, e_4, e_7)$;

$P(f_5) = (e_8, e_7, e_{10}, e_9)$;



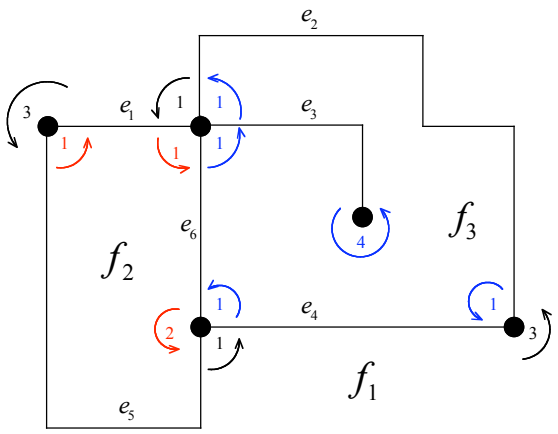
Sono necessarie ancora più informazioni quando si vuole codificare un *disegno ortogonale*. In particolare è necessario fornire informazioni circa le svolte su ciascun arco (numero e direzione). Un modo per memorizzare questa informazione è il seguente: per ogni arco nella lista di una faccia, si descriva la sequenza di svolte con una stringa binaria che rappresenta l'elenco delle svolte incontrate percorrendo quell'arco. In particolare, un *1* rappresenta una svolta a sinistra e uno *0* rappresenta una svolta a destra; archi rettilinei sono caratterizzati dalla stringa vuota *ε*. Si assegni ad ogni arco anche un angolo, cioè un numero che rappresenta una costante moltiplicativa di 90°, che esprima quale sia l'angolo racchiuso tra l'ultimo segmento dell'arco in considerazione e il primo segmento dell'arco successivo della faccia (v. Figura 4.3).

Figura 4.3.

H(f_1) = ((e_1,ε,3), (e_5,11,1), (e_4,ε,3), (e_2,1011,1));

H(f_2) = ((e_1,ε,1), (e_6,ε,2), (e_5,00,1));

H(f_3) = ((e_2,0010,1), (e_4,ε,1), (e_6,ε,1), (e_3,0,4), (e_3,1,1));



Questa rappresentazione deve seguire delle *regole di consistenza*. In particolare:

1. deve esistere un grafo 4-planare a cui la rappresentazione si riferisce;
2. ogni arco deve essere memorizzato due volte, una per ciascuna faccia a cui appartiene, e le due memorizzazioni devono essere consistenti;
3. la somma degli angoli lungo il perimetro di una faccia deve essere consistente col fatto che la faccia sia un poligono chiuso;
4. per ogni nodo, la somma degli angoli che insistono su quel nodo deve essere 4.

Si osservi che questa rappresentazione porta con se' solo un'informazione geometrica del disegno, ma nulla sulla lunghezza degli archi. Esiste un algoritmo per ricavare una rappresentazione grafica del grafo su griglia di area minima a partire dalla rappresentazione teorica ora descritta, dovuto a Tamassia [T87], che ha complessità computazionale $O((n+b)^{7/4} \log^{1/2}(n+b))$, dove n è il numero di nodi e b il numero di svolte. Qualunque sia l'algoritmo usato per determinare una rappresentazione ortogonale del grafo, è sempre possibile applicare poi questo algoritmo che garantisce area minima. Tuttavia, è da notare che quest'ultima fase fa perdere la linearità a tutti gli algoritmi, e questo è il motivo per cui, di solito, si rinuncia all'area minima del disegno pur di ottenere una rappre-sentazione "ragionevolmente compatta" molto più velocemente.

Vengono di seguito riportati degli algoritmi per disegnare grafi ortogonalmente su griglia. Essi si dividono in due categorie, che si basano su due approcci completamente diversi, e cioè la rappresentazione di visibilità e la st-numerazione.

Entrambi i metodi richiedono al grafo in input di essere 2-connesso. Nel seguito verranno date delle tecniche algoritmiche per eliminare questa condizione.

Prima di entrare nel dettaglio degli algoritmi, viene premesso un paragrafo con alcune definizioni fondamentali e l'algoritmo di st-numerazione, utile nel seguito.

4.1.2. st-grafi ed st-numerazione

Riferimenti: [CN??] pp. 34-40.

Definizione: Dato un grafo 2-connesso G, una *st-numerazione* dei nodi di G è un'etichettatura dei nodi con gli interi $1, 2, \ldots n$ tale che ogni nodo etichettato j $(2 \leq j \leq n-1)$ abbia almeno un vicino etichettato con $i<j$ ed almeno un vicino etichettato con $k>j$. Inoltre i nodi etichettati con 1 ed n sono collegati.

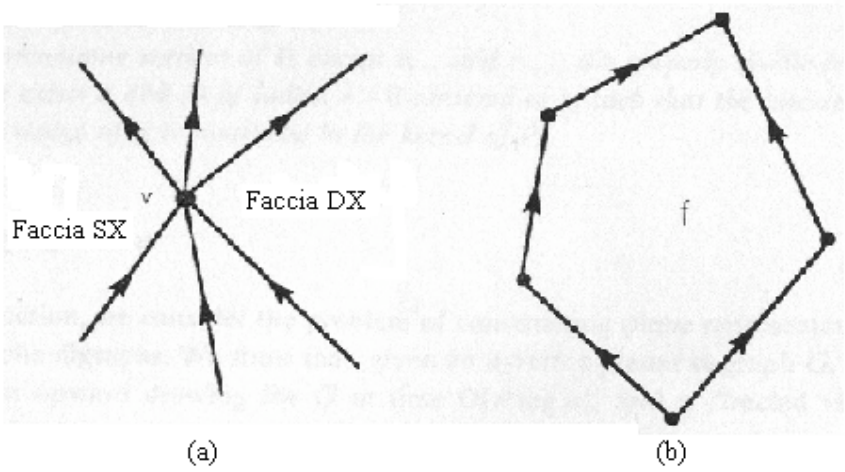
E' immediato indurre un verso agli archi, dai numeri minori a quelli maggiori, e quindi la definizione precedente equivale all'esistenza, in tutti i nodi, di almeno un arco entrante ed almeno un arco uscente, esclusi 1 ed n, che sono collegati e sono, rispettivamente, l'unica sorgente s (dall'inglese *source*) e l'unico pozzo t (dall'inglese *tale*) di G.

Un grafo orientato con le suddette proprietà si chiama *st-grafo*.

Proprietà. Un *st*-grafo $G=(V,E)$ gode delle seguenti proprietà:

1. ogni nodo $v \in V$ è su un cammino da s a t;
2. per ogni nodo $v \in V$, gli archi entranti appaiono consecutivamente intorno a v, e così gli archi uscenti (vedi Fig. 4.4.a); in questo modo è sempre possibile distinguere in maniera univoca due facce distinte, quella a sinistra di v (tra il primo arco entrante e il primo arco uscente) e quella a destra di v (tra l'ultimo arco entrante e l'ultimo arco uscente);
3. per ogni faccia f di G, i confini di f sono delimitati da due cammini distinti con origine e destinazione comune (vedi Fig. 4.4.b);
4. tutti i cammini sono orientati da s a t, ossia G ammette sempre un disegno upward.

Figura 4.4.



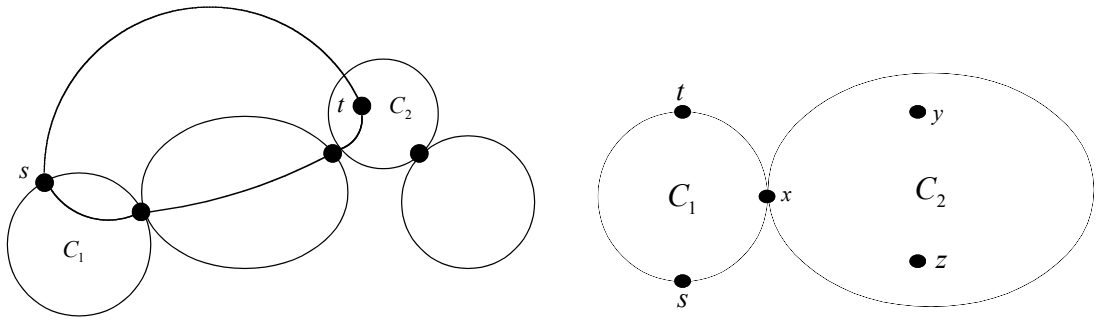
Teorema (Even e Tarjan [ET76]): *Sia $G=(V,E)$ un grafo 2-connesso, siano s e t due nodi di V tali che (s,t) sia un arco di E. Allora esiste una st-numerazione tale che s sia il primo nodo e t sia l'ultimo. Essa può essere calcolata in tempo $O(m)$.*

Prima di dimostrare il teorema, si osservi che se G non è 2-connesso, non è possibile trovare una st-numerazione valida. Infatti, siano dati un grafo G semplicemente connesso (quindi con almeno un punto di articolazione) e i due nodi s e t.

Per assurdo, supponiamo che su G sia stata trovata una st-numerazione valida.

Supponiamo che s e t giacciano in componenti 2-connesse diverse (v. Fig. 4.5 parte sinistra); si consideri un cammino da s a t in cui tutti i nodi abbiano st-numero via via crescente; tale cammino esiste certamente per la definizione di st-numerazione. D'altra parte, si osservi che s e t devono essere connessi da un arco. Ma allora, tra s e t esistono due cammini disgiunti, e quindi s e t giacciono in realtà nella stessa componente. Supponiamo allora, che s e t siano nella stessa componente 2-connessa (v. Fig. 4.5 parte destra) e dimostriamo che G non può avere una st-numerazione. Infatti, se per assurdo ci fosse, senza perdere di generalità, G sia costituito solo da due componenti 2-connesse, quella che contiene s e t ed un'altra. Sia x il punto di articolazione, e siano y e z, rispettivamente, i nodi con numerazione massima e minima nella componente che non contiene s e t. Se x è distinto da y, esso ha numerazione $< y$ (che ha numerazione massima), allora y non ha un vicino con numerazione maggiore di lui, e quindi la numerazione non è st. Se invece x coincide con y, allora z non ha alcun vicino con numerazione minore di lui.

Figura 4.5.



La dimostrazione del teorema precedente consiste nel fornire un algoritmo di st-numerazione. Per fare ciò abbiamo bisogno di definire alcune funzioni:

- *DFN(v):* il numero assegnato al nodo v da una visita in profondità da s a t;
- *FATH(v):* il padre del nodo v;
- *LOW(v) = min{DFN(v), DFN(w)* tali che ∃ un arco all'indietro *{u,w}* con u discendente di v (o v stesso) e w antenato di v*}*.

È possibile dare anche una definizione ricorsiva della funzione *LOW(v)*, che sfrutti i valori dei nodi già calcolati:

LOW(v) = min { DFN(v), LOW(x) con x figlio di v, *DFN(w)* con *{v,w}* arco all'indietro*}*;

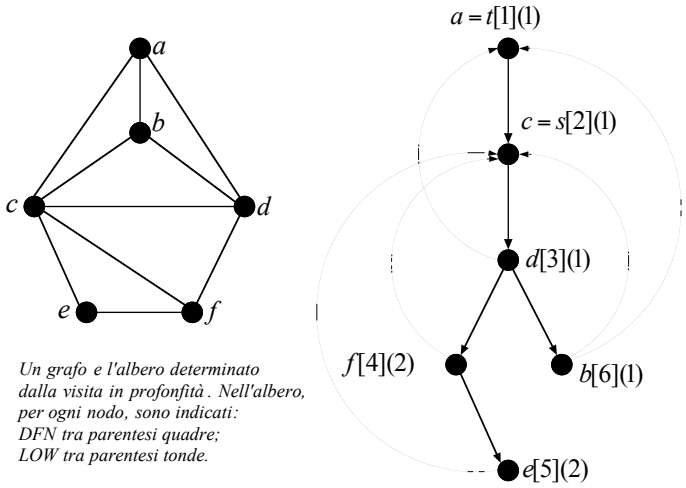
Esempio (v. Fig. 4.6):

Si consideri ora la funzione *PATH(v)* che, inizialmente, marca il nodo t, il nodo s e l'arco che li congiunge, poi, per ogni nodo v restituisce, se possibile, un cammino non marcato da v ad un altro nodo marcato, marcando tutti i nodi e gli archi su questo cammino.

Ci sono quattro casi possibili:

1. c'è un arco di riporto non marcato *{v,w}* (v. Fig. 4.7, sinistra); in questo caso viene marcato l'arco *{v,w}* e *PATH(v) = v w*.
2. esiste un arco dell'albero non marcato *{v,w}* (v. Fig. 4.7, centro); in questo caso *PATH(v)* si basa sulla funzione *LOW(w)*: sia $w=w_0w_1\ldots w_ku$ il cammino che percorre l'albero e termina con un arco all'indietro in un nodo u tale che $LOW\ (w_i) = DFN(u)$, $i=0, \ldots k$. Allora $PATH(v)= vw_0w_1\ldots w_ku$ e vengono marcati tutti i nodi e gli archi nel cammino.

Figura 4.6.

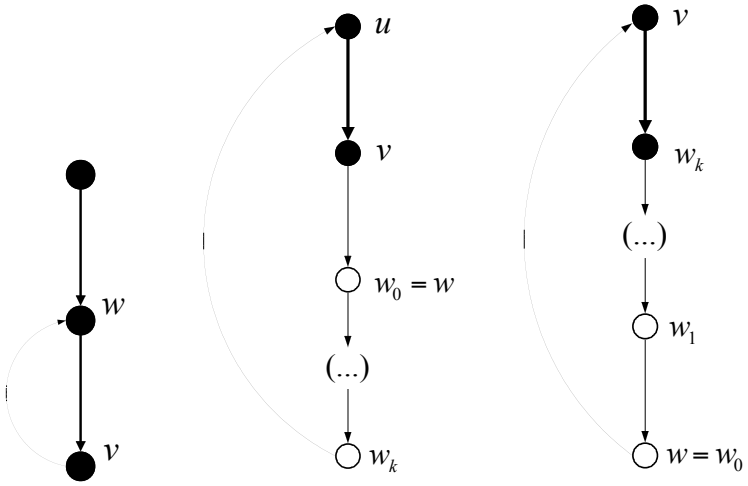


Un grafo e l'albero determinato dalla visita in profonfità . Nell'albero, per ogni nodo, sono indicati: DFN tra parentesi quadre; LOW tra parentesi tonde.

3. esiste un arco di riporto non marcato $\{w,v\}$, cioè per cui $DFN(w) > DFN(v)$ (v. Fig. 4.7, destra) e supponiamo che $w=w_0w_1\ldots w_k$ sia il cammino all'indietro che risale sugli archi dell'albero seguendo la funzione *FATH*, da w ad un nodo marcato. In questo caso $PATH(v) = vw_0w_1\ldots w_k$ e vengono marcati tutti i nodi e gli archi nel cammino.
4. tutti gli archi incidenti a v sono marcati; in questo caso PATH(v) =∅.

Si osservi che i valori di *FATH* e *LOW* di ogni nodo, sono calcolabili contestualmente alla visita in profondità.

Figura 4.7.



Descriviamo ora l'algoritmo che calcola una st-numerazione, dovuto ad Even e Tarjan [ET76].

Algoritmo ST-NUMBER(G)
Input: un grafo $G=(V,E)$ 2-connesso con $s, t \in V$ tali che $(s,t) \in E$;
Output: una st-numerazione di G;
esegui una visita in profondità calcolando *DFN, FATH, LOW* di ogni nodo;
marca s,t, ed $\{s,t\}$;
PUSH(PILA, t);
PUSH(PILA, s);

```
cont = 1;
v = POP(PILA);              //al primo passo sarà il nodo s
while (v ≠ t) do
    if PATH(v) = ∅
            then STN(v) = cont;
                 cont++;
            else (supponendo che PATH(v)= vv_1…v_kw)
                    PUSH(PILA, v_k…v_1v);
                    v = POP(PILA);
endwhile
STN(t) = cont;
```

Esempio (v. Fig. 4.8):

Inizialmente viene effettuata la *DFS* che produce l'albero in Figura 4.8 coi valori di *DFN* e *LOW*. La variabile cont viene inizializzata ad *1*. Vengono messi in pila *t* (cioè *a*) ed *s* (cioè *c*).

Viene estratto dalla pila il nodo *c* e *PATH*(*c*)=(*c*,*d*,*a*) (che corrisponde al secondo caso tra quelli presentati per la determinazione della funzione), quindi vengono messi in pila, nell'ordine: *d*,*c*.

Viene estratto dalla pila di nuovo il nodo *c* e *PATH*(*c*)=(*c*,*b*,*d*) (terzo caso), quindi vengono messi in pila, nell'ordine: *b*,*c*.

Viene estratto dalla pila di nuovo il nodo *c* e *PATH*(*c*)=(*c*,*e*,*f*,*d*) (terzo caso), quindi vengono messi in pila, nell'ordine: *f*,*e*,*c*.

Viene estratto dalla pila di nuovo il nodo *c* e *PATH*(*c*)=(*c*,*f*) (terzo caso), quindi viene messo in pila il solo nodo *c*.

Viene estratto dalla pila di nuovo il nodo *c* e *PATH*(*c*)= ∅ (quarto caso) quindi *STN(c)*=cont=*1* e cont viene incrementata.

A questo punto viene estratto dalla pila il nodo *e* e *PATH*(*e*)= ∅ (quarto caso) quindi *STN*(*e*)=cont=*2* e cont viene incrementata.

Viene estratto dalla pila il nodo *f* e *PATH(f)*= ∅ (quarto caso) quindi *STN(f)*=cont=*3* e cont diviene *4*.

Viene estratto dalla pila il nodo *b* e *PATH(b)=(b,a)* (primo caso) quindi viene messo in pila il solo nodo *b*. Tale nodo viene riestratto e questa volta *PATH(b)*= ∅ (quarto caso) quindi *STN(b)*=cont=*4* e cont diventa 5.

Viene estratto dalla pila il nodo *d* e *PATH(d)*= ∅ (quarto caso) quindi *STN(d)*=cont=*5* e cont viene incrementata a 6.

Infine, viene estratto dalla pila il nodo *a*, cioè *t*, quindi si esce dal ciclo while e viene assegnato l'ultimo valore *STN(a)=6*.

Teorema. *L'algoritmo ora descritto trova una st-numerazione di un grafo 2-connesso G in tempo O(n+m).*

Dimostrazione. Ogni nodo riceve una numerazione solo quando viene rimosso definitivamente dalla pila e questo accade solo quando ogni suo arco incidente viene marcato. Visto che il grafo è 2-connesso, è possibile raggiungere da *s* ogni nodo senza passare per *t*, quindi ogni nodo è inserito nella pila prima che *t* venga rimosso (ricordiamo che *t* è il primo nodo ad essere inserito nella pila). Abbiamo così dimostrato che ogni nodo viene numerato.

Proviamo ora che la numerazione assegnata è proprio una st-numerazione.

Banalmente *STN(s)=1* e *STN(t)=n* per come è fatto l'algoritmo. Ogni altro nodo viene posizionato nella pila, per la prima volta, come nodo intermedio di un cammino; in questo modo, due tra i vicini di ogni nodo verranno messi nella pila uno sotto di lui e l'altro sopra di lui; tra l'altro, se anche tale nodo fosse l'ultimo del cammino, troverebbe sempre il suo vicino *w* già in pila,

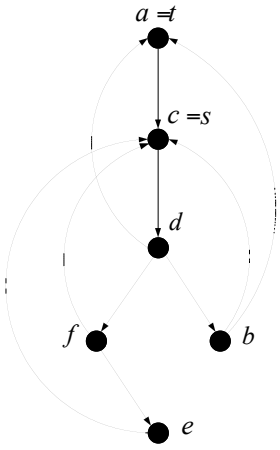
mentre se fosse il primo, avrebbe sempre *v* sopra di lui. Inoltre, visto che ogni nodo viene numerato nel momento della sua definitiva rimozione dalla pila, i due vicini che precedono e seguono il nodo considerato otterranno, rispettivamente, un st-numero superiore e un st-numero inferiore rispetto a quel nodo. Quindi la numerazione trovata dall'algoritmo rispetta le proprietà della st-numerazione.

Infine, l'algoritmo lavora in tempo lineare nella dimensione del grafo poiché la prima parte dell'algoritmo, come detto, calcola le varie funzioni (*LOW, DFN, FATH*) tramite una visita in profondità che, come noto, ha complessità $O(n+m)$. Per quanto riguarda il ciclo while, si osservi che il punto cruciale è dato dal calcolo della funzione *PATH* di ogni nodo. Per questa funzione, come visto, si possono avere due casi:

- trova un cammino, ed in questo caso marca x archi e $x+1$ nodi (con $0 \le x \le n-2$);
- non trova un cammino (cioè tutti gli archi incidenti al nodo in questione sono marcati) ed in questo caso assegna *STN*;

Questa considerazione assicura che il ciclo non sia infinito. Inoltre, ogni arco e nodo viene visitato esattamente una volta; questo permette di affermare che la complessità totale è, nel caso peggiore, $O(n+m)$. **CVD**

Figura 4.8.



4.1.3. Un algoritmo basato sulla rappresentazione di visibilità

Riferimenti: [Dal99] pp. 99-102, 130-132, [KW98] pp. 133-137, [DT88] sezioni 1, 2 e, della sezione 3, solo i teoremi 3.4, 3.5 e figura 12.

Prima di descrivere l'algoritmo vediamo alcune definizioni e proprietà utili nel seguito.

Definizione: Un grafo planare si dice *planare massimale* quando non è possibile aggiungervi un arco senza perdere la planarità.

E' facile convincersi che tutte le facce di un grafo planare massimale sono dei triangoli; ne segue che, detto n il numero dei nodi, m il numero di archi ed f il numero delle facce, risulta $3f=2m$.

Per ogni grafo planare vale quindi

$$3f \le 2m.$$

Le tre grandezze n, m ed f sono legate anche dalla formula di Eulero:

$$n - m + f = 2.$$

Sostituendo la disuguaglianza trovata nella formula di Eulero, si ha:

$n - m + f = 2 \Rightarrow 2n - 3f + 2f \ge 4 \quad \Rightarrow \quad f \le 2n - 4 \quad \Rightarrow \quad f\text{-}1 \le 2n - 5.$

Definizione: Il *duale* G^* di un grafo orientato G è il grafo orientato così definito (vedi Fig.4.9):

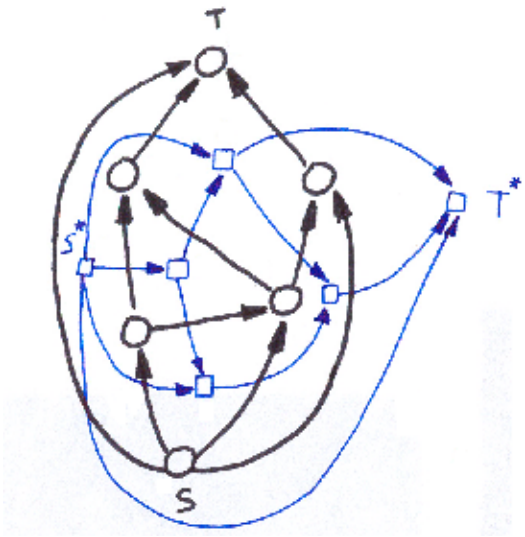
1. i nodi di G^* sono le facce di G, dove s^* e t^* sono rispettivamente la faccia a destra dell' arco (s, t) e la faccia esterna;
2. un arco (f,g) è in G^* se la faccia f condivide con la faccia g un arco $(v,w) \neq (s, t)$;
3. l'arco (f,g) è orientato da f verso g se f si trova a sinistra dell'arco orientato (v,w);
4. G^* contiene l'arco (s^*,t^*).

Si osservi che il duale di un grafo planare è ancora planare, ed è mantenuto l'orientamento (upward → rightward e downward → leftward).

Dato un qualunque nodo v di G, il valore $\alpha(v)$ è definito come la lunghezza del più lungo cammino per andare da s a v. Per ogni nodo f di G^* il valore $\beta(f)$ è definito come la lunghezza del più lungo cammino per andare da s^* a f.

Sia $Right(u,v)$ la faccia che si trova a destra dell'arco (u,v).

Figura 4.9.



Definizione [OW78]: La *rappresentazione di visibilità* $\Gamma(G)$ di un grafo $G=(V,E)$ è una sua rappresentazione che disegna ogni nodo $v \in V$ come un segmento orizzontale $\Gamma(v)$, ed ogni arco $(v,w) \in E$ come un segmento verticale le cui estremità giacciono su $\Gamma(v)$ e $\Gamma(w)$ e non interseca nessun altro segmento nodo $\Gamma(u)$ con $u \neq v,w$. Inoltre, ne' i segmenti orizzontali ne' i segmenti verticali si sovrappongono.

Un segmento orizzontale $\Gamma(s)$ è detto *sorgente* se tutti i segmenti verticali ad esso incidenti vanno verso l'alto; un segmanto orizzontale $\Gamma(t)$ è detto *pozzo* se tutti i segmenti verticali ad esso incidenti provengono dal basso.

Lemma [RT86]: *Sia G un grafo planare 2-connesso con n nodi. Allora, comunque si scelgano due nodi s e t adiacenti, esiste una rappresentazione di visibilità* $\Gamma(G)$ *per G tale che* $\Gamma(G)$ *ha esattamente una sorgente* $\Gamma(s)$ *ed esattamente un pozzo* $\Gamma(t)$.

Dimostrazione. La dimostrazione è costruttiva, quindi verrà provato come una rappresentazione con queste proprietà possa essere costruita in tempo $O(n)$.

Il grafo G può essere trasformato in un st-grafo in tempo $O(n+m)=O(n)$. Un algoritmo che ne determini una rappresentazione di visibilità è il seguente:

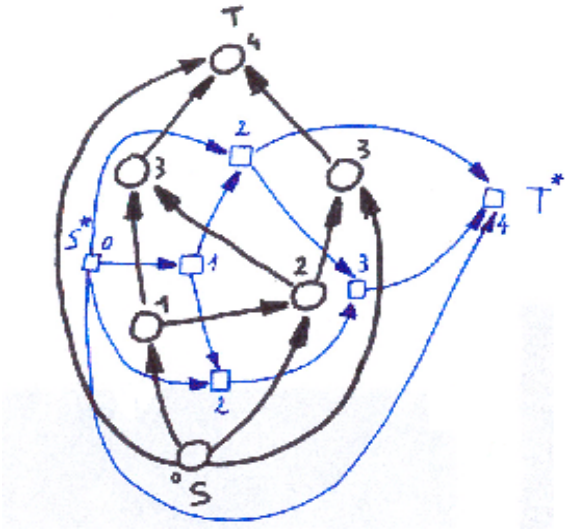
Input: Un *st*-grafo planare G dato tramite una sua rappresentazione piana;
Output: Una rappresentazione di visibilità Γ del grafo G.

1. costruisci il grafo duale G^* di G;
2. calcola, per ogni nodo v di G il valore $\alpha(v)$;
3. calcola, per ogni nodo f di G^* il valore $\beta(f)$;
4. per ogni arco (u,v) di G do
   disegna un segmento verticale con ascissa $x(u,v) = \beta(Right(u,v))$
   tra le ordinate $\alpha(u)$ e $\alpha(v)$;
5. per ogni nodo v do
   disegna un segmento orizzontale con ordinata $y(v) = \alpha(v)$ tra le ascisse più a sinistra e più a destra dei segmenti verticali associati agli archi incidenti su v.

La correttezza di questo algoritmo discende direttamente dalle proprietà degli st-grafi. **CVD**

Esempio: Si supponga di prendere in ingresso il grafo orientato di Fig. 4.9. Dopo l'esecuzione dei primi tre passi dell'algoritmo, si otterrà la rappresentazione del digrafo mostrato in Figura 4.10, in cui sono mostrati i valori $\alpha(v)$ e $\beta(f)$. Dopo l'esecuzione del passo 4 si ottiene quanto mostrato in figura 4.11, parte sinsitra, mentre dopo il passo 5 si ha la rappresentazione di visibilità di Figura 4.11, parte destra.

Figura 4.10.



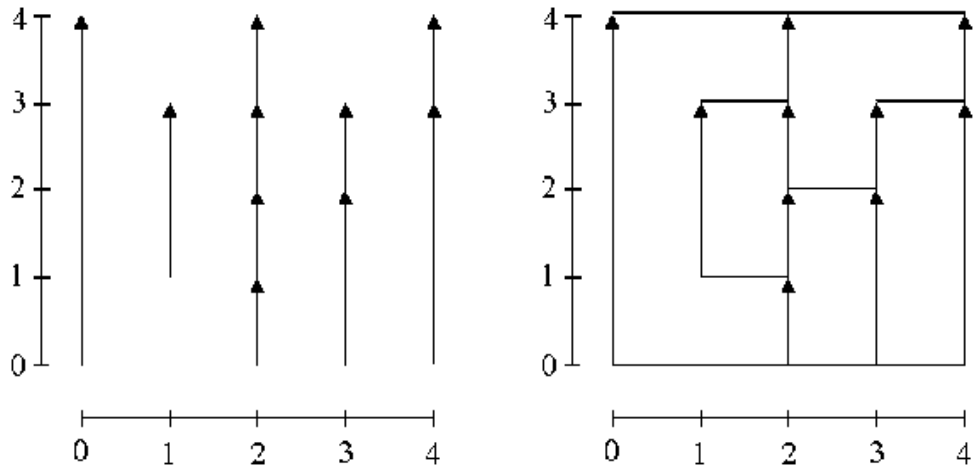
Teorema. *Sia G un st-grafo planare con n nodi, m archi ed f facce. Le dimensioni dell'area che delimita la rappresentazione Γ determinata dall'algoritmo precedente sono limitate da n-1 (altezza) ed f -1 ≤ 2n-5 (larghezza).*

Dimostrazione. Per come è progettato l'algoritmo, l'altezza e la larghezza della rappresentazione valgono rispettivamente quanto i massimi valori di $\alpha(v)$ e $\beta(f)$.

E' semplice verificare che il caso peggiore per entrambi i valori si ha quando il grafo G (per α e il grafo G^* per β) è un cammino. In questo caso la massima distanza possibile di un nodo da s è proprio $n-1$ così come quella da s^* è $f-1$ in G^*. Ne seguono le due limitazioni.

La disuguaglianza $f - 1 \leq 2n-5$ è già stata provata in relazione alla definizione di grafo planare massimale. **CVD**

Figura 4.11.



A questo punto, è possibile descrivere un algoritmo, dovuto a Tamassia e Tollis [TT89] che, preso in input un grafo planare con grado al più 4, sfruttando la sua rappresentazione di visibilità, ne determina una rappresentazione piana ortogonale con al più 2.4n+2 svolte, nessun arco con più di 4 svolte, lunghezza di ogni arco $O(n)$ e area totale $O(n^2)$. Se il grafo è 2-connesso si ottiene un limite migliore, cioè al più $2n+4$ svolte e solo 2 archi hanno più di 2 svolte.

L'algoritmo ha tre fasi. Nella prima fase viene costruita una rappresentazione di visibilità; nella seconda fase tale rappresentazione è trasformata in un disegno ortogonale planare su griglia; la terza fase modifica tale disegno in modo da ridurre il numero di svolte.

Prima fase: creazione della rappresentazione di visibilità

Precedentemente, è stato mostrato come determinare in tempo lineare la rappresentazione di visibilità di un *st*-grafo planare (quindi di un grafo 2-connesso). Se il grafo in input non è 2-connesso, non è possibile garantire l'esistenza di una rappresentazione di visibilità con un solo pozzo ed una sola sorgente; tuttavia, si può partizionare il grafo nelle sue componenti 2-connesse, determinare per ciascuna di esse una rappresentazione di visibilità, e riconnettere tali rappresentazioni, in modo che la rappresentazione risultante sia upward, abbia una sola sorgente ed un certo numero di pozzi. E' possibile garantire una sola sorgente se viene scelto un punto di articolazione come sorgente e se, in ciascuna componente 2-connessa, si ha l'accortezza di numerare con il più piccolo numero possibile uno degli adiacenti dei punti di articolazione. Il numero ed il grado delle sorgenti e dei pozzi di tale rappresentazione sono cruciali per la qualità del disegno risultante, ma non per la sua fattibilità.

Al termine di questo passo, si può dimostrare il seguente lemma:

Lemma: *Sia $G=(V,E)$ un grafo planare connesso di massimo grado 4. Allora, in $O(n)$ tempo può essere costruita una rappresentazione di visibilità $\Gamma(G)$ tale che $\Gamma(G)$ abbia solo una sorgente.*

Seconda fase: creazione di una rappresentazione piana ortogonale su griglia

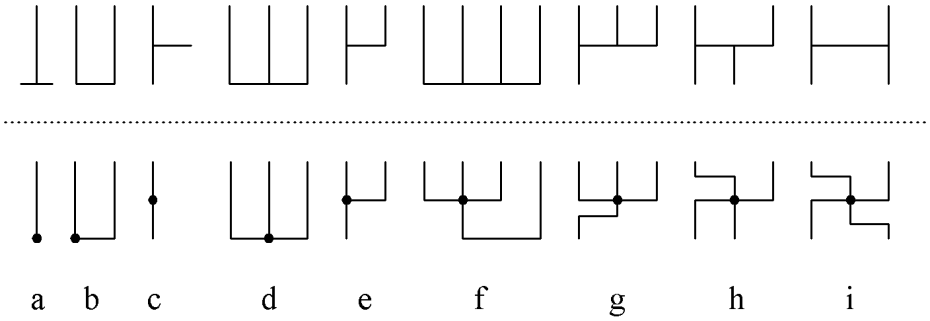
Si opera una trasformazione della rappresentazione di visibilità in una rappresentazione ortogonale. Ciò viene fatto sostituendo ogni segmento orizzontale della rappresentazione con una struttura formata da un singolo nodo e alcuni segmenti orizzontali e verticali.

Il tempo impiegato è $O(n)$.

Solo un numero costante di svolte vengono inserite per ogni segmento, per un totale di $O(n)$ svolte.

Ogni segmento orizzontale viene sostituito secondo le regole mostrate in Fig. 4.12, che rappresentano tutti i possibili casi a meno di simmetrie:

Figura 4.12.



Terza fase: riduzione delle svolte:

Nella fase precedente si considera un solo nodo per volta, e questo può portare all'inserimento di svolte inutili. In questa fase si tenta di rimuoverne alcune con una serie di passi di ottimizzazione locale.

Una *svolta* è considerata *convessa* se forma un angolo di 90° e concava se forma un angolo di 270°.

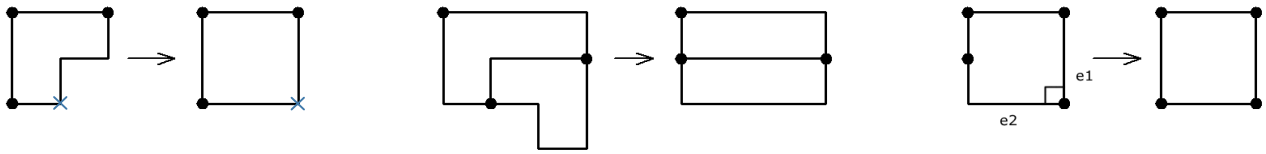
Ci sono 3 tipi di trasformazioni (vedi Figura 4.13):

T1: Se un arco *(u,v)* ha sia svolte concave che convesse, si rimuova una svolta di ciascun tipo finché l'arco non abbia solo svolte dello stesso tipo (N.B. il nodo in basso può trovarsi o a sinistra – pallino - o a destra - crocetta).

T2: Se tutti gli archi intorno ad un nodo hanno svolte dello stesso tipo, queste possono essere rimosse

T3: Se 2 archi e_1,e_2, che si susseguono in senso orario intorno ad un nodo v, formano un angolo di 90°, ed e_2 ha una svolta convessa rispetto a v, allora la svolta può essere rimossa.

Figura 4.13.

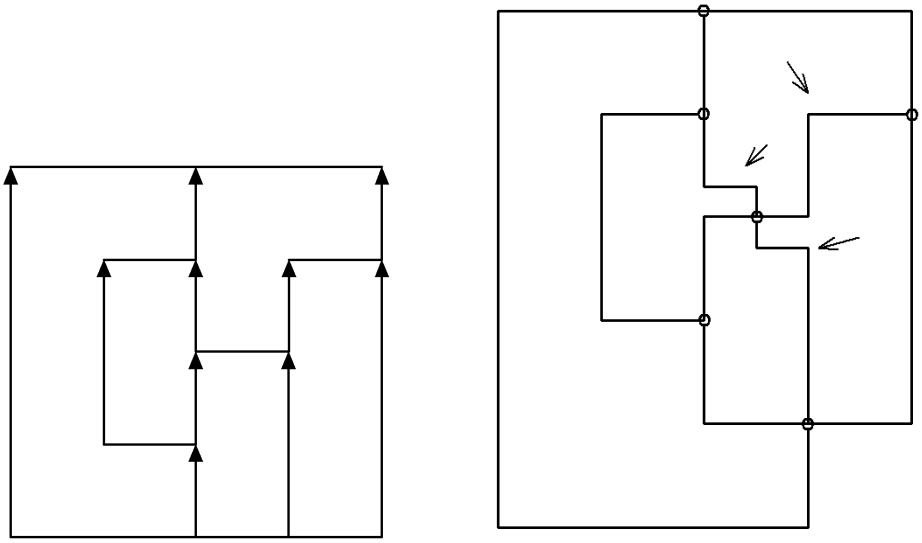


Le trasformazioni ora definite generano una nuova rappresentazione con meno svolte della precedente. In particolare, ogni trasformazione T1 riduce il numero di svolte di almeno 2, mentre ogni trasformazione di tipo T2 e T3 riducono tale numero di almeno 1. Ogni trasformazione può essere eseguita in tempo costante. Poiché ogni trasformazione rimuove almeno una svolta e le svolte introdotte al passo precedente erano $O(n)$, tutta questa fase può essere eseguita in tempo $O(n)$.

Esempio:

La rappresentazione di visibilità mostrata in Figura 4.14, parte sinistra, sia il risultato della prima fase dell'algoritmo. Trasformando i segmenti orizzontali secondo le regole della seconda fase si ottiene in output quanto mostrato in Figura 4.14, parte destra.

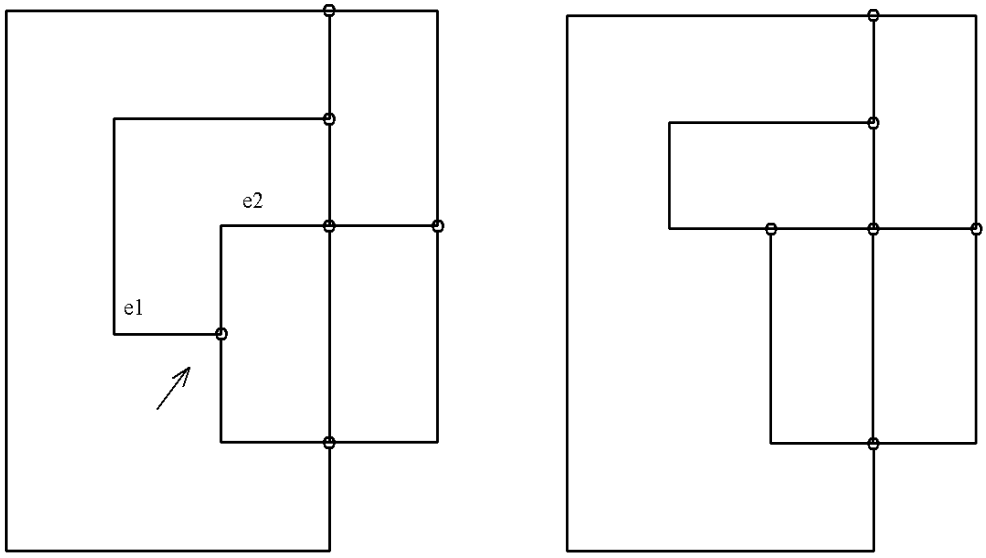
Figura 4.14.



Applicando la trasformazione T1 alle 3 svolte indicate con le frecce si ottiene la Figura 4.15, parte sinistra. Non può essere applicata la regola T2. Con la regola T3 si possono eliminare delle svolte intorno al nodo indicato con la freccia. Si ottiene quanto riportato in Figura 4.15, parte destra.

Si osservi che il disegno **non** è ottimo, ne' in termini di area ne' di numero di svolte.

Figura 4.15.



Questa sottosezione viene conclusa con l'analisi delle prestazioni dell'algoritmo descritto:

Lemma. *Se la rappresentazione di visibilità costruita nella Prima fase dell'algoritmo ha una sorgente, t_i pozzi ed n_i segmenti orizzontali non sorgente di grado i, i=2, 3, 4, allora il numero di svolte è al più $n_3+2n_4+t_2+2t_3+4t_4+\delta$, dove δ=0, 1, 2, 4, a seconda che la sorgente abbia grado 1, 2, 3 o 4.*

Dimostrazione. Sia n_4' il numero di nodi risultanti dalle sostituzioni di tipo *g* e *h* di Figura 4.4, e sia n_4'' il numero di nodi risultanti dalla sostituzione di tipo *i*. Dopo le sostituzioni dei segmenti orizzontali, il numero delle svolte è dato da: $n_3+4n_4'+6n_4''+t_2+2t_3+4t_4+\delta$. Successivamente, la riduzione T1 è applicata almeno una volta per ogni nodo di tipo *g* e *h*, e almeno due volte per ogni nodo di tipo *i*. Questo significa che almeno $2n_4'+4n_4''$ svolte vengono eliminate. Quindi l'equazione precedente porta ad un numero di svolte pari a $n_3+2n_4'+2n_4''+ t_2+2t_3+4t_4+\delta \leq n_3+2n_4+ t_2+2t_3+4t_4+\delta$. A causa delle riduzioni T2 e T3 il risultato può solo migliorare. **CVD**

NB questo risultato e' peggiore di quello enunciato sull'articolo ma la dimostrazione è più chiara

Teorema: *Sia G un grafo planare connesso con n nodi di grado massimo 4. Allora l'algoritmo presentato produce una rappresentazione planare e ortogonale su griglia di G con al massimo 2n+4 svolte se G è 2-connesso, e 2.4n+2 svolte se è semplicemente connesso. L'area è* $O(n^2)$ *e il tempo di esecuzione è O(n).*

Idea della Dimostrazione: Nel caso 2-connesso, la limitazione segue dal lemma precedente considerando che la rappresentazione di visibilità ha un solo pozzo (nel caso peggiore l'unico pozzo ha grado 4) e il grado della sorgente è 4. Inoltre bisogna tener presente che $n_1+n_2+n_3+n_4=n-2$ (dal numero di nodi si sottraggono la sorgente e il pozzo, conteggiati a parte), da cui si deduce che $n_3+2n_4 \leq 2(n_1+n_2+n_3+n_4)=2(n-2)$.

Nel caso semplicemente connesso, è possibile mostrare che $t_3+2t_4 \leq m/5$ a meno che il grafo non appartenga ad una certa famiglia di grafi. Con l'aiuto di questa limitazione e del precedente lemma si giunge alla tesi.

Il tempo di esecuzione è ovviamente lineare, a causa di tutte le considerazioni fatte durante la spiegazione dell'algoritmo. Per la dimostrazione completa, si rimanda a [TT89]. **CVD**

Vedere i dettagli di questa dimostrazione

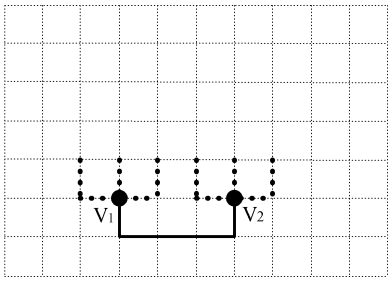
4.1.4. Un algoritmo basato sulla st-numerazione

Riferimenti: [BK94] pp.1-5 (ecluso Lemma 3.4); [KW98] pp.137-141.

In questa sottosezione sarà descritto un algoritmo per determinare una rappresentazione ortogonale su griglia basato sulla st-numerazione [BK94]. Poiché l'algoritmo precedente lavora su grafi planari, è stato scelto il seguente algoritmo, che lavora anche su grafi non planari.

L'idea dell'algoritmo consiste nel calcolare una st-numerazione per il grafo in input, quindi nell'inserire sulla griglia, uno dopo l'altro, i nodi in ordine di st-numerazione. Più precisamente, per ciascun nodo da inserire si aggiunge al disegno una nuova riga; tale nodo ha $d \leq 4$ archi ad esso incidenti, altrimenti non potrebbe essere rappresentato su griglia; alcuni dei suoi vicini (almeno uno e al più *d-1*, per la definizione di st-numerazione) sono già presenti nel disegno, mentre altri (almeno uno e al più *d-1*, per la definizione di st-numerazione) devono ancora essere inseriti, perché hanno un st-numero maggiore. D'ora in poi, fissato un nodo *v*, gli archi che connettono *v* con un nodo avente st-numero maggiore saranno detti *archi uscenti* da *v*, mentre gli archi che connettono *v* con nodi aventi st-numero minore saranno detti *archi entranti* in *v*. Per ciascun arco uscente dal nodo corrente, che quindi ha ancora un solo estremo nel disegno, si aggiunge una colonna a destra o a sinistra della porzione esistente del disegno. Segue il dettaglio dell'algoritmo:

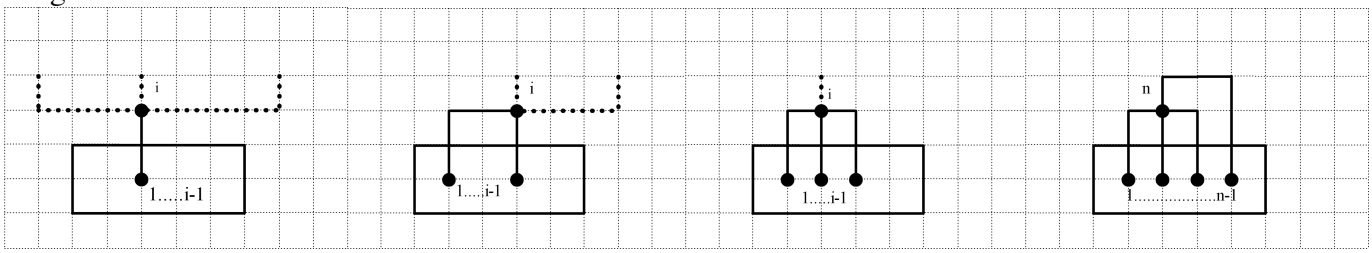
Figura 4.16.



Input: Grafo G biconnesso con grado massimo 4;
Output: Disegno ortogonale su griglia di G.

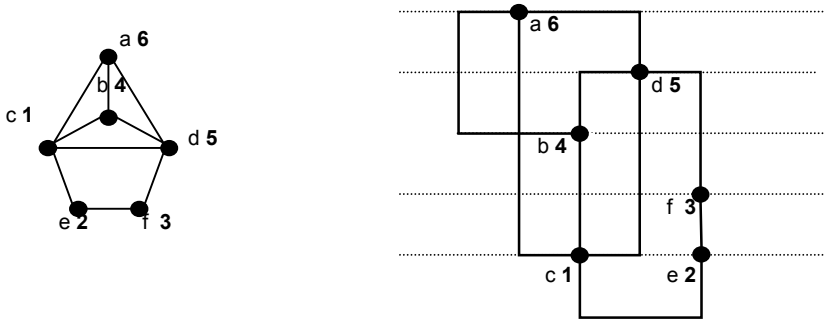
- calcola una st-numerazione per G a partire da due nodi adiacenti s e t qualunque; i nodi possono quindi essere ordinati come $s=v_1, v_2, \ldots, v_n=t$, dove v_i ha numerazione i;
- metti i nodi v_1 e v_2 nella griglia e connetti tali nodi come mostrato in figura 4.16;
- alloca una colonna nella griglia per ciascun arco di v_1 e v_2, tranne per l'arco che connette v_1 a v_2, per il quale è già stata allocata una nuova riga.
- FOR $3 \leq i \leq n-1$ DO
  - metti v_i su una nuova riga e su una colonna già assegnata ad un arco non completo entrante in v_i; se possibile, non utilizzare la colonna più a sinistra o quella più a destra tra quelle disponibili;
  - disegna tutti gli archi entranti in v_i utilizzando le colonne già assegnate;
  - alloca delle colonne per gli archi uscenti da v_i a destra o a sinistra del disegno esistente, come mostrato nelle figure 4.17.a, .b e .c;
- metti v_n su una nuova riga e su una colonna già assegnata ad un arco non completo entrante in v_n; se possibile, non utilizzare la colonna più a sinistra o quella più a destra tra quelle disponibili;
- disegna tutti gli archi entranti in v_n come mostrato in figura 4.17.d.

Figura 4.17.



a. b. c. d.

Figura 4.18.



(a) (b)

Esempio. In Fig. 4.18 è mostrato un esempio del funzionamento dell'algoritmo ora descritto. I nodi sono etichettati con una lettera che li identifica e con un numero che corrisponde al loro st-numero.

Calcoliamo ora l'area del disegno ed il numero di svolte, ricordando la nota relazione:
$\sum_{v\in V}$ indeg(v)= $\sum_{v\in V}$ outdeg(v)=m.

Teorema. *L'algoritmo precedente determina un disegno ortogonale su griglia di un grafo con grado al più 4 con area al più (m-n+1) × n.*

Dimostrazione. Calcoliamo prima l'altezza e poi la larghezza del disegno.

Per quanto riguarda l'altezza, si osservi che si utilizzano due linee della griglia per disegnare i nodi v_1 e v_2, quindi partiamo da un'altezza unitaria. Ciascun nodo v_i, $i=2, \ldots, n-1$, incrementa l'altezza di 1. Il nodo v_n incrementa l'altezza di 1, se ha grado non superiore a 3; di 2, se ha grado = 4.

Sommando tutti i contributi si ottiene un'altezza pari a $1+(n-3)+2 = n$ nel caso peggiore.

Per quanto riguarda la larghezza, si osservi che i nodi v_1 e v_2 introducono complessivamente $outdeg(v_1)-1 + outdeg(v_2)-1$nuove colonne. I nodi da v_3 a v_{n-1} introducono $outdeg(v_i)-1$ colonne ciascuno. Infine, il nodo v_n, che ha grado uscente pari a 0, introduce $outdeg(v_n)-1+1=0$ colonne.

Sommando i contributi, otteniamo:

$\sum_{i=1\ldots n} (outdeg(v_i)-1)+1=\sum_{i=1\ldots n}outdeg(v_i)-\sum_{i=1\ldots n} 1+1=m-n+1.$ **CVD**

Teorema. *L'algoritmo precedente produce un disegno del grafo in input in cui il massimo numero di svolte è 2m-2n+4 e tutti gli archi hanno al più due svolte, tranne uno che può averne tre.*

Dimostrazione. L'inserimento di v_1 e v_2 aggiunge al disegno $deg(v_1)+deg(v_2)-2=6$ svolte; l'inserimento dei nodi v_i, $i=2, \ldots, n-1$, aggiunge al disegno $deg(v_i)-2$ svolte; infine, v_n introduce $deg(v_n)$ svolte se $deg(v_n)=4$, e $deg(v_n)-1$ svolte negli altri casi.

Sommando i contributi, otteniamo:

$deg(v_1)+deg(v_2)-2+\sum_{i=3\ldots n-1}(deg(v_i)-2)+deg(v_n)=\sum_{i=1\ldots n}deg(v_i)-\sum_{i=1\ldots n}2+4=2m-2n+4$ se $deg(v_n)=4$, ed $=2m-2n+3$ negli altri casi.

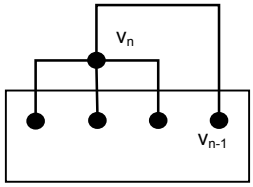
Ogni arco (v_i, v_j), $i<j\neq n$, viene piegato al più una volta al momento dell'inserimento di v_i, e al più un'altra volta quando viene inserito v_j. Se $j=n$, un solo arco può essere piegato al più due volte, e questo è il motivo per cui un solo arco può avere 3 svolte. **CVD**

Teorema. *L'arco con tre svolte si può sempre evitare.*

Dimostrazione. Si consideri un grafo con meno di $2n$ archi, cioè un grafo in cui non tutti i nodi abbiano grado 4. Allora esiste almeno un nodo di grado inferiore a 4, e questo può essere scelto come nodo t. L'arco con 3 svolte viene così evitato.

Se il grafo è 4-regolare, procediamo come mostrato in figura 4.19: al momento di disegnare v_{n-1}, assegnamo la direzione centrale (arco senza svolte) all'arco che congiunge v_{n-1} e v_n; scegliamo quindi proprio quest'ultimo arco come quello a cui aggiungere le due svolte. Risulta anche in questo caso che ogni arco ha al più 2 svolte. **CVD**

Figura 4.19.



E' semplice convincersi che il risultato sull'area garantita dal precedente algoritmo può essere migliorato in quanto questo algoritmo usa una nuova riga per ogni nodo che viene inserito e una nuova colonna per ogni arco che viene inserito. Questo è l'argomento di un lavoro di Papakostas e Tollis [PT97] di cui forniremo l'idea, senza entrare nei dettagli.

L'algoritmo è simile a quello di Biedl e Kant appena visto, ma cerca di diminuire il numero di righe e colonne utilizzate per disegnare il grafo. L'idea centrale dell'algoritmo consiste nel formare delle coppie di nodi che, per qualche ragione, possono giacere sulla stessa riga o sulla stessa colonna. Ci sono due differenti tipi di coppie:

coppie di riga: i due nodi della coppia sono piazzati in modo da riusare una riga nel disegno finale;

coppie di colonna: i due nodi della coppia sono piazzati in modo da riusare una colonna nel disegno finale.

Ad ogni nodo di G viene assegnato un tipo: Un nodo con a archi entranti e b archi uscenti è chiamato nodo di tipo a-b o nodo a-b ($1 \le a, b \le 4$). Se esiste un nodo *1-1* tale che l'arco uscente sia entrante in un nodo di tipo *1-2* o *1-3*, tale nodo viene rimosso e si crea un nuovo arco tra il suo predecessore e il suo successore. Questo nodo che è stato rimosso può essere inserito nel disegno, come punto dell'arco creato, al termine dell'algoritmo senza avere effetto ne' sull'area e ne' sul numero di svolte. Il grafo G' ottenuto rimuovendo questi nodi ha $n' \le n$ nodi. Per formare le coppie, i nodi sono considerati nell'ordine inverso rispetto all'ordinamento della st-numerazione di G. Se si incontra un nodo *1-2* o *1-3*, esso viene accoppiato con il suo immediato predecessore nell'ordinamento della st-numerazione di G. Se si incontra un nodo *2-2*, esso è accoppiato con il nodo successivo che non sia di tipo *1-1*, *2-1* o *3-1* oppure con un suo predecessore. Dopo questo passo, tutti i nodi v_i di tipo *1-2*, *1-3* e *2-2*, con $3 \le i \le n$, sono accoppiati. A questo punto, i nodi vengono inseriti nella griglia in accordo con l'ordinamento della st-numerazione di G. I nodi che non sono stati ancora inseriti e che non sono accoppiati seguono le regole dell'algoritmo di Biedl e Kant. Gli altri vengono inseriti insieme al nodo con il quale costituiscono la coppia.

In conclusione, l'algoritmo può essere così descritto:

Input: un grafo G di grado massimo *4*;
Output: un disegno ortogonale su griglia di G;

- calcola una st-numerazione per G a partire da due nodi adiacenti s e t qualunque;
- calcola un accoppiamento di nodi;
- FOR $i=1$ TO n DO
  IF i non è stato già inserito
  THEN
  IF i non è accoppiato
  THEN inserisci come Biedl-Kant
  ELSE (i e j, $j>i$, sono nella stessa coppia di riga/colonna)
  inserisci i e j nella stessa riga/colonna.

E' possibile dimostrare che il numero di nodi non accoppiati è $\le \lfloor (n+2)/2 \rfloor$. Detto k_1 e k_2 il numero di nodi che non allocano una nuova riga o una nuova colonna, rispettivamente, segue che $k_1 + k_2 \ge \lceil (n-2)/4 \rceil$. Osservando che l'area viene massimizzata quando $k_1 = k_2$, e ripercorrendo ragionamenti simili a quelli fatti per determinare l'area massima occupata dal disegno fatto con l'algoritmo di Biedl e Kant, è possibile dimostrare che l'area prodotta con questo algoritmo è limitata da $0.77\, n^2$. Anche l'analisi per il numero di svolte è simile, e dà come risultato $2n+4$.

La complessità computazionale dell'algoritmo è O(n), a patto di usare delle strutture dati piuttosto sofisticate, dovute a Dietz e Sleator [DS87].

RIFERIMENTI BIBLIOGRAFICI

[BK94] T.C. Biedl, G. Kant: A Better Heuristic for Orthogonal Graph Drawing. *Proc. 2nd European Symp. On Algorithms (ESA '94)*, Lecture Notes in Computer Science 855 24-35, 1994. (anche UU-CS-1995-04).

[CN??] Chiba, Nishizeki : Planar graphs: Theory and Algorithms, 19??.

[Dal99] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis*: Graph Drawing – Algorithms for the visualization of graphs*, Prentice Hall, 1999.

[DT88] G. Di Battista, R. Tamassia: Algorithms for plane representations of acyclic digraphs, *Theoretical computer science* 61, 175-198, 1988.

[DS87] P.F. Dietz, D.D. Sleator: Two Algorithms for maintaining order in a list, *Proc. 19th Annual ACM Symp. Of Theory of Comput. (STOC '87)*, 365-372, 1987.

[ET76] S. Even, R.E. Tarjan: Computing an st-Numbering. *Theoret. Comput. Sci.* 2, 339-344, 1976.

[Fal90] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F.T. Leighton, A. Simvonis, E. Welzl, G. Woeginger: Drawing Graphs in the Plane with High Resolution. *Proc. of the 31th Symp. On the Foundation of Computer Science (FOCS'90)*, 86-95, 1990.

[KW98] M. Kaufmann, D. Wagner (Eds.): *Drawing Graphs – Methods and Models*. Lecture Notes in Computer Science 2025, Springer 1998.

[OW78] R.H.J.M. Otten, J.G. van Wijk: Graph Representation in Interactive Layout Design. *IEEE Int.l Symp. On Circuits and Systems*, 914-918, 1978.

[PT97] A. Papakostas, I.G. Tollis: A Pairing Technique for area-efficient orthogonal drawing. *Proc. 4th Int.l Symp. On Graph Drawing (GD '96)*, Lecture Notes in Computer Science 1190, 354-370, 1997.

[RT86] P. Rosenstiehl, R.E. Tarjan: Rectilinear Planar Layouts of Planar Graphs and Bipolar Orientations. *Discrete & Combinatorial Geometry*, no. 1(4), 342-351, 1986.

[T87] R. Tamassia: On Embedding a Graph in the Grid with the Minimum Number of Bends. *SIAM Journal on Computing*, 16(3), 421-444, 1987.

[TT89] R. Tamassia, J. Tollis: Planar Grid Embedding in Linear Time. *IEEE Trans. On Circuits and Systems*. 36(9), 1230-1234, 1989.