8. VISUALIZZAZIONE DI TOPOLOGIE DI INTERCONNESSIONE

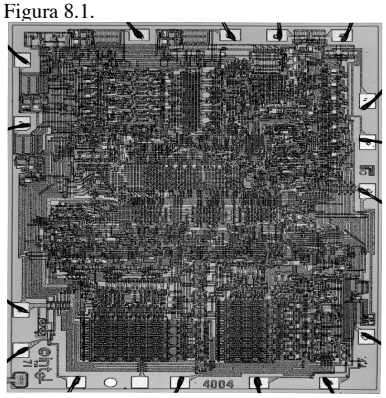
Riferimenti: [L92] per la definizione di butterfly, [W81] solo disegno e osservazioni su di esso, [EE00] tutto esclusa sez. 3.3.

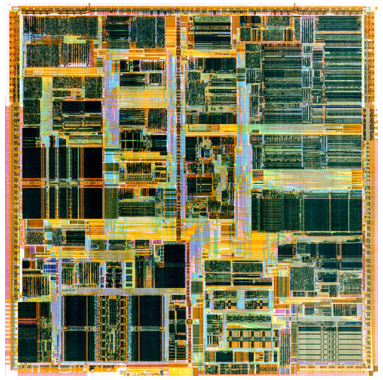
8.1. Il modello di Thompson per il layout; vincoli dettati dalla tecnologia VLSI

Data una rete di interconnessione, si definisce *topologia di interconnessione* il grafo soggiacente questa rete.

Il *problema del layout* di topologie di interconnessione nasce dal problema di stampare circuiti VLSI (Very Large Scale Integration) su un supporto di silicio. Esso ebbe origine negli anni '40, ma ha ottenuto un significativo studio solo in tempi relativamente recenti, quando la tecnologia ha consentito di stampare circuiti in due e tre dimensioni a prezzi ragionevoli. In Figura 8.1. sono rappresentati due microprocessori: a sinistra l'Intel 4004 e a destra l'Intel Pentium.

Figura 8.1.





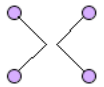
La tecnologia di produzione VLSI impone molti vincoli al layout [MC80]; in particolare, bisogna tenere conto:

- che la macchina che traccia i fili può solo approssimare segmenti obliqui con sequenze di segmenti orizzontali e verticali (⇒ disegno ortogonale);
- che per evitare interferenze bisogna garantire che i fili siano ad una certa distanza l'uno dall'altro(⇒ disegno su griglia);
- che i fili non si possono incrociare; per evitare gli incroci si possono far viaggiare i fili uno su una faccia e uno sull'altra, introducendo dei "buchi" per permettere il passaggio dei fili da una faccia all'altra; il numero di tali fori deve comunque essere contenuto poiché il costo di realizzazione è elevato (⇒ minimizzazione del numero di incroci);
- che i fili non devono essere troppo lunghi in quanto il ritardo di propagazione è proporzionale alla loro lunghezza e che, in caso di struttura a livelli, i fili dello stesso livello dovrebbero avere approssimativamente la stessa lunghezza, in modo da evitare problemi di sincronizzazione (⇒ minimizzazione della lunghezza degli archi);
- che, di tutto il processo di produzione, il materiale è la cosa più costosa; per questo motivo è necessario che il layout abbia area minima(⇒ minimizzazione dell'area).

Thompson [T??] ha sviluppato un modello formale per il layout di topologie di interconnessione che si basa sulle regole di progettazione ora descritte ed è consistente con esse.

Nel modello di Thompson, il layout di una topologia G è una rappresentazione su un insieme di tracce orizzontali e verticali a distanza unitaria l'una dall'altra che mappa i nodi del grafo nei punti di intersezione delle tracce, e gli archi del grafo in percorsi disgiunti costituiti da segmenti di tracce orizzontali e verticali; tali percorsi non possono intersecare nodi a loro non adiacenti e possono avere intersezioni tra loro solo in corrispondenza degli incroci; in altre parole, non sono permesse sovrapposizioni, non sono permessi incroci arco-nodo, non sono permessi "knock-knees" (vedi Figura 8.2).

Figura 8.2.



Ne segue che il layout è un disegno ortogonale su griglia per cui si richiede la minimizzazione dell'area, del numero di incroci e della lunghezza degli archi. Intuitivamente, si potrebbe allora pensare di utilizzare gli algoritmi già visti per disegnare in modo ortogonale su griglia un grafo; tuttavia questa tecnica non funzionerebbe in modo soddisfacente: gli algoritmi per disegnare grafi garantiscono certe limitazioni delle funzioni di ottimizzazione per OGNI grafo dato in input che soddisfi i requisiti necessari; le topologie di interconnessione, però sono grafi con forte struttura (tipicamente regolari, molto simmetriche, con proprietà ricorsive, …) e, sfruttando queste proprietà, si possono ottenere risultati migliori. Per questo, se gli algoritmi per disegnare grafi prendono in input un grafo e lo disegnano, gli algoritmi per il layout sono progettati per una peculiare topologia di interconnessione e, perciò, ne prendono in input la dimensione.

Infine, è da sottolineare che l'ottimizzazione al meglio delle funzioni di ottimizzazione (in special modo l'area), anche nelle costanti, è un obiettivo assai più importante che nel caso di disegni di grafi per quanto già osservato: se un layout viene fatto nella metà dell'area rispetto ad un altro, il suo costo sarà dimezzato!

8.2. Diversi layout di una stessa topologia di interconnessione (butterfly)

Definizione. Sia $N=2^n$ (e quindi $n=\log N$); una Butterfly n-dimensionale (vedi Figura 8.3) è un grafo livellato con $N(n+1)$ nodi (n+1 livelli ognuno con 2^n nodi) e $2Nn$ archi, i cui nodi sono etichettati con una coppia (i, w), dove i è il *livello* del nodo e w è un numero binario di n bit che indica la riga del nodo.

Due nodi (i,w) e (i', w') sono adiacenti se e solo se $i'=i+1$ e:

- $w=w'$ (*straight edge*) o
- w e w' differiscono esattamente nell'i-esimo bit (*cross edge*).

I nodi della Butterfly sono *crossbar switches*, cioè switches che hanno due ingressi e due uscite e possono assumere due stati (illustrati in Figura 8.4), cross e bar.

Possiamo quindi vedere la butterfly come una switching network che connette $2N$ ($N=2^n$) unità di input a $2N$ unità di output tramite una rete di $\log N+1$ livelli di N nodi ciascuno.

La butterfly ha una struttura ricorsiva come evidenziato nella parte destra della Figura 8.3: una butterfly n-dimensionale contiene come sottografi due butterfly (n-1)-dimensionali che si possono ottenere rimovendo i nodi di livello n.

Figura 8.3.

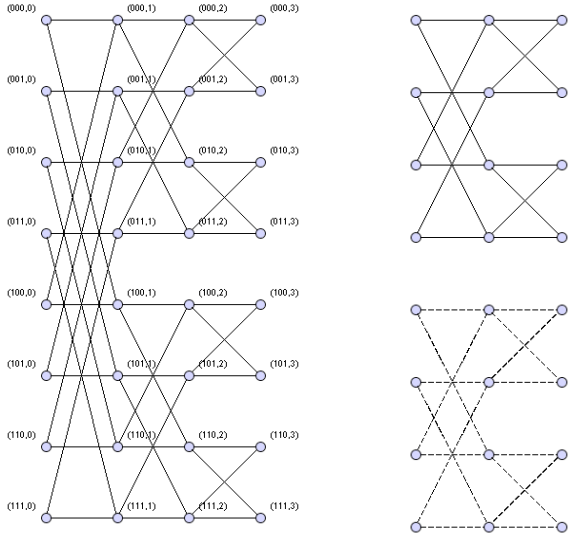
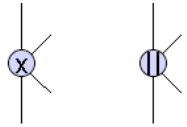


Figura 8.4.

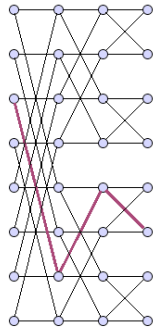


Un'altra proprietà della butterfly *n*-dimensionale è che, per ogni coppia di righe *w* e *w'*, esiste uno ed un solo cammino di lunghezza *n* che collega un nodo di livello 0 sulla riga *w* e uno di livello *n* su una riga *w'*, questo cammino attraversa ogni livello esattamente una volta, usando un cross edge dal livello *i* al livello *i*+1 (*i*=0,…,*n*) se e solo se *w* e *w'* differiscono nell' i-esimo bit (vedi Figura 8.5).

8.2.1. Layout 'slanted'

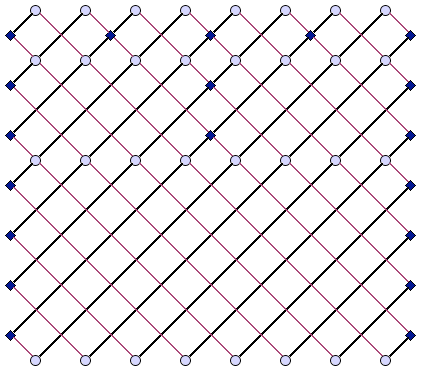
Un layout, proposto da D.S.Wise [W81], per la butterfly, è quello riportato in figura 8.6; questo ha la proprietà di avere tutti i fili di uno stesso livello della stessa lunghezza, dove la lunghezza cresce esponenzialmente con il livello, quindi, sebbene ci siano solo *logN+1* livelli, da un nodo di input ad un nodo di output, la lunghezza del cammino è lineare in *N*. Si può, infatti, facilmente dimostrare che ogni cammino è lungo *2(N-1)*: consideriamo per semplicità il percorso dal nodo più in alto a sinistra a quello più in basso a destra, la lunghezza di questo cammino può essere calcolata come la diagonale di un quadrato di lato $\sqrt{2}(N-1)$, la lunghezza di tale cammino è quindi uguale a *2(N-1)*.

Figura 8.5.



Il layout presentato è formato da due strati, stampati sulle due facce della lastra di silicio; su uno si trovano le linee che vanno da in alto a sinistra a in basso a destra (linee viola), sull'altro si trovano i fili che vanno da in alto a sinistra a in basso a destra (linee nere), si può pensare a questi due strati come alle due facce della lastra di supporto; questo posizionamento dei fili in due strati è necessario per evitare gli incroci tra i fili. Alcuni fili formano dei "knock-knees", che non sono permessi; affinché la prossimità tra questi fili non causi interferenze è necessario inserire dei dispositivi speciali, rappresentati come dei piccoli quadrati in Figura 8.6.

Figura 8.6.



La dimensione del layout ottenuto è $\sqrt{2}(N-1) \times \sqrt{2}(N-1) = 2N^2+o(N^2)$.

Questo layout presenta numerose proprietà, alcune vantaggiose ed altre meno, che evidenziamo qui di seguito:

Vantaggi

- l'area è molto buona;
- la lunghezza dei fili è costante su ogni livello; osserviamo che questo non è vero in ogni layout: nel disegno classico della butterfly, ad esempio, sull'ultimo livello, gli straight-

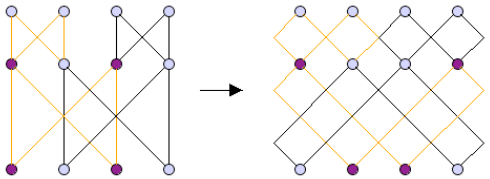
edges hanno lunghezza unitaria, mentre i cross-edges hanno lunghezza lineare in N; questa è una proprietà estremamente negativa perché si perde il sincronismo del flusso delle informazioni;

- i nodi di input ed output giacciono sul contorno, che può essere richiesto da alcune applicazioni.

Svantaggi

- le linee sono "slanted", quindi l'area misurata non è quella del rettangolo con lati paralleli agli assi cartesiani, ma con i lati a 45°; seguendo la definizione standard di area essa risulta 2(N-1) $\times$ $2(N-1)= 4N^2+o(N^2)$, infatti il quadrato circoscritto al layout con lati paralleli alle linee della griglia ha lato di lunghezza pari al cammino dal nodo più in alto a sinistra a quello più in basso a destra e cioè $2(N-1)$;
- non è un layout 'pulito', poiché i "knock-knees" non vengono evitati ma sistemati con dei dispositivi che, oltre tutto, non hanno area nulla, quindi il loro inserimento causa un peggioramento delle dimensioni del layout;
- per ottenere il layout proposto bisogna permutare l'ordine dei nodi, come evidenziato in Figura 8.7.

Figura 8.7.



8.2.2. Layout tramite layered cross product

Per cercare di eliminare tutti gli svantaggi del layout ora descritto sono stati proposti altri metodi. Qui presentiamo quello dovuto ad Even ed Even [EE00], basato sulla seguente definizione di Layered Cross Product.

Ricordiamo che un *grafo livellato con $l+1$ livelli* $G=(V_0, V_1, \ldots, V_l, E)$ consiste di $l+1$ livelli di nodi; V_i è l'insieme (non vuoto) di nodi di livello i; E è l'insieme degli archi: ogni arco (u,v) connette due nodi di livelli adiacenti, cioè, se u è al livello i allora v è al livello $i+1$.

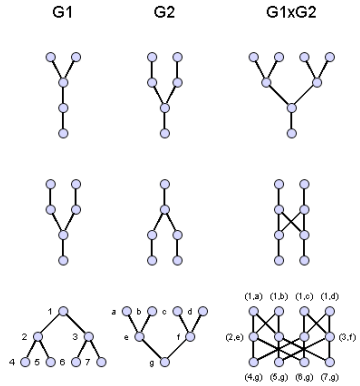
Definizione. [EL92] Il *Layered Cross Product* (*LCP*) di due grafi livellati di $l+1$ livelli, $G^1=(V_0^1, V_1^1, \ldots, V_l^1, E^1)$ e $G^2=(V_0^2, V_1^2, \ldots, V_l^2, E^2)$, è un grafo livellato con $l+1$ livelli $G=(V_0, V_1, \ldots, V_l, E)$ tale che:

- per ogni $i=0, \ldots, l$ $V_1 = V_1^1 \times V_1^2$ (ogni livello è il prodotto cartesiano dei livelli dei grafi fattore);
- esiste un arco (u,v) che connette i nodi (u^1,u^2) e (v^1,v^2) se e solo se (u^1,v^1) ed (u^2,v^2) sono archi dei grafi fattore.

In Figura 8.8 sono mostrati alcuni esempi di LCP.

Even e Litman [EL92] hanno dimostrato che molte topologie di interconnessione ben note sono il risultato del prodotto di semplici grafi livellati come gli alberi. In particolare, la butterfly è il LCP di due alberi binari completi uno con la radice posta in alto, l'altro con la radice posta in basso (Figura 8.8, parte bassa).

Figura 8.8.

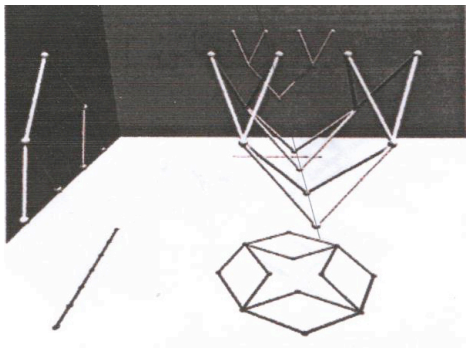


Possiamo sfruttare questa proprietà per produrre un disegno per la butterfly usando il *metodo di proiezione* proposto da G.Even e S.Even [EE00] descritto nel seguito.

Siano G^1 e G^2 due grafi livellati di $l+1$ livelli e sia G il loro LCP. Un disegno bidimensionale di G si ottiene col metodo di proiezione nel modo seguente (vedi Figura 8.9):

- Considera lo spazio cartesiano xyz;
- Disegna sul piano xy il grafo G^1 e sul piano yz il grafo G^2 in modo tale che la coordinata y di ogni nodo di livello i sia uguale a i, le altre coordinate siano interi;
- Costruisci un disegno 3D di G come segue: se $u \in V_i^1$ è disegnato alle coordinate $(x_u, i, 0)$ e $v \in V_i^2$ è disegnato alle coordinate $(0, i, z_v)$, allora le coordinate del nodo (u,v) sono (x_u, i, z_v), cioè i nodi di G sono i punti di intersezione delle rette perpendicolari al piano xy passanti per i nodi di G^1 e delle rette perpendicolari al piano yz passanti per i nodi di G^2.
- Il disegno 2D di G si ottiene proiettando il disegno 3D sul piano xz

Figura 8.9.

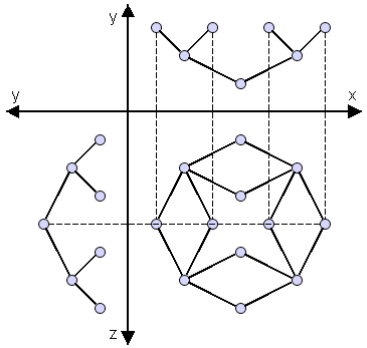


Si può evitare di passare per la rappresentazione 3D, infatti i nodi del LCP coincidono con le intersezioni dei prolungamenti sul piano *xz* delle proiezioni dei nodi di livello *i* di G^1 sull'asse *x* e dei nodi di livello *i* di G^2 sull'asse *z* per *i=0,…,l* (vedi Figura 8.10). Nel seguito sarà adottata questa tecnica semplificata.

Come si può vedere dalla Figura 8.10, il metodo di proiezione può produrre rappresentazioni che non rispettano i vincoli dettati dal layout di topologie, infatti il disegno costruito pur essendo su griglia non è ortogonale.

Analizziamo i diversi tipi di archi risultanti dall'LCP; ciò ci permetterà di imporre le condizioni necessarie e sufficienti affinché il metodo di proiezione produca archi che corrono lungo le linee della griglia, nodi mappati in punti distinti della griglia e archi che non si sovrappongono.

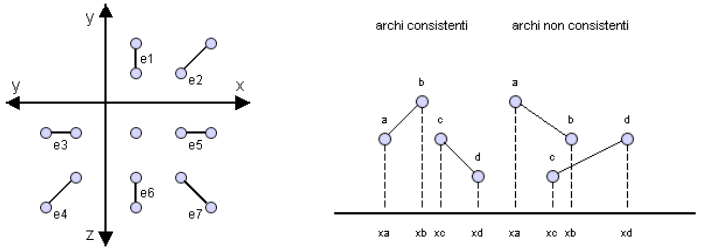
Figura 8.10.



Come mostra la Figura 8.11 parte sinistra, ci sono quattro tipi di archi nel grafo prodotto *G*:

1. il prodotto di due archi obliqui dà come risultato un arco obliquo;
2. il prodotto di un arco verticale ed uno obliquo dà come risultato un arco verticale;
3. il prodotto di un arco obliquo ed uno verticale dà come risultato un arco orizzontale;
4. il prodotto di due archi verticali mappa due nodi distinti in uno stesso punto della griglia.

Figura 8.11.



Per ottenere un layout ammissibile tramite il metodo di proiezione bisogna imporre che non ci siano archi generati dal prodotto di due archi obliqui o di due archi verticali. Più precisamente, possiamo formalizzare al modo seguente:

Affermazione. *Il metodo di proiezione genera un layout di G i cui archi giacciono su linee della griglia se e solo se i disegni di G^1 e G^2 soddisfano la seguente condizione: per ciascun arco di G, esattamente uno dei suoi fattori è obliquo.*

Questa affermazione previene anche le sovrapposizioni di nodi nell'ambito di ciascun livello. Tuttavia, è necessario imporre anche che nodi di diversi livelli non si sovrappongono. Per fare questo, è sufficiente che sia verificata la seguente:

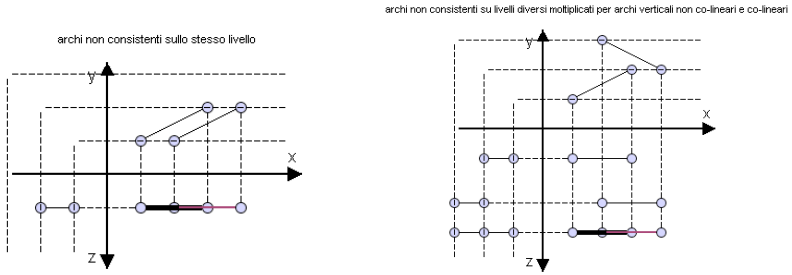
Affermazione. *Il metodo di proiezione genera un layout di G in cui al più un nodo è mappato su ogni punto della griglia se e solo se, per ogni i=0, 1, …, l, gli insiemi $\{(x_u, z_v)$: $u \in V_i^1$ e $v \in V_i^2\}$ sono disgiunti.*

Si considerino ora due archi obliqui *(a,b)* e *(c,d)* in G^1; i quattro nodi hanno coordinate:
nodo *a*: $(x_a, i, 0)$; nodo *b*: $(x_b, i+1, 0)$; nodo *c*: $(x_c, j, 0)$; nodo *d*: $(x_d, j+1, 0)$.
I due archi si dicono *consistenti* se e solo se gli intervalli aperti (x_a, x_b) e (x_c, x_d) sono disgiunti, vedi Figura 8.11 parte destra.

Affermazione. Il metodo di proiezione genera un layout di G in cui nessun arco di griglia viene usato due volte se e solo se per ogni coppia di archi inconsistenti in G^1 (G^2):

- i due archi non sono sullo stesso livello (vedi Figura 8.12 parte sinistra), e
- sui due livelli su cui si trovano non esistono due archi co-lineari in G^2 (G^1) (vedi Figura 8.12 parte destra).

Figura 8.12.



Affinché il metodo di proiezione produca un layout ammissibile del grafo *G* dobbiamo quindi disegnare i due grafi fattori in modo che siano rispettate le tre affermazioni appena enunciate. In particolare, per seguire la **prima** affermazione, introduciamo nel disegno dei due grafi fattori dei nodi e degli archi fittizi (vedi Figura 8.13) in modo tale che:

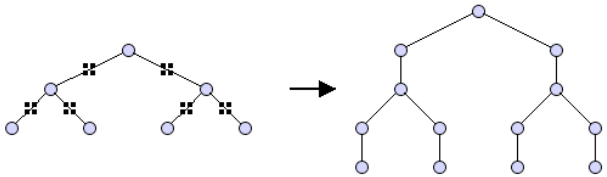
- ogni arco sia spezzato in due archi uno obliquo e uno verticale;
- nel disegno di G^1 gli archi obliqui siano nei livelli dispari e quelli verticali nei livelli pari;
- nel disegno di G^2 gli archi obliqui siano nei livelli pari e quelli verticali nei livelli dispari.

In questo modo viene simulata la creazione delle svolte.

Per fare in modo che la **seconda** affermazione sia vera, bisogna fare in modo che non ci siano due nodi, esclusi gli estremi di un arco verticale, nel disegno di ciascun fattore, che condividano la stessa coordinata; questo è sempre possibile allargando opportunamente il disegno.

Soddisfare la **terza** affermazione è molto difficile in caso di grafi qualsiasi, e rappresenta in effetti il collo di bottiglia di questo metodo. Nel caso della butterfly comunque, questo può essere fatto.

Figura 8.13.



Concludendo, possiamo ottenere il layout della butterfly *n*-dimensionale come prodotto di due alberi binari completi di *n*+1 livelli, uno downward e l'altro upward come segue:

- nel disegno dell'albero binario, aggiungere i nodi fittizi;
- nel disegno dell'albero binario, assegnare ad ogni nodo una colonna per evitare collisioni di nodi;
- disegnare un albero sul piano *xy*, l'altro sul piano *yz*;
- (opzionale) costruire il LCP in 3D;
- proiettare il LCP sul piano *xz*.

Il layout così ottenuto, mostrato in Figura 8.14, ha le seguenti proprietà:

- è simmetrico;
- ha altezza H=2(*N*-1);
- ha ampiezza W=2(*N*-1);
- ha area $4N^2+o(N^2)$;
- i nodi di input e output non sono sui bordi; questa è una proprietà negativa nel caso in cui i nodi di input siano delle connessioni alle quali la rete va attaccata all'esterno, ma è irrilevante se i nodi sono dei processori;
- tutti gli archi di ciascun livello hanno la stessa lunghezza (per la proprietà di simmetria già enunciata).

Con questa tecnica possono essere disegnate parecchie topologie di interconnessione. Tra tutte, ricordiamo l'albero binario completo, che si può ottenere come LCP di due alberi "allungati" (vedi Figura 8.15); in tal caso il layout risultante è l'H-tree.

Figura 8.14.

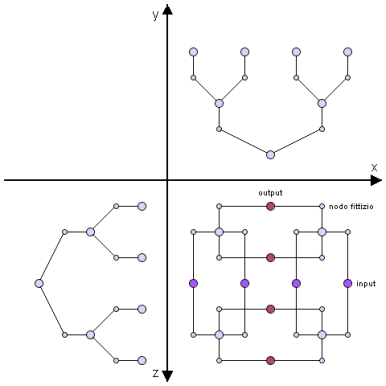
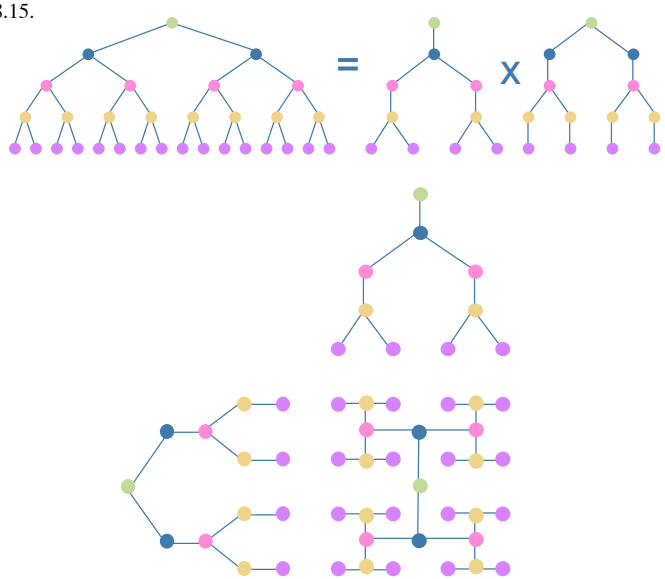


Figura 8.15.



8.2.3. Raffronto delle due rappresentazioni e cenni all'ottimizzazione dell'area

Abbiamo visto due differenti tecniche per costruire il layout di una tra le più note toplogie di interconnessione, la butterfly. Il primo metodo ha il vantaggio di fornire un disegno di area relativamente bassa e di "sembrare" una butterfly; di contro, ha il problema dei knok-knees e della griglia "slanted". L'altro metodo, invece, elimina questi due difetti, ma aumenta l'area e pone i nodi di input/output all'interno del disegno.

Nell'ottica di ottimizzare al meglio l'area, sono stati proposti altri latyouts della butterfly. In particolare, Dinitz [D98] dimostra che il layout di Even ed Even può essere migliorato, con degli aggiustamenti locali in modo da avere un'area pari a $11/6\ N^2+o(N^2)$. Successivamente, Avior et al. [Aal98] dimostrano che un layout con rettangolo circoscritto a lati paralleli agli assi cartesiani non può avere area minore di $N^2 + o(N^2)$ e forniscono un algoritmo che produce un layout di area ottima. Dinitz et al. [Dal99] dimostrano infine che, se si accetta che il layout possa avere rettangolo circoscritto con i lati non necessariamente paralleli agli assi (cioè 'slanted'), allora area $1/2N^2+o(N^2)$ è necessaria e sufficiente.

RIFERIMENTI BIBLIOGRAFICI

[Aal98] E. Avior, T. Calamoneri, S. Even, A. Rosenberg, "A tight layout of the butterfly network", *Th. of Computing Systems*, **31**, pp. 475-487, 1998.

[D98] Y. Dinitz, comunicazioni private, 1998.

[Dal99] Y. Dinitz, Shimon Even, Roni Kupershtok, Maria Zapolotsky: "Some Compact Layouts of the Butterfly", *Proc. Symp. on Parallel Algorithms and Architectures (SPAA '99)*, pp. 54-63, 1999.

[EE00] G. Even, S. Even: "Embedding Interconnection Networks in Grids via the Layered Cross Product", *Networks* **36**(2), pp. 91-95, 2000

[EL92] S.Even and A.Litman, "Layered Cross Product-A Technique to construct interconnection Networks", Networks 29, 1992.

[L92] F.T.Leighton, *"Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes."* Morgan Kaufmann ed. 1992.

[T??] Thompson ????

[W81] D.S. Wise: "Compact Layouts of Banyan/FFT Networks", *VLSI Systems and Computations*, pp.186-195, 1981

9. VISUALIZZAZIONE DI OGGETTI MOLTO GRANDI (INFINITI)

Riferimenti: [EF96] tutto escluso approccio gerarchico all'interno della sez. 3.1, [C04] pp 30-32
Per approfondire: [KW98] pp.193-227.

9.1. Tecnica della clusterizzazione

Molti grafi che scaturiscono da problemi reali hanno dimensioni tali da non poter in alcun modo essere rappresentati su un unico supporto. Nasce, dunque, la necessità di progettare nuove tecniche per poter avere una visione globale dell'intera struttura. Una di tali tecniche è la clusterizzazione, che consiste nel raggruppare oggetti che hanno tra loro una qualche affinità, tenendoli separati da tutti gli altri. Spesso, i grafi "reali" hanno associata una semantica che induce in qualche modo delle gerarchie nel grafo. Per esempio, si consideri un grafo che rappresenta il traffico di una rete telefonica, in cui i nodi rappresentano i chiamanti e gli archi le connessioni effettuate. Questo grafo ha associata una ovvia gerarchia indotta dai prefissi e dalla prima parte del numero.

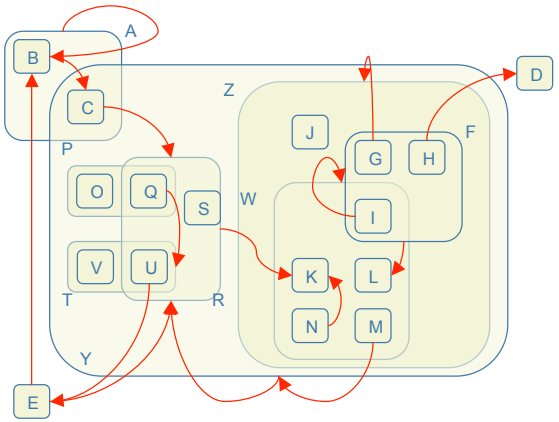
Anche i grafi che rappresentano il traffico nel WWW hanno una struttura simile, basata sui domini e sui sottodomini dell'indirizzo IP. La semantica definisce naturalmente il modo in cui le applicazioni si focalizzano interattivamente sulle aree di maggior interesse, mentre aggrega l'informazione nelle aree di minor interesse.

Altri esempi sono: la classificazione di documenti secondo l'argomento per una ricerca efficiente, la visualizzazione di strutture dati del software object-oriented e parallelo, il mantenimento della traccia della navigazione su Internet o attraverso le informazioni in un database.

In tutti questi casi, l'ammontare delle informazioni che si vogliono visualizzare diventano sempre più grandi e le relazioni tra esse sempre più complesse, quindi il classico modello di grafo tende ad essere inadeguato. In molte occasioni, infatti, è necessario visualizzare alcune informazioni strutturali sulle entità, ed è necessario un formalismo più complesso.

Partendo dal primo modello di grafo non classico, cioè l'*ipergrafo* [B73], Harel [H88] presenta un modello di grafo detto *higraph* (un esempio è in Fig. 9.1). Gli higraphs possono rappresentare relazioni complesse, usando dei raggruppamenti, detti *blobs*, multilivello, che possono includersi o intersecarsi arbitrariamente. Gli higraphs sono utili per una vasta gamma di applicazioni, tra cui la rappresentazione di databases, la rappresentazione della conoscenza e di diagrammi degli stati nei sistemi concorrenti. Tuttavia, per queste strutture è molto difficile sviluppare metodi di disegno automatico, ed il problema principale sembra provenire dalle complicazioni che sorgono quando i blobs e i sotto-blobs si intersecano arbitrariamente.

Figura 9.1.



Sugijama e Misue [SM91] presentano allora un nuovo modello, detto *compound graph*. Esso permette sia l'inclusione che l'adiacenza tra blobs, ma è meno generale del modello higraph.

Definizione. Un *compound graph (orientato)* è una tripla $D=(V, E, I)$ tale che $D_a=(V,E)$ è un grafo (orientato) (vedi Figura 9.2 parte destra) e $D_c=(V, I)$ è un grafo orientato (vedi Figura 9.2 parte sinistra). Gli elementi di E sono detti *archi di adiacenza*, gli elementi di I *archi di inclusione*. Quindi dire che un arco (v,w) appartiene ad I equivale a dire che v include w; questa interpretazione ha senso solo se il grafo diretto D_c è aciclico.

In Figura 9.3. sono riportate due diverse rappresentazioni di un compound graph (relativo ai grafi di adiacenza e di inclusione di Figura 9.2.).

Figura 9.2.

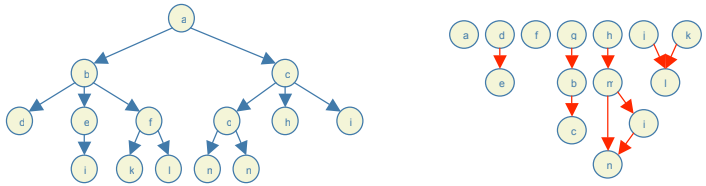
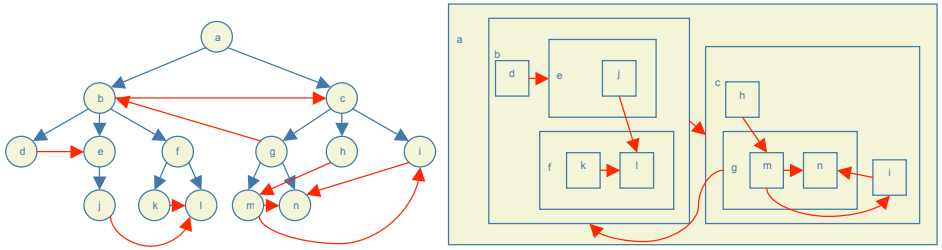


Figura 9.3.



Successivamente, Feng, Cohen e Eades [FCE95], formalizzano un modello introdotto per altri scopi in modo intuitivo, e lo introducono nell'area del disegno dei grafi: è il *clustered graph*. Esso è relativamente generale, poiché riesce ad essere utilizzabile per molte applicazioni; inoltre, sembra essere un modello ben rappresentabile con gli algoritmi di disegno di grafi.

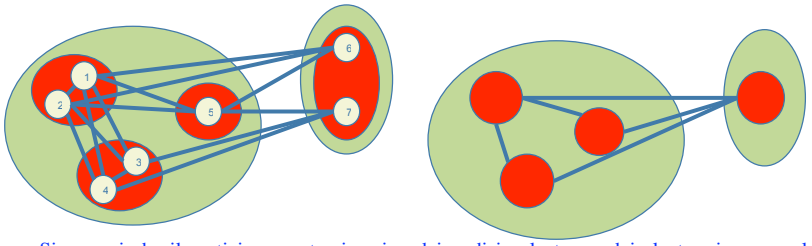
Un clustered graph consiste in un grafo G ed un partizionamento ricorsivo dei nodi di G; ogni parte è un *cluster*. Un esempio è mostrato in Figura 9.4. parte sinistra. Più formalmente:

Definizione. Una *partizione* di un insieme C è una famiglia di sottinsiemi $C=\{C_1, C_2, \dots C_k\}$ tale che l'unione di tutti i C_i sia pari a C e l'intersezione tra qualunque coppia di insiemi C_i e C_j sia pari all'insieme vuoto. Un *clustered graph* è un grafo su cui sia definita una partizione sull'insieme dei nodi; gli insiemi della partizione sono detti *clusters*. Sull'insieme dei clusters può, a sua volta, essere definita una partizione e così via ricorsivamente per un numero arbitrario, ma finito, di volte.

Per una fissata partizione $C_1, C_2, \dots C_k$ dell'insieme dei nodi di un grafo $G=(V,E)$, il *grafo quoziente* $\mathcal{G}=(\mathcal{V},\mathcal{E})=G/C$ è definito contraendo ogni insieme della partizione in un singolo nodo (vedi Figura 9.4. parte destra), cioè:

- $\mathcal{V}=\{C_1, C_2, \dots C_k\}$
- $(C_i, C_j) \in \mathcal{E} \Leftrightarrow i \neq j$ *ed* $\exists v \in C_i, w \in C_j$ *t.c.* $(v,w) \in E$.

Figura 9.4.

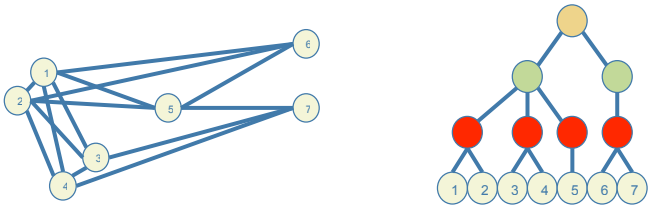


Si osservi che il partizionamento ricorsivo dei nodi in clusters e dei clusters in sovra-clusters suggerisce una struttura di inclusione ad albero. Pertanto, non di rado, si trova in letteratura la seguente definizione alternativa di clustered graph:

Definizione. Un *clustered graph* $C=(G,T)$ consiste di un grafo G ed un albero radicato T tale che le foglie di T sono esattamente i nodi di G; ogni nodo v di T rappresenta un cluster dei nodi di G che sono le foglie del sottoalbero radicato a v. T descrive una relazione di inclusione tra i clusters.

In Figura 9.5. è mostrato il clustered graph di Figura 9.4. parte sinsitra secondo questa nuova definizione.

Figura 9.5.



L'*altezza* di un cluster v, denotata con $h(v)$, è definita come l'altezza del sottoalbero di T radicato in v. Lo *span* di un arco (u,v) di T è $|h(u)-h(v)|$. Se lo span di un arco di T è maggiore di 1, diremo che quell'arco è *lungo*. Nel resto di questo capitolo assumiamo che ogni arco abbia span esattamente 1; questo può essere imposto considerando gli archi lunghi di T ed aggiungendo dei nodi fittizi in modo tale che tutti gli archi abbiano span 1.

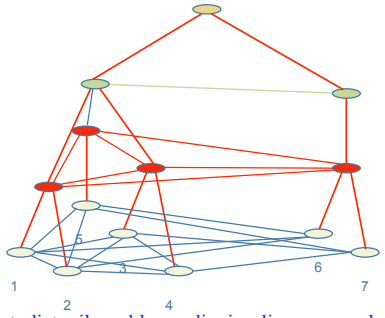
Un buon disegno di un clustered graph deve mostrare il disegno del grafo soggiacente come grafo classico, e rappresentare la struttura di clustering sovrastante. Quindi, successivamente alla formalizzazione di questo modello per sintetizzare le informazioni è nato un interesse dei ricercatori per tentare di generalizzare le tecniche di disegno di grafi a queste strutture più complesse. Infatti, non è sufficiente applicare banalmente algoritmi noti per il disegno di grafi perché, per ovvie ragioni di leggibilità, nodi che appartengono allo stesso cluster devono essere disegnati vicini, e questo non è sempre facile, soprattutto se si vogliono mantenere visibili alcune proprietà del grafo, come la planarità o le simmetrie. Lo scopo è quindi quello di preservare la struttura globale del

grafo sottostante, clusterizzando ricorsivamente sottografi sempre più grandi e disegnandoli come singoli nodi o regioni.

Nella classica visualizzazione bidimensionale, la struttura di clustering è rappresentata dall'inclusione di regioni del piano, cioè, un cluster è rappresentato da una regione che contiene il disegno dei nodi che appartengono al cluster (vedi fig. 9.4. parte sinistra). Più formalmente, un cluster v è disegnato come una regione chiusa $R(v)$ tale che: *i*. le regioni di tutti i sotto-clusters di v sono completamente contenuti dentro $R(v)$; *ii*. le regioni degli altri clusters sono completamente fuori da $R(v)$; *iii*. se esiste un arco tra due nodi interni a v, la sua rappresentazione è completamente contenuta in $R(v)$. Tuttavia, se la struttura clusterizzata cresce, la rappresentazione bidimensionale tende a divenire insufficiente. Infatti, oltre ai problemi dei tempi computazionali e della complessità intrinseca della struttura dati, c'è da sottolineare che i più sofisticati sistemi di visualizzazione si limitano, al più, ad alcune centinaia di nodi, mentre i grafi che provengono da applicazioni di visualizzazione dell'informazione contengono tipicamente migliaia o anche milioni di nodi. Una strategia comunemente utilizzata consiste nel visualizzare grandi grafi con una struttura di clustering tramite livelli di astrazione multipli. Un modo naturale per gestirli è quello di utilizzare la terza dimensione, in cui ogni livello di clustering utilizza un piano ad una diversa coordinata z, e la struttura di clustering è disegnata come un albero in tre dimensioni (v. fig. 9.6.).

Dato un clustered graph, sono stati sviluppati diversi algoritmi per visualizzarlo, sia in 2 che in 3 dimensioni. Particolarmente significativo è quello dovuto a Eades e Feng [EF96], che vedremo nel seguito, poiché in esso sono riassunte diverse classiche tecniche di disegno di grafi.

Figura 9.6.



Mentre è abbastanza studiato il problema di visualizzare un clustered graph dato in input, differente è il discorso per quello che riguarda il problema di creare dei clusters prima di rappresentare il grafo. In effetti, è molto probabile che da applicazioni reali scaturiscano grafi i cui clusters non sono noti a priori. In tal caso, a volte si possono applicare varie euristiche per trovare una clusterizzazione che usi proprietà specifiche, come la connettività, la dimensione prefissata dei clusters, la prossimità geometrica, e così via. Tuttavia, se non viene data una clusterizzazione, e non sono note proprietà particolari, si può fare ben poco, perché classicamente il problema di partizionare grafi secondo dei criteri che vanno ottimizzati è quasi sempre NP-arduo, e la maggior parte dei problemi di partizionamento sono ben noti come tipici problemi difficili. Per questo, sono molto pochi i lavori che vanno in questa direzione. In particolare, Duncan, Goodrich e Kobourov [DGK98] mostrano come creare una particolare decomposizione gerarchica che ammetta un disegno 3D; ma per grafi planari, la clusterizzazione proposta può portare ad incroci tra archi o ad attraversamenti di clusters da parte di archi, il che crea indubbi problemi alla leggibilità del disegno. Gli stessi autori propongono quindi una clusterizzazione che preserva la planarità e produce un albero di clusters di altezza logaritmica nella grandezza del grafo soggiacente. Questa proprietà è di particolare importanza per garantire l'efficienza della successiva operazione di disegno.

Si osservi che, mentre quando la clusterizzazione è già data i vari clusters possono essere dimensionati arbitrariamente – a seconda della semantica data alla clusterizzazione stessa – quando il grafo va clusterizzato in modo automatico, si cerca di rendere i clusters tutti approssimativamente della stessa dimensione e con il minor numero di archi possibile fuori dei clusters. Ciò avviene per andare incontro alle numerose applicazioni di queste problematiche, nel campo del VLSI, del calcolo parallelo e degli algoritmi *divide et impera*.

In proposito, ci sono tre categorie principali di metodologie per clusterizzare: *i*. graph-theoretical, *ii*. single-pass, *iii*. iterative. Fanno parte degli algoritmi graph-theoretical quelli che rappresentano la similitudine tra oggetti (ad es. documenti) - e quindi tra i nodi del grafo associato - tramite una matrice delle similitudini; i cluster vengono formati dagli oggetti più simili tra di loro, secondo una certa soglia di similitudine. Fanno parte degli algoritmi single-pass gli algoritmi così detti seed-oriented, in cui i cluster crescono a partire da un unico rappresentante, detto appunto *seme*; tipicamente gli oggetti vengono aggiunti ad un certo cluster se sono molto simili al seme che lo rappresenta; per questo tipo di algoritmi il numero richiesto di cluster deve essere noto a priori. Gli algoritmi iterativi sono volti a migliorare una clusterizzazione preesistente secondo qualche funzione euristica; di solito un algoritmo iterativo usa quindi la clusterizzazione prodotta da un altro algoritmo, ad esempio uno seed-oriented.

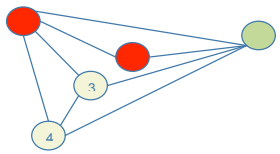
Un uso particolarmente indicato dei clustered graphs è quello della rappresentazione di grafi 'infiniti'. Infatti, quando i grafi diventano grandi, gli algoritmi classici per disegnare grafi si comportano male, quando non falliscono, e questo è ovvio perché la complessità computazionale degli algoritmi di disegno di grafi è direttamente proporzionale alla dimensione del grafo. Il problema viene peggiorato dalla complessità intrinseca di gestire grandi grafi. In questo senso, il graph clustering ci porta un passo avanti, poiché gli algoritmi per disegnare grafi possono essere applicati a piccole porzioni, così che il tempo complessivo rimanga ancora soddisfacente: il vantaggio di raggruppare i nodi in clusters è che il grafo si può ricorsivamente suddividere in livelli di dettaglio decrescente che possono essere visitati top-down, e tale vantaggio è tanto maggiore quanto più chiara è la visualizzazione che se ne riesce a dare.

Un altro vantaggio della clusterizzazione sta nel fatto che nodi correlati possono essere racchiusi in clusters rappresentati come super-nodi. Quindi, l'utente vede, almeno in prima istanza, un 'riassunto' del grafo, costituito da super-nodi e da super-archi tra super-nodi. Solo a richiesta, alcuni clusters potranno essere mostrati in maggior dettaglio rispetto ad altri. Questo tipo di rappresentazione non solo facilita la visualizzazione a differenti livelli di dettaglio, ma mantiene anche la mappa mentale dell'utente. Più precisamente:

Definizione. Sia dato un grafo G ed un albero radicato e arbitrario T, tale che le foglie di T corrispondano ai nodi di G. Una *vista* del grafo G è un sottinsieme U dei nodi di T, tale che ogni foglia di T (e quindi nodo di G) abbia un unico antenato in U. La vista induce un grafo quoziente G/U, formato – come già detto - contraendo ogni nodo di G nel suo nodo rappresentante in U e cancellando archi doppi e cappi.

In Figura 9.6. è raffigurata una vista del clustered graph di Figura 9.4. parte sinistra.

Figura 9.6.



Un problema che sorge da questo modo di vedere le cose è il seguente: un super-nodo è collegato ad altri super-nodi allo stesso livello gerarchico; nel momento in cui l'utente espande un super-nodo

nella sua rappresentazione più dettagliata, come si fa a determinare come connettere i nodi appena espansi al resto del grafo senza esaminare l'intero grafo sottostante? Infatti, non è chiaro, tramite la semplice struttura di vista, come si possa risalire efficientemente agli archi che collegano diversi livelli della struttura.

9.2. Un algoritmo per la rappresentazione di grafi clusterizzati

Gli autori rappresentano il grafo soggiacente tramite una rappresentazione piana, rettilinea o ortogonale su griglia, mentre utilizzano la terza dimensione per evidenziare la struttura di clustering, che si presenta come un albero, per rappresentare il quale utilizzano una classica tecnica di disegno rettilineo 3D. L'algoritmo è organizzato in tre passi:

1. eliminazione degli archi lunghi
2. costruzione del disegno della vista corrispondente ad ogni livello (bottom-up)
3. costruzione dell'albero di inclusione (bottom-up).

Questa parte è scritta in modo molto conciso. Per i dettagli, vedi [EF96]

Il primo passo dell'algoritmo consiste nell'eliminare gli archi lunghi come abbiamo già evidenziato precedentemente, cioè aggiungendo dei vertici fittizi, in modo che gli archi della struttura gerarchica colleghino sempre nodi di livelli adiacenti.

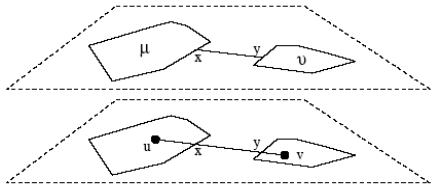
Durante il secondo passo, per rappresentare il grafo in due dimensioni, viene utilizzato l'algoritmo di Tutte per ottenere un disegno rettilineo, o un algoritmo di disegno ortogonale. Per fare ciò, si considera la struttura di clusters, si prende ciascun cluster e si disegna il grafo che ne è all'interno, posizionando ciascun cluster su un diverso poligono (un rettangolo nel caso ortogonale). L'algoritmo viene applicato ad ogni livello della clusterizzazione, tenendo conto che ora i nodi non sono più puntiformi ma dei poligoni.

Una volta disegnato i diversi livelli, vediamo come disegnare gli archi che collegano clusters allo stesso livello.

Dati due clusters *c* e *d* sullo stesso livello, sappiamo di doverli congiungere con un arco se esiste almeno un nodo in *c* connesso ad almeno un nodo in *d*. Se queste connessioni sono più di una, possiamo arbitrariamente sceglierne una (ad esempio tra u e v) e procedere al modo seguente:

- considera l'arco *(u,v)* disegnato sul livello sottostante,
- considera le sue intersezioni *x* e *y* con i poligoni rappresentanti *c* e *d*,
- disegna l'arco *(c,d)* come la parte di arco *(u,v)* compreso tra i punti *x* e *y*. (vedi Figura 9.7).

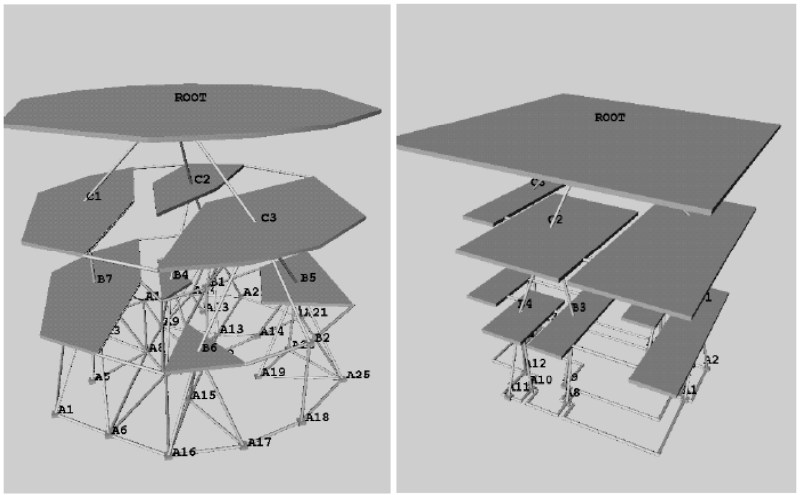
Figura 9.7.



Non rimane ora che disegnare gli archi tra livelli (terzo passo), come segue:
FOR i=1 TO *h* (dove *h* è l'altezza dell'albero delle inclusioni) DO
IF i=1 per ogni nodo (foglia), posizionalo dove è stato messo dal disegno 2Dim. del grafo
ELSE per ogni nodo-poligono *c* calcola la media delle coordinate dei suoi figli
(a livello *i-1*) ed usa questa media come punto di riferimento per *c*.

Con questo metodo ogni nodo-poligono viene posizionato sopra la sua rappresentazione più dettagliata (vedi Figura 9.8.); gli archi dell'albero possono essere rappresentati come segmenti rettilinei.

Figura 9.8.



Poiché, durante il primo passo, abbiamo rimpiazzato gli archi lunghi con dei segmenti tra livelli adiacenti, non si possono verificare incroci. Inoltre, quando andiamo ad eliminare i nodi fittizi per rendere nuovamente lunghi gli archi che lo necessitano, l'aver usato la media delle coordinate garantisce che questi archi saranno rettilinei, salvo al più una svolta, in corrispondenza del livello successivo a quello del nodo padre.

La rappresentazione di clustered graphs sembra a tutt'oggi un metodo promettente per visualizzare grandi grafi, sebbene ci siano ancora numerosi problemi da risolvere (sia per quello che riguarda la creazione dei clusters che nel campo delle strutture dati). Un problema particolarmente rilevante è che ci sono pochissimi algoritmi incrementali in letteratura: è tipico che l'insieme degli oggetti costituenti i nodi sia spesso soggetto a modifiche o ad incrementi (si pensi ad una collezione di documenti), ma la strategia comune – sebbene altamente inefficiente – consiste nel ripetere interamente la procedura di clusterizzazione di tanto in tanto; tra un aggiornamento e l'altro vengono fatti solo piccoli aggiustamenti.

9.3. Tecnica della navigazione

Un'alternativa alla clusterizzazione per visualizzare grandi grafi consiste nella *navigazione*, in cui l'utente vede solo un piccolo sottinsieme di nodi e di archi alla volta, avendo a disposizione delle facilities per navigare attraverso il grafo. Più in dettaglio, si suppone di avere il disegno del grafo sua una pagina virtuale sufficientemente grande, e si fornisce l'utente di una finestra dotata di scroll bars per permettergli di vedere la parte di interesse e di navigare attraverso il grafo (vedi Figura 9.9).

Figura 9.9.

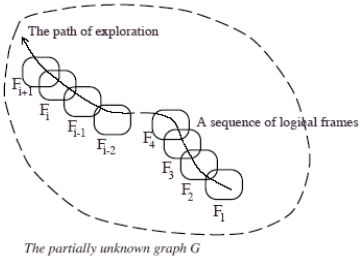


Fig 7. Esplorazione di un grafo.

Questo approccio nasconde diversi problemi:

- si basa sull'assunzione, spesso poco sensata, che il grafo da visualizzare sia completamente noto e, quindi, memorizzato: in molte applicazioni non è così, ed il grafo potrebbe non essere noto per intero;
- l'utente può spostarsi attraverso il grafo solo linearmente, e non seguendo un suo percorso logico, cioè non può saltare ad una parte completamente diversa del grafo, ma deve arrivarci tramite una sequenza di movimenti successivi;
- gli archi lunghi, che non entrano interamente nella finestra, sono difficili da seguire;
- si perde facilmente la mappa mentale, perché non viene mantenuta alcuna mappa della navigazione dell'utente;
- è molto dispendioso (laddove non impossibile) dal punto di vista della memoria memorizzare, oltre che la sua struttura astratta, anche il disegno dell'intero grafo sulla pagina virtuale.

Relativamente a quest'ultimo problema, volendo pensare di avere a disposizione l'intero grafo ma non il suo disegno, assumendo di visualizzare una finestra definita ad esempio in modo da mostrare il sottografo indotto dai nodi che hanno una certa distanza topologica da un dato nodo (considerato come il centro della finestra), non appare facile costruire i disegni che seguono la navigazione dell'utente in una qualche direzione in modo da mantenere la sua mappa mentale. In altre parole, disegni consecutivi hanno, tipicamente, grandi intersezioni, e queste porzioni di grafo dovrebbero essere rappresentate nello stesso modo, affinché l'utente si orienti nella navigazione.

Un approccio che potrebbe essere seguito è quello di aggiungere la parte del disegno verso cui l'utente si sposta alla porzione già costruita ed ancora visibile. Tuttavia, questa tecnica dà risultati sperimentali pessimi, poiché – man mano che ci si allontana dalla porzione iniziale di disegno – la visualizzazione peggiora progressivamente, in termini di numero di svolte, di lunghezza degli archi, di numero di incroci, ecc.

9.4. Discussione delle problematiche collegate a queste due tecniche e loro raffronto

Probabilmente, l'uso integrato delle due tecniche, di clustering e di navigazione, può garantire una soluzione a molti problemi che nascono tipicamente dalla visualizzazione di grosse quantità di dati: la navigazione dell'utente attraverso il grafo, passando da un cluster all'altro utilizzando la struttura sovrastante di clustering, potrebbe essere un possibile compromesso per evitare di mantenere in memoria l'intero grafo e, allo stesso tempo, per non alterare eccessivamente la mappa mentale dell'utente. Tuttavia, non sono ancora stati progettati algoritmi che funzionino efficientemente sfruttando le due tecniche in modo integrato, sebbene molti tentativi siano stati fatti.

9.5. Occhio di Pesce

Per ovviare agli inconvenienti presenti nelle due tecniche ora descritte sono state proposte diverse soluzioni. Ad esempio, se il grafo è di dimensioni moderatamente grandi, una tecnica particolarmente indicata, ideata da Sarkar e Brown [SB94] è quella detta "dell'occhio di pesce", in cui il grafo è disegnato per intero ma molto piccolo sullo schermo, e l'utente può vedere in dettaglio un piccolo sottografo avendo a disposizione anche il "contesto" di quel sottografo: l'area di interesse viene evidenziata, mostrando le altre porzioni dell'immagine con dettaglio sempre minore man mano che ci allontaniamo dal centro (vedi Figura 9.10).

Questo metodo risolve anche i problemi creati dalla sola operazione di *zoom* che, focalizzandosi su un punto, farebbe perdere il contesto.

Vediamo, in modo semplificato, come sia possibile implementare questa tecnica:

- il *punto del fuoco* è definito di solito dall'utente;
- la *posizione di un nodo* generico nella vista ad occhio di pesce dipende dalla sua posizione nella vista normale e dalla sua distanza dal fuoco;
- la *dimensione di un nodo* generico nella vista ad occhio di pesce dipende dalla distanza dal fuoco e dalla sua dimensione nella vista normale;
- la *distanza dal fuoco* di ogni nodo del grafo è distorta da una funzione $h(x)$ il cui dominio e codominio sono l'intervallo unitario;
- la *distorsione* creata dall'occhio di pesce è la conseguenza della forma della funzione che ha un incremento più veloce intorno a 0 (intorno al fuoco), e un incremento tanto più lento quanto più ci avviciniamo ad 1 (lontano dal fuoco). La definizione esatta della funzione può produrre un minore o maggiore effetto di distorsione.

Figura 9.10.

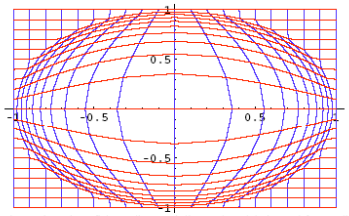


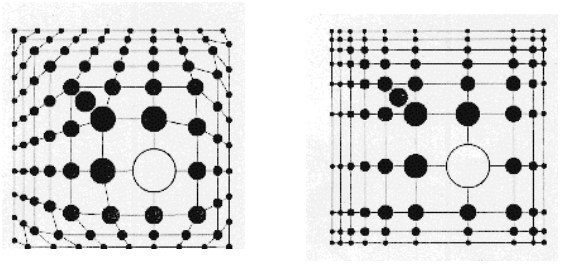
Fig 2. Distorsione fisheye di una griglia regolare del piano. Il fattore di distorsione è 4.

Ci sono alcune variazioni a questo schema di base. La tecnica appena descritta è di solito nota come *distorsione polare* (vedi Figura 9.11. parte sinistra), in quanto si applica radialmente ai nodi in tutte le direzioni che partono dal punto del fuoco. Un'alternativa è usare una *distorsione cartesiana*: la distorsione della distanza è applicata indipendentemente su x ed y prima di stabilire la posizione finale del nodo (vedi Figura 9.11. parte destra).

L'operazione fondamentale della tecnica ad occhio di pesce è quella di distorcere la posizione di ogni punto. Se la distorsione è applicata fedelmente, anche gli archi che connettono i nodi saranno distorti. Matematicamente, il risultato di questa distorsione è una curva generica. Sistemi grafici standard (per es. X11, Java2D OpenGL) non offrono i necessari strumenti per trasformare le linee in queste curve facilmente (gli strumenti possono essere piuttosto complessi). La scelta dei progettisti è quindi, di solito, quella di approssimare gli intervalli di linea originali con un numero

alto di punti, trasformare questi punti, e mostrare una spezzata che approssimi l'ideale curva trasformata.

Figura 9.11.



Il problema è che il numero di punti dell'approssimazione deve essere relativamente alto per ottenere un buona qualità dell'immagine, il che porta ad un proibitivo numero di calcoli e riduce nettamente le prestazioni del sistema. L'unica soluzione possibile consiste allora nell'applicare la distorsione dell'occhio di pesce sul nodo singolo, e connettere poi i nodi trasformati da un arco diretto. La conseguenza di questa soluzione inesatta è l'introduzione di incroci (si veda, per esempio, il quadrante di Figura 9.12b in alto a sinistra).

Come già evidenziato, questa tecnica ha senso solo per grafi relativamente piccoli, il cui disegno completo, anche se rimpicciolito, riesca ad entrare tutto nello schermo. Questo perché il maggior lato positivo della tecnica ad occhio di pesce è quella di contestualizzare la vista dei dettagli, e se l'ipotesi di poter vedere tutto il grafo cade, allora perde significato anche l'utilizzo della tecnica stessa.

Figura 9.12. ->

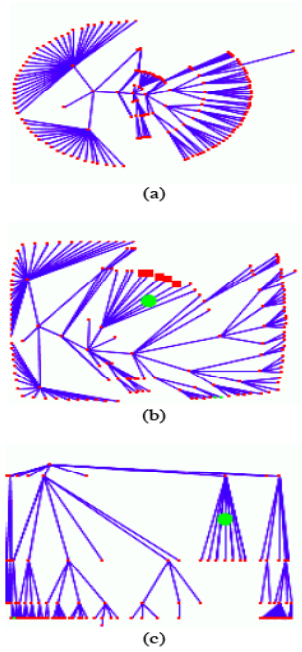


Fig 4. Distorsione Fisheye. Figura (a) rappresenta il grafo senza il fisheye. Figura (b) usa fisheye polare, mentre Figura (c) usa un fisheye cartesiano con un layout differente dello stesso grafo. Il punto verde in figura (b) e (c) denotano i punti focali della distorsione fisheye. Notare l'incrocio extra in figura (b) causato dalla distorsione.

9.6. Altre tecniche

Tecniche alternative possono essere utilizzate nel caso in cui il grafo non abbia un numero eccessivo di nodi, ma sia troppo denso, cioè contenga troppi archi. E' facile immaginare i problemi di visualizzazione che scaturiscono da un grafo molto denso, per cui è strategia comune ridurre il numero di archi da visualizzare. Ci sono molti modi per ottenere questo obiettivo; uno è quello di assegnare dei pesi agli archi secondo certi criteri, e poi imporre una soglia al peso, così che solo gli archi che superino la soglia verranno inclusi nella visualizzazione [ZB94]. Questo metodo è banale da implementare, ma non tiene conto della struttura intrinseca del grafo, così che il grafo visualizzato potrebbe non rispecchiare l'essenza del grafo originale.

Un altro metodo è quello di estrarre un albero ricoprente da un grafo, riducendo il numero di archi da $O(n^2)$ ad $n-1$. Questo approccio garantisce che il numero di archi rappresentati sia sempre basso.

Tali tecniche, tuttavia, non possono essere utilizzate indiscriminatamente, poiché nella maggior parte delle applicazioni non è possibile perdere una porzione di informazioni.

RIFERIMENTI BIBLIOGRAFICI

[B73] Berge???

[C04] C. Chen: *Information Visualization beyond the horizon* – 2nd Edition, Springer 2004.

[EF96] P. Eades, Q.-W. Feng: Multilevel Visualization of Clustered Graphs. *Proc. Graph Drawing '96*, LNCS 1190, pp. 101-112, 1996.

[FCE95] Q.-W. Feng, R.F. Cohen, P. Eades: Planarity for clustered graphs. *Proc. ESA '95,* LNCS 979, pp. 213-226, 1995.

[H88] Harel???

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

[DGK99] C.A. Duncan, M.T. Goodrich, S.G. Kobourov. Balanced aspect ratio trees and theri use for drawing very large graphs. *Proc. GD'98,* LNCS 1547, pp.111-124, 1999.

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

[SB94] M. Sarkar, M.H. Brown. Graphical Fisheye Views. *Communications of the ACM*, 37(12), 73-84, 1994.

[SM91] K. Sugiyama, K. Misue. Visualization of structural information: Automatic drawing of compound graphs. *IEEE Trans. on Systems, Man and Cybernetics,* 21(4), pp. 876-892, 1991.

[ZB94] M. Zizi, M. Beaudouin-Lafon: Accessing hyperdocuments through interactive dynamic maps. Proc. *ECT '94*, 126-135, 1994.