3. VISUALIZZAZIONE DI OGGETTI CON STRUTTURA AD ALBERO

Riferimenti: [KW98] pp. 46-52, [Dal99] pp.41-60

3.1. Esempio di oggetti con struttura ad albero: mappa di un file system, indice di una base di dati, alberi genealogici, …

L'importanza dell'albero radicato come struttura dati astratta e il suo utilizzo per rappresentare delle semplici gerarchie hanno reso disponibile una grande varietà di algoritmi per visualizzare alberi. In effetti, sono molti gli esempi di strutture ad albero che può aver senso rappresentare: alcune applicazioni che richiedono una opportuna visualizzazione sono organigrammi di aziende, alberi di ricerca, parse trees di programmi, alberi genealogici. Un'applicazione particolarmente significativa è l'organizzazione dei dati nelle basi di dati: un passo fondamentale nella creazione di una base di dati è la fase di disegno. L'informazione da memorizzare deve essere suddivisa in tipi e categorie, si devono determinare le relazioni tra i tipi ed evitare conflitti. Il principale *modello* di base di dati usato è quello *relazionale*:

- gli *elementi* sono linee di tabelle (*tuple* in relazioni)
- ogni *tabella* rappresenta un insieme di oggetti
- gli *attributi* sono componenti delle tuple
- le *relazioni* e le *dipendenze* sono organizzate anch'esse in tabelle.

I *diagrammi entità-relazione* (ER) sono il modo più diffuso per rappresentare queste informazioni:

- gli oggetti (entità) e i loro attributi formano i nodi;
- gli archi legano un attributo alla sua entità;
- gli archi possono essere etichettati quando ci sono dei vincoli di cardinalità.

I diagrammi entità-relazioni sono di solito ipergrafi, perché gli archi possono essere incidenti a più di due nodi. Un algoritmo apposito può gestire questa eventualità o inserire dei nodi fittizi per trasformare il diagramma in grafo; non è dunque restrittivo supporre che i diagrammi ER siano grafi, spesso piuttosto complessi.

Il *modello gerarchico*, in contrasto con quello relazionale, propone un'organizzazione più piatta dei dati e molte basi di dati orientate agli oggetti usano questo modello. L'idea è simile alla precedente, ma ogni entità è chiamata *oggetto* e appartiene ad una certa tabella o tipo, detta *classe*, che condivide lo stesso insieme di *attributi*. Le classi possono ereditare insiemi di attributi da altre classi. Questo modello si rappresenta come un grafo diretto aciclico (albero) in cui sono rappresentate le ereditarietà tra tutte le classi.

Visualizzare questo albero è importante per non introdurre cicli in fase di progettazione.

3.2. Terminologia per gli alberi

Riferimenti: [Dal99] pp 42-43, un qualunque libro di teoria dei grafi o di algoritmi su grafi di base.

Un *albero T* è un grafo semplice connesso aciclico. Un *albero radicato* consiste di un albero *T* ed un nodo di *T* distinto dagli altri, detto *radice*. Un albero radicato induce naturalmente un orientamento sugli archi, quindi può essere considerato come un particolare grafo semplice orientato aciclico (tutti gli alberi radicati sono grafi orientati aciclici, ma non è vero il viceversa). Più avanti osserveremo come, nonostante siano molti gli algoritmi per visualizzare grafi orientati aciclici, e ciascuno di essi potrebbe essere applicato ad un albero radicato, tuttavia, gli alberi sono strutture più semplici, e ciò spinge a considerare approcci diversi volti a meglio ottimizzare i criteri estetici.

Se *(u,v)* è un arco di un albero radicato *T*, *u* è detto *padre* di *v*, e *v* è detto *figlio* di *u*. Una *foglia* è un nodo senza figli.

Un *albero ordinato* è un albero radicato con un ordinamento dei figli di ciascun nodo *v*. Un *albero binario* è un albero radicato in cui ogni vertice ha al più due figli. In molte applicazioni gli alberi binari sono ordinati, e quindi – laddove non meglio specificato – si assumerà ordinato ogni albero binario. In tal caso, i figli di ciascun nodo vengono definti *figlio sinistro* e *figlio destro*; se un nodo ha un solo figlio, deve essere definito se esso sia sinistro o destro.

Dato un vertice *v* di un albero radicato *T*, il *sottoalbero radicato* in *v* è il sottografo indotto da tutti i cammini (orientati) che partono da *v*, e *v* è detto *radice* del sottoalbero. Se *T* è binario e *v* ha due figli, il sottoalbero radicato al figlio sinistro di *v* è detto *sottolabero sinistro* di *v*, e il sottolabero radicato al figlio destro di *v* è detto *sottolabero destro* di *v*.

La *profondità* di un vertice *v* di un albero *T* è il numero di archi del cammino tra la radice e *v* (quindi la radice ha altezza 0 ed i suoi figli altezza 1). L'*altezza* di *T* è il massimo delle profondità dei vertici di *T*. Dato un albero binario *T* con *n* vertici di altezza *h*, vale la seguente relazione: $n \leq 2^{h+1}-1$. Un albero binario si dice *completo* se, per esso, nella relazione precedente vale il segno di uguaglianza.

3.3. Visualizzazione ortogonale su griglia rettilinea in area ottima (H-tree)

Il risultato presentato nel seguito è stato pubblicato, indipendentemente, da Leiserson [L80] e da Valiant [V81]. Essi dimostrano che è possibile costruire una rappresentazione rettilinea su griglia ortogonale piana di un albero binario completo con *n* nodi in area $O(n)$. Si osservi che questo risultato è ottimo, poiché la rappresentazione su griglia di ogni grafo con *n* nodi non può occupare area inferiore all'ordine di *n*, pertanto $\Omega(n)$ è una limitazione inferiore per l'area di ciascun disegno.

Tuttavia, Brent e Kung [BK80] dimostrano che, se le foglie sono vincolate a giacere sul contorno del disegno, allora è necessaria almeno $\Omega(n \log n)$ area.

Segue un algoritmo ricorsivo per la visualizzazione di alberi binari completi. E' ovvio che, per ogni albero binario non completo, ne esiste uno più grande completo che lo contenga, e quindi l'algoritmo riesce comunque a visualizzare ogni tipo di albero binario radicato; tuttavia l'area risulta ottima solo se l'albero considerato è completo.

Sia T_h un albero binario completo di altezza *h*. L'albero T_h si ottiene connettendo le radici di due copie di alberi T_{h-1} ad un nuovo nodo, che rappresenta la radice di T_h. L'algoritmo sfrutta questa definizione ricorsiva come segue:

- *passo base*: l'albero T_0, costituito da un solo nodo, viene rappresentato in modo ovvio;
- *passo induttivo*: l'albero T_h viene rappresentato sfruttando il disegno dei due alberi T_{h-1} che lo costituiscono, già costruito per ipotesi induttiva, e connettendo le loro radici ad un nuovo nodo. Si distinguono due casi, a seconda che l'altezza *h* di T_h sia pari o dispari.
  Se *h* è dispari, i disegni dei due T_{h-1} si connettono orizzontalmente alla nuova radice, mentre se *h* è pari, la connessione è effettuata verticalmente. La Figura 3.1 mostra la rappresentazione di T_h, data la rappresentazione di T_{h-1}.

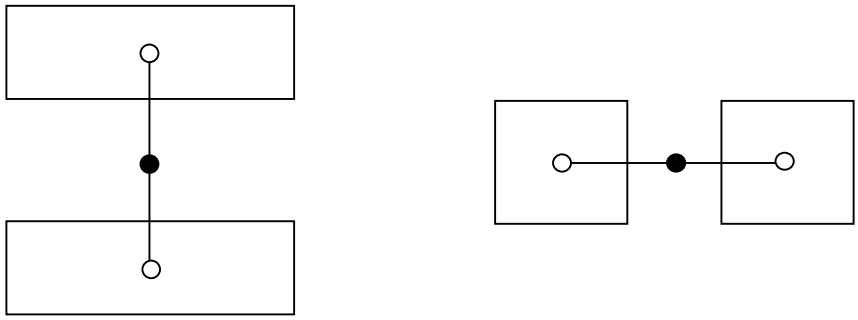
Il metodo ora descritto produce un disegno detto *H-tree* a causa della regola di costruzione, la cui forma ricorda una H.

Teorema. *L'area occupata da un H-tree con n nodi è 2(n+1)+o(n).*

Dimostrazione.Siano l_h e w_h l'altezza e la larghezza del più piccolo rettangolo circoscritto ad un H-tree di altezza *h*. Gli H-trees di altezza 0 e 1 hanno area nulla; gli H-trees di altezza 2 e 3 hanno

$l_2=2, w_2=2$, $l_3=2$ e $w_3=6$, e rappresentano il caso base della prossima formula induttiva. Partendo dalla costruzione dell'H-tree di altezza h si deduce che:

Figura 3.1.



h pari *h* dispari

se h è dispari: $l_h = l_{h-1}$, $w_h = 2\,w_{h-1} + 2$

se h è pari $l_h = 2\,l_{h-1} + 2$, $w_h = w_{h-1}$

Si osservi che, nel caso pari, le due dimensioni sono uguali, e pertanto è sufficiente risolvere solo una delle due equazioni, ad esempio quella per w_h. Per semplificare la comprensione, verrà sottolineato ad ogni passo se il valore corrente di h sia pari o dispari:

$$w_{h(pari)} = w_{h-1(dispari)} = 2w_{h-2(pari)} + 2 = 2w_{h-3(dispari)} + 2 =$$
$$= 2^2 w_{h-4(pari)} + 2^2 + 2 = 2^2 w_{h-5(dispari)} + 2^2 + 2 =$$
$$= 2^3 w_{h-6(pari)} + 2^3 + 2^2 + 2 = \quad \text{generalizzando:}$$
$$= \ldots = 2^k w_{h-2k(pari)} + \Sigma_{i=1..k}\, 2^i = \quad \text{raggiungendo il caso base } h-2k=2\text{:}$$
$$= 2^{(h-2)/2} w_2 + \Sigma_{i=1..(h-2)/2}\, 2^i = \quad \text{ricordando che } \Sigma_{i=1..r}\, 2^i = 2^{r+1}-2\text{:}$$
$$= 2\cdot 2^{(h-2)/2} + 2^{(h-2)/2+1} - 2 = 2\cdot 2^{(h-2)/2+1} - 2 = \quad \text{ricordando che } n=2^{h+1}-1\text{:}$$
$$= (2(n+1))^{1/2} - 2.$$

Ne segue che, se h è pari, l'area occupata dall'H-tree è $2(n+1)+o(n)$.

Nel caso di h dispari, osservando che $l_{h(disdpari)} = l_{h-1(pari)}$ e che $w_{h(dispari)} = w_{h+1(pari)}$, è facile dedurre che $l_h = (n+1)^{1/2} - 2$ e $w_h = 2(n+1)^{1/2} - 2$, e quindi l'area è nuovamente $2(n+1)+o(n)$. **CVD**

3.4. Algoritmi di disegno upward planare basati sul livellamento dell'albero, con nodi a coordinate intere

Riferimenti: [Dal99] pp 43-51.

E' evidente che il metodo di visualizzazione di alberi binari completi come H-trees ha come pregio quello di determinare un disegno di area ottima perché lineare nel numero di nodi; tuttavia, l'H-tree non evidenzia la struttura gerarchica dell'albero; in altre parole, non si riesce ad individuare immediatamente la radice, ne' ad evincere l'organizzazione per livelli dell'albero rappresentato, ne' – infine – a capire quali e quante siano le foglie. Inoltre, se l'albero non è completo, non dà risultati soddisfacenti.

E' allora necessario aggiungere la convenzione di disegno downward a quella di disegno piano per evidenziare il livellamento dell'albero indotto dalla distanza dalla radice.

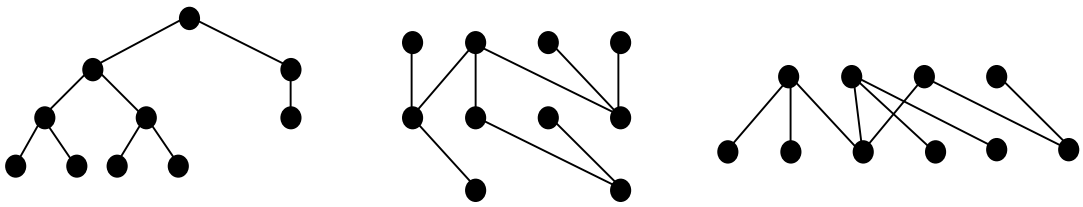
Un *grafo livellato* (*layered graph*) è un grafo non orientato $G = (V,E)$ tale che:

- $V = V_1 \cup V_2 \cup ... \cup V_k$, $V_i \cap V_j = \varnothing$ per ogni $i \neq j$ e
- per ogni arco $e = \{u,v\} \in E$ esiste un i tale che $u \in V_i$ e $v \in V_{i+1}$, $1 \leq i < k$.

Più informalmente, un grafo livellato è un grafo per il quale sia possibile assegnare ad ogni nodo v un livello in modo tale che v sia adiacente solo a nodi del livello precedente o successivo. Ovviamente non tutti i grafi sono livellati. Gli alberi sono grafi livellati ed un livellamento ammissibile è indotto dalle distanze dalla radice: tutti i nodi a distanza k dalla radice appartengono al livello k. Questa informazione si ottiene eseguendo una visita in ampiezza.

La vastità e l'importanza della classe di grafi livellati ha portato alla progettazione di molti algoritmi efficienti per il disegno di tali grafi in cui la proprietà del livellamento venga ben evidenziata. E' ragionevole pensare che un albero potrebbe essere ben visualizzato tramite uno di questi algoritmi. Tuttavia, un algoritmo pensato per grafi livellati non si adatta, in generale, agli alberi, non solo perché è pensato per grafi molto più generali, ma soprattutto perché non garantisce di rispettare la struttura dell'albero.

Figura 3.2.



In Figura 3.2 è mostrato un albero (a sinistra) ed una sua possibile visualizzazione come grafo livellato (al centro) e come grafo bipartito (a destra), visto che un criterio estetico potrebbe richiedere di minimizzare il numero dei livelli utilizzati. Diverso è il discorso se si fissano *a priori* i livelli dell'albero come quelli indotti dalla distanza dei nodi dalla radice, come nella definizione seguente.

Definizione: Si fissi un riferimento di assi cartesiani XY in cui l'origine sia in alto a sinistra. Un *disegno livellato* è un disegno di un grafo livellato, tale che ogni nodo v sul livello i abbia $y(v)=-i$, dove $y(v)$ denota l'ordinata del nodo v nel sistema di assi cartesiani XY. Essendo le ordinate fissate, un qualsiasi algoritmo che produca un disegno livellato deve restituire per ogni nodo soltanto l'ascissa.

Mentre è ovvio che un disegno livellato di un albero sia downward, per la planarità il discorso non è così immediato. Si tratta di assicurare che, se due vertici v' e v'' sono sullo stesso livello i nel disegno, allora il loro ordine relativo da sinistra a destra è lo stesso dei loro padri u' ed u'' nel livello $i-1$. Usando le coordinate cartesiane, se $x(u')<x(u'')$ allora si deve imporre $x(v')<x(v'')$. Ciò è automaticamente assicurato se l'albero è ordinato.

Un'altra richiesta ragionevole da fare è che ogni nodo abbia ascissa compresa tra le ascisse dei suoi figli, e meglio ancora sarebbe se l'ascissa del padre si trovasse nel punto medio dell'intervallo delle ascisse dei suoi due figli. Un esempio in cui questa richiesta è particolarmente significativa è l'albero binario di ricerca, in cui l'ordine dei nodi lungo l'asse X è semanticamente importante.

Un semplice algoritmo consiste nell'assegnare ad ogni nodo di un albero binario radicato e ordinato T un'ascissa pari al suo numero di visita *in order* dell'albero T. Formalmente, se un nodo v dell'albero T è l'i-esimo nella visita *in order* allora $x(v)=i$.

Teorema. *L'area del disegno prodotto dall'algoritmo precedente è* $O(n^2)$.

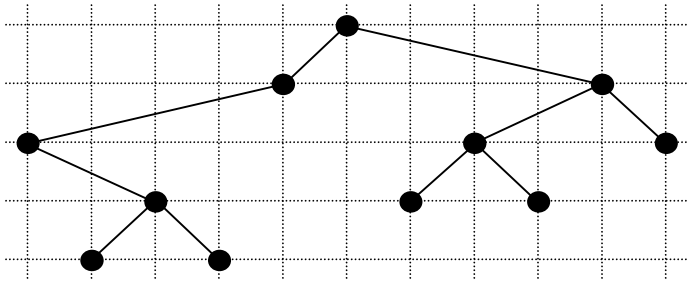
Dimostrazione. Le ascisse degli n nodi sono intere e tutte distinte. Segue che l'ampiezza della rappresentazione è esattamente n. Il numero delle ordinate è limitato dall'altezza dell'albero T, che nel caso peggiore è n. **CVD**

Il semplice algoritmo appena esposto non garantisce disegni particolarmente gradevoli, come evidenziato in Figura 3.3; in partcolare, l'output ha due difetti:

- il disegno in genere è molto più largo del necessario, e quindi l'area è troppo grande;
- l'ascissa di un nodo non è necessariamente centrata rispetto all'ascissa dei suoi figli.

L'algoritmo di Reingold e Tilford [RT81] si prefigge di migliorare l'algoritmo precedente. Esso si basa sul paradigma del divide et impera. Intuitivamente il *passo impera* si basa sull'idea che ogni sottoalbero sia disegnato su un foglio di carta separato; si spostano quindi i due fogli di carta l'uno verso l'altro fino a che i livelli siano allineati e la distanza ad ogni livello sia almeno 2.

Figura 3.3.



L'algoritmo può essere così schematizzato:

Input: un albero binario radicato ordinato T;
Output: un disegno livellato di T;
Passo base:
Se T è costituito da un singolo vertice, disegnalo banalmente;
Passo divide:
ricorsivamente applica l'algoritmo sul sottoalbero sinistro e destro del nodo corrente r;
Passo impera:
se r ha due figli:

- muovi i disegni dei suoi sottoalberi sinistro e destro in maniera che la loro distanza ad ogni livello sia ≥ 2.
- fissa l'ascissa di r al centro tra le ascisse dei suoi figli;

se r ha il solo figlio sinistro:

- fissa l'ascissa di r pari a quella del figlio +1;

se r ha il solo figlio destro:

- fissa l'ascissa di r pari a quella del figlio –1.

Prima di discutere l'algoritmo, diamo delle definizioni preliminari:

Definizione: Due alberi binari ordinati T' e T'' si dicono *semplicemente isomorfi* se o T' e T'' sono entrambi vuoti oppure tanto i sottoalberi sinistri quanto i sottoalberi destri di T' e T'' sono semplicemente isomorfi.

Definizione: Due alberi binari ordinati T' e T'' si dicono *assialmente isomorfi* se T' e l'albero ottenuto da T'' scambiando il sottoalbero sinistro con il destro di ogni suo nodo sono semplicemente isomorfi.

Informalmente, due alberi sono semplicemente isomorfi se sono ordinatamente uguali, mentre sono assialmente isomorfi se sono l'uno l'immagine riflessa dell'altro.

L'algoritmo di Reingold e Tilford disegna alberi semplicemente isomorfi in modo congruente, e disegna alberi assialmente isomorfi in modo congruente a meno di una riflessione.

Si osservi che l'algoritmo, così com'è, non garantisce che i nodi siano a coordinate intere, cioè che il disegno sia su griglia. Supponiamo infatti di avere le due radici r' ed r'' dei sottoalberi sinistro e destro a distanza 3; allora il padre di r' e di r'' non è a coordinata intera. Questo problema si può risolvere spostando uno dei due disegni in maniera da allontanarlo dall'altro e garantire che l'intervallo tra le due radici sia sempre pari. E' facile dimostrare che questa accortezza garantisce che il disegno ottenuto sia su griglia.

Teorema: *Dato un albero binario ordinato T con n nodi, l'algoritmo di Reingold e Tilford costruisce un disegno Γ di T in $O(n)$ tempo tale che:*

- *Γ è livellato, cioè l'ordinata di ogni nodo è uguale all'opposto della sua profondità;*
- *Γ è piano, rettilineo e strettamente downward;*
- *Γ mantiene l'ordine da sinistra a destra dei nodi;*
- *comunque si scelgano due nodi in Γ, la loro distanza è almeno 1, sia in orizzontale che in verticale;*
- *l'area di Γ è $O(n^2)$;*
- *l'ascissa di ogni nodo padre di due figli è la media delle ascisse dei suoi figli;*
- *alberi semplicemente isomorfi hanno disegni congruenti;*
- *alberi assialmente isomorfi hanno disegni congruenti, a meno di una riflessione rispetto all'asse y.*

Dimostrazione: Introduciamo dapprima il concetto di contorno sinistro e destro di un albero binario con radice r: il *contorno sinistro* (*destro*) di un albero binario T di altezza h, denominato con $cs(T)$ ($cd(T)$) è la sequenza di nodi $v_0,v_1,...,v_h$ tale che v_i è il nodo di profondità i più a sinistra (destra) di T.

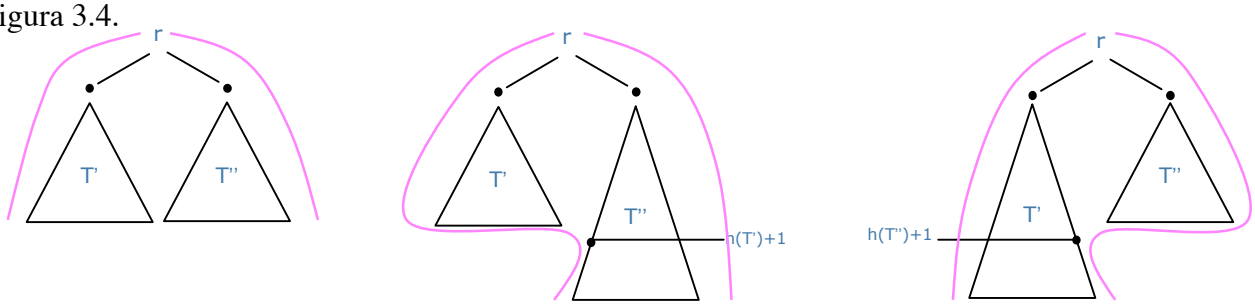
L'algoritmo prevede di mantenere la proprietà che, ad ogni livello, i disegni dei sottoalberi sinistro e destro distino di almeno due unità, cioè che l'i-esimo nodo del contorno destro del sottoalbero sinistro disti almeno 2 dall'i-esimo nodo del contorno sinistro del sottoalbero destro. Memorizzando tali contorni tramite delle liste concatenate che rappresentino le sequenze di nodi costituenti i contorni del sottoalbero, è facile scandirle e, ad ogni passo, accumulare una quantità che indica di quanto sia necessario allontanare i due sottoalberi. La complessità temporale di questo passo è dell'ordine del minimo tra le altezze dei sottoalberi sinistro e destro. Siano h' ed h'' le altezze di T' e T'', rispettivamente sottoalbero destro e sinistro di r. Si definisca ora il nodo w come il nodo a profondità $h'+1$ sul contorno sinistro di T'' nel caso in cui h' sia minore di h'', e come il nodo a profondità $h''+1$ sul controno destro di T' nel caso in cui h'' sia minore di h'. Dopo aver confrontato i contorni livello per livello, è facile – con un tempo aggiuntivo costante – determinare il nodo w.

E' possibile calcolare i contorni sinistro e destro per il sottoalbero $T(r)$ radicato in r, noti i contorni sinistro e destro dei suoi due sottoalberi T' e T'' come segue. Distinguiamo tre casi:

1. se T' e T'' hanno la stessa altezza (v. Figura 3.4 parte sinistra) allora i contorni destro e sinistro di $T(r)$ sono ricavabili da quelli dei suoi sottoalberi destri e sinistri aggiungendo semplicemente la radice.
   $cs(T(r))= r---cs(T');$
   $cd(T(r))=r---cd(T'')$ (dove il simbolo --- rappresenta concatenazione)
2. se T' ha altezza minore di T'' (v. Figura 3.4 centro) allora il contorno destro di $T(r)$ è:
   $cd(T(r))=r---cd(T'');$
   mentre per il contorno sinistro bisogna considerare la differenza di altezze: sia $h''>h'$. Il contorno sinistro di $T(r)$ è costituito dalla concatenazione della radice r, del contorno sinistro di T', e della parte di contorno sinistro di T'' da w in poi.

3. se *T'* ha altezza maggiore di *T''* (v. Figura 3.4 parte destra) si proceda in modo simmetrico al caso precedente.

Figura 3.4.

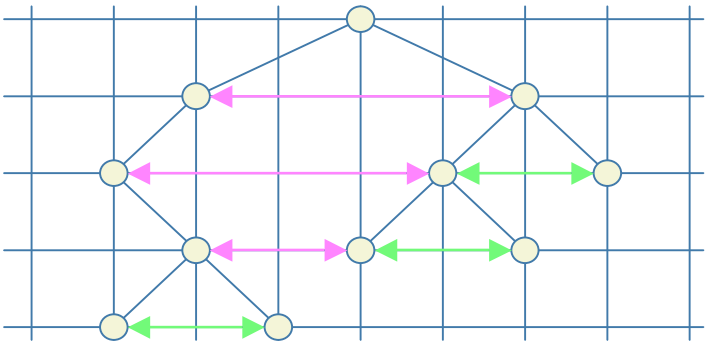


E' facile vedere che, noto *w*, la complessità per l'aggiornamento delle liste è *O(1)*, infatti basta semplicemente concatenare la porzione di lista a partire dal nodo *w* di seguito alla lista del vecchio contorno, ottenendo la lista che rappresenta il nuovo contorno.

La totale quantità di lavoro è dunque quella fatta per confrontare tutte le coordinate, proporzionale alla somma di tutte le minori altezze tra ciascuna coppia di sottoalberi, più una costante per ogni nodo, per mantenere le informazioni sui contorni: detta $h_s(v)$ (rispettivamente, $h_d(v)$) l'altezza del sottoalbero sinistro (rispettivamente destro) del generico nodo *v*, si ha:

$\sum_{v \in T}(O(1)+min(h_s(v), h_d(v))= O(n)+ \sum_{v \in T}(min(h_s(v), h_d(v))$.

Figura 3.5.



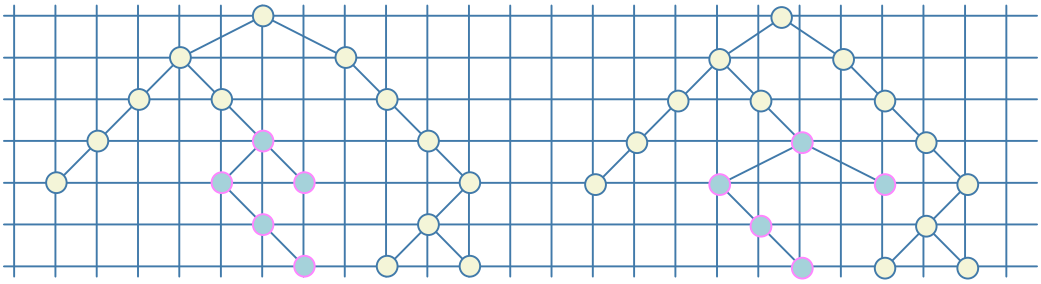
Per stimare la sommatoria, si consideri la Figura 3.5: in essa sono evidenziati con delle frecce tutti i confronti effettuati durante l'intero algoritmo, ed il loro numero corrisponde dunque alla sommatoria; poiché è facile far corrispondere a ciascuna freccia al più un nodo distinto (ad esempio l'estremo sinistro), è immediato verificare che il numero di confronti è limitato da *n*. La sostituzione di questo valore nella sommatoria conclude la prova. **CVD**

L'algoritmo di Reingold e Tilford cerca di ridurre la larghezza del disegno eseguendo un'operazione di compattamento locale basata sulla tecnica del *divide et impera*; tuttavia ciò non garantisce di trovare l'ottimo. In Figura 3.6 è mostrato a sinistra un esempio in cui l'algortimo non trova un disegno di larghezza ottima, come invece è quello di destra; in quest'ultimo disegno si evidenzia come per raggiungere una larghezza ottima sia necessario, ad un passo intermedio, fare una scelta non ottima. Ciò dimostra che una qualunque strategia divide et impera per disegni livellati di alberi binari che non modifichi i disegni calcolati ai passi precedenti, in generale, non potrà mai raggiungere la larghezza ottima.

In effetti, Supowit e Reingold [SR83] hanno dimostrato che il problema di costruire un disegno di un albero binario che soddisfi le proprietà enunciate nel teorema precedente ed abbia larghezza minima si può risolvere in tempo polinomiale con tecniche di programmazione lineare; tuttavia, se si aggiunge la convenzione di disegno su griglia, il problerma diviene NP-arduo. Questo risultato conferma quanto è abbastanza usuale in letteratura: in problemi risolvibili con la programmazione

lineare, il passaggio da variabili reali a variabili intere porta la complessità da polinomiale ad esponenziale.

Figura 3.6.



L'algoritmo descritto in precedenza lavora su alberi binari. Detto r il nodo generico di un albero k-ario radicato e ordinato, e $T_i(r)$ il suo i-esimo sottoalbero, è piuttosto semplice generalizzare tale algoritmo in modo che operi su alberi k-ari qualunque, come segue:

Input: un albero k-ario radicato ordinato T;
Output: un disegno livellato di T;
Passo base:
Se T è costituito da un singolo vertice, disegnalo banalmente;
Passo divide:
ricorsivamente applica l'algoritmo su ciascun sottoalbero del nodo corrente r;
Passo impera:
for $i=2$ to k:

- posiziona il disegno di $T_i(r)$ alla destra del disegno di $T_{i-1}(r)$ in maniera che la loro distanza ad ogni livello sia ≥ 2.
- fissa l'ascissa di r al centro tra le ascisse delle radici di $T_1(r)$ e di $T_k(r)$.

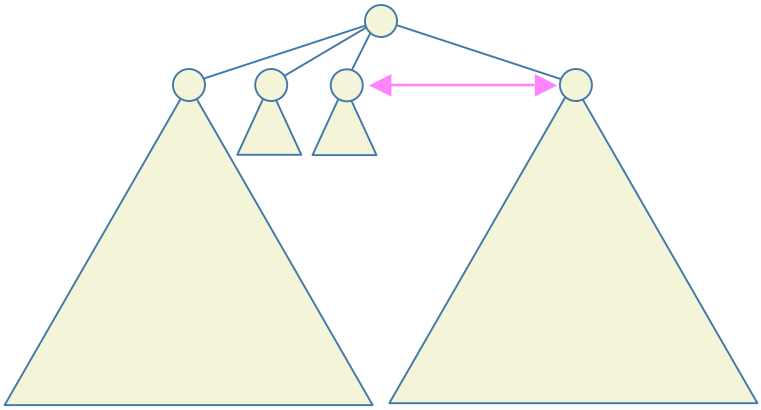
Estendendo le definizioni di alberi semplicemente ed assialmente isomorfi al caso di alberi k-ari, si può enunciare il seguente:

Teorema. *Dato un albero k-ario ordinato T con n nodi, l'algoritmo ora descritto costruisce un disegno Γ di T in $O(n)$ tempo tale che:*

- *Γ è livellato, cioè l'ordinata di ogni nodo è uguale all'opposto della sua profondità;*
- *Γ è piano, rettilineo e strettamente downward;*
- *Γ mantiene l'ordine da sinistra a destra dei nodi;*
- *comunque si scelgano due nodi in Γ, la loro distanza è almeno 1, sia in orizzontale che in verticale;*
- *l'area di Γ è $O(n^2)$;*
- *alberi semplicemente isomorfi hanno disegni congruenti;*
- *alberi assialmente isomorfi hanno disegni congruenti, a meno di una riflessione intorno all'asse y.*

L'algoritmo precedente fornisce disegni ragionevoli per la maggior parte dei casi. Tuttavia, i suoi output soffrono spesso di un problema di sbilanciamento, evidenziato in Figura 3.7: si supponga che T abbia come sottoalberi T_1, T_2, T_3 e T_4, e che T_1 e T_4 siano molto più grandi di T_2 e T_3. Poiché l'algoritmo lavora da sinistra a destra, esso posizionerà T_2 e T_3 molto più vicino a T_1 che non a T_4. Per risolvere il problema, sarebbe necessario spaziare i disegni dei sottoalberi in modo uniforme, o modificando il passo impera [T81] o tramite una fase di post-processing [W90].

Figura 3.7.



3.5. Un algoritmo di disegno upward planare più efficiente

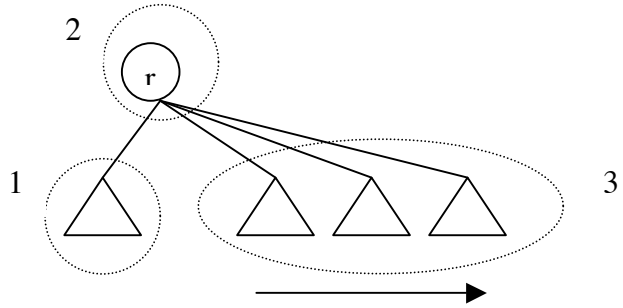
Riferimenti: [GGT96] pp 2-6.

La rappresentazione upward planare è quella più naturale per rappresentare alberi radicati in modo da evidenziarne la struttura gerarchica. Tuttavia, l'area occupata passa da $O(n)$, nel caso dell'H-tree, ad $O(n^2)$, nel caso dell'algoritmo di Reingold e Tilford. E' dunque naturale chiedersi se si riesca a migliorare il risulato di Reingold e Tilford, cioè a garantire ancora un'area $O(n)$ introducendo la convenzione upward ma eliminando l'ortogonale. Crescenzi, Di Battista e Piperno [CDP92] hanno risposto negativamente a questa domanda mostrando che, se il disegno è strettamente upward, esiste una famiglia di alberi che richiede almeno $\Omega(n \log n)$ area, e questa è sufficiente se non si richiede di mantenere l'ordinamento dei nodi. Tuttavia, se ci si accontenta della convenzione upward (e non strettamente upward) si può ottenere di meglio: verrà mostrato con il prossimo algoritmo – dovuto a Garg, Goodrich e Tamassia [GGT96] - che, per ogni albero T con n nodi, è possibile costruire un disegno upward polyline piano in $O(n)$ area. Aggiungendo la condizione che i nodi debbano mantenere lo stesso ordine relativo nella rappresentazione, cioè che l'albero sia ordinato, è nuovamente necessaria e sufficiente area $\Theta(n \log n)$, mentre se il disegno è upward e ortogonale (ma non deve preservare l'ordine dei nodi), si ha un'area di $\Theta(n \log \log n)$. E' interessante osservare che il vincolo di ortogonalità penalizza l'area meno di quanto lo faccia il vincolo di mantenere i nodi ordinati. Esistono algoritmi che garantiscono area lineare per alberi AVL [CP95] e di Fibonacci [CDP92].

L'algoritmo di Garg, Goodrich e Tamassia si basa sulla generalizzazione della visita *in-order* e su un'etichettatura dei nodi detta *upward layering*.

Dato un albero k-ario radicato e ordinato T, la sua visita *in-order* è così definita (vedi Fig. 3.8):

Figura 3.8.



1. Ricorsivamente visita il sottoalbero più a sinistra di T;
2. Visita la radice r di T;
3. Ricorsivamente visita gli altri sottoalberi da sinistra a destra.

Definizione. Una etichettatura *upward layering* λ dei nodi di T è una funzione che assegna ad ogni nodo v di T un intero non negativo $\lambda(v)$ tale che:

1. se w è il figlio più a sinistra di $v \Rightarrow \lambda(v) \leq \lambda(w)$;
2. se w è un altro figlio di $v \Rightarrow \lambda(v) < \lambda(w)$;
3. $\lambda(r)=0$, dove r è la radice di T.

In generale, un albero T può avere più di una etichettatura upward layering. Si dice che un nodo v è *assegnato al livello* i se $\lambda(v)=i$, e che un arco (u,v) *attraversa* un livello i se $\lambda(v) < i < \lambda(u)$. Si definisce inoltre:

- l'*altezza* $H(\lambda)$come $max_{v \in T}\ \lambda(v)$;
- l'*ampiezza* $W(i)$ *del livello* i come il numero di nodi v tali che $\lambda(v) = i$ più il numero di archi che attraversano il livello i;
- l'*ampiezza* $W(\lambda)$come il $max_i\ W(i)$.

Teorema. *Dato un albero qualunque radicato e ordinato T su cui sia definita una etichettatura upward layering* λ *con altezza H e ampiezza W, è possibile dedurre un disegno piano, polyline e upward di T di area* $H(\lambda) \times W(\lambda)$.

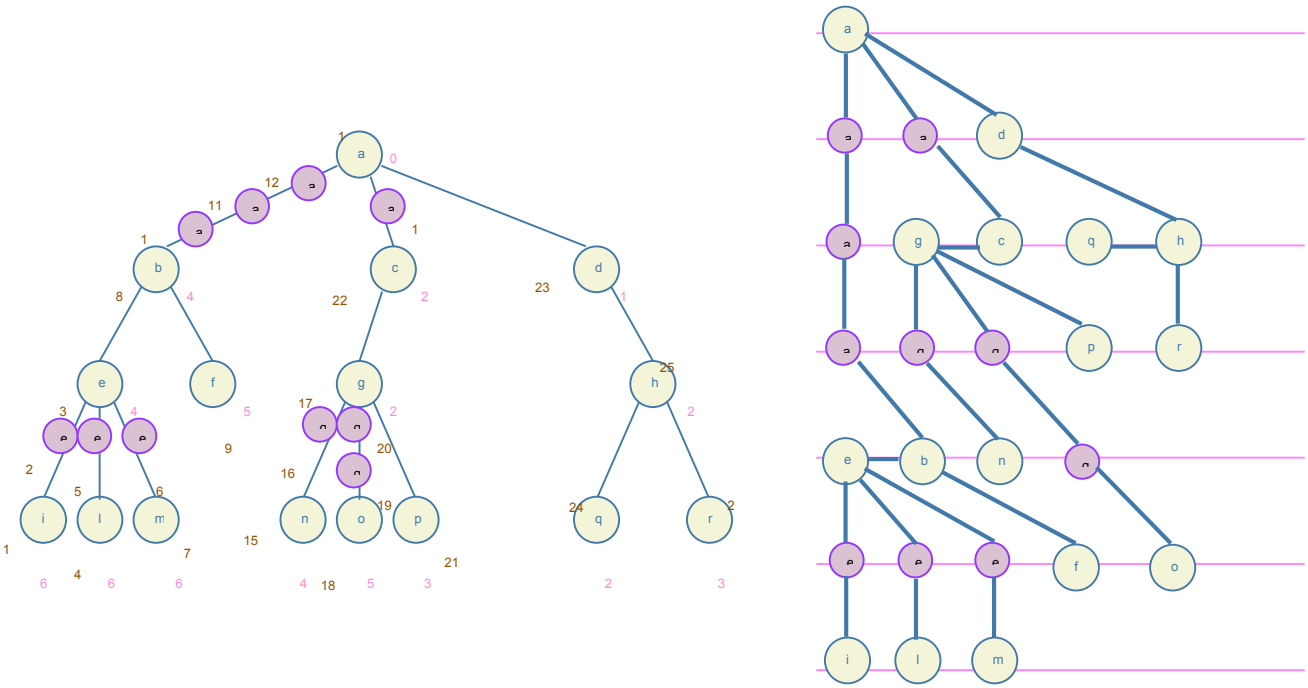
Dimostrazione. L'enunciato verrà provato in modo costruttivo, fornendo un algoritmo di disegno:

Input: l'albero T con radice r, l'etichettatura λ.

Output: un disegno planare, polyline, layered, su griglia e upward di T.

1. Per ogni arco (u,v) tale che $\lambda(u) < \lambda(v) - 1$, si inseriscano $\lambda(v) - \lambda(u) - 1$ nodi fittizi, ottenendo così l'albero T' tale che $\forall (u,v) \in T'$ o $\lambda(u) = \lambda(v)$ oppure $\lambda(u) = \lambda(v) - 1$;
2. $\forall v \in T'$ si ponga $y(v) = -\lambda(v)$;
3. Si effettui una visita *in-order* di T' e si calcoli $n(v) = \#$ di visita *in-order* di v;
4. $\forall v \in T'$ si ponga $x(v) = \#$ di nodi sul livello $\lambda(v)$ che precedono v nella visita *in-order*.

Figura 3.9.



In Figura 3.9 è rappresentato un esempio dell'esecuzione dell'algoritmo ora descritto.

Si osservi che il disegno di T' prodotto dall'algoritmo è:

- su griglia: tutti i nodi (reali e fittizi) hanno coordinate intere;
- rettilineo: ogni arco di T' è disegnato come un segmento;
- upward: ogni nodo padre giace su un livello minore o uguale a quello dei suoi figli;
- piano: si consideri il generico arco (u,v); allora o $\lambda(u) = \lambda(v)$ oppure $\lambda(u) = \lambda(v) - 1$.

  Se $\lambda(u) = \lambda(v)$, si supponga, per assurdo, che l'arco (u,v) sia attraversato da un altro arco; poiché il disegno in output è tale che tutti gli archi congiungono nodi sullo stesso livello o su livelli consecutivi, allora l'unica possibilità di avere un incrocio è che esista un nodo w tale che $\lambda(u) = \lambda(v) = \lambda(w)$ e che w giaccia tra u e v (vedi Figura 3.10, parte sinistra). Tuttavia, il fatto che $\lambda(u) = \lambda(v)$ implica che u sia il figlio più a sinistra di v; d'altra parte, il nodo w è tale che $\lambda(w) = \lambda(u) = \lambda(v)$ ed $n(u) < n(w) < n(v)$; essendo $n(u)<n(w)$, w non si trova nel sottoalbero più a sinistra di u, e quindi non può essere $\lambda(u) = \lambda(w)$; da questa contraddizione si deduce che non può esserci un incrocio di questo tipo.

  Ci si ponga ora nel caso in cui $\lambda(u) = \lambda(v) - 1$ e si ipotizzi ancora la presenza di un incrocio tra l'arco (u,v) ed un altro arco (u',v'); tale incrocio deve necessariamente essere come quello mostrato in Figura 3.10, parte destra. Da come sono posizionati i nodi si deduce che $n(u)<n(u')$ e $n(v')<n(v)$. La disuguaglianza $n(u)<n(u')$ implica che, nella visita *in-order* dell'albero, u precede u' e, di conseguenza, ogni nodo nel sottoalbero radicato ad u precede ogni nodo nel sottoalbero radicato ad u', e quindi in particolare $n(v)<n(v')$. Ma deve valere anche la disuguaglianza opposta, e quindi scaturisce un assurdo. Si conclude che il disegno debba essere piano.

Figura 3.10.

Dal disegno di T' prodotto dall'algoritmo si ottiene un disegno di T su griglia, polyline, upward e piano sostituendo ogni nodo fittizio di T' con una svolta; il disegno è polyline perché gli archi che connettono vertici su livelli non consecutivi sono formati da linee spezzate.

Per determinarne l'area del disegno si osservi che l'altezza e la larghezza del disegno corrispondono rispettivamente all'altezza e all'ampiezza di λ, per cui l'area è $H(\lambda) \times W(\lambda)$. **CVD**

Per concludere l'algoritmo di Garg, Goodrich e Tamassia, rimane da definire come possa essere determinata una etichettatura upward layering di T.

L'algoritmo che calcola l'etichettatura λ costruisce una sequenza ordinata di nodi Λ e marca alcuni di essi in modo che siano verificate le proprietà:

- se u precede v in Λ, allora $\lambda(u) \geq \lambda(v)$;
- un nodo è marcato in Λ se e solo se è il nodo più a sinistra del suo livello.

All'inizio, la sequenza Λ è rappresentata dalla stringa vuota; la costruzione di Λ è realizzata tramite ripetuti inserimenti di contorni sinistri di T che vengono via via eliminati dall'albero; alla

fine Λ conterrà tutti i nodi di T una ed una sola volta, in modo che Λ e la marcatura di alcuni dei suoi nodi definiscano in modo univoco una etichettatura λ.

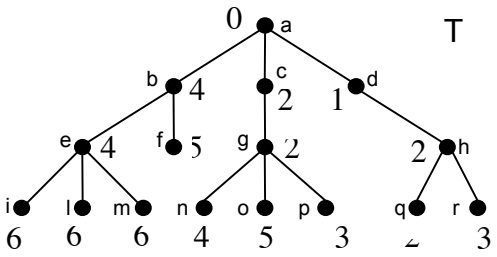
Segue il dettaglio dell'algoritmo:

Input: l'albero T radicato in r;
Output: l'etichettatura λ di T.
Preprocessing: inizializza Λ con la sequenza ordinata di nodi individuati dal contorno sinistro di T dalla foglia alla radice;
for $k=1$ to $\log n$ do
seleziona tutti i nodi $v \in T$ tali che: i) v sia figlio di un nodo in Λ; ii) v non sia ancora in Λ e iii) il sottoalbero di T radicato in v abbia almeno $n/2^k$ nodi;

- ordina i nodi trovati in una sequenza secondo l'ordine dei loro padri in Λ;
- partiziona la sequenza in blocchi di $2^{k/2}$ nodi (l'ultimo blocco può essere più corto)
- per ogni blocco $(v_1, \ldots, v_s)$ considera u= padre di v_1, marca u e inserisci in Λ subito prima di u tutti i contorni sinistri dei sottoalberi radicati in $v_1, \ldots, v_s$ nel loro ordine

Postprocessing: scorri Λ e marca un nodo se ha $n^{1/2} - 1$ nodi consecutivi non marcati che lo precedono direttamente;
$\lambda(v)=i$ se esistono i nodi marcati che seguono v in Λ.

Figura 3.11.



Esempio (vedi Figura 3.11):

Preprocessing: $\Lambda = i\ e\ b\ a$
Ciclo for:

- $k=1;\ n/2^k = 8$: non ci sono sottoalberi con almeno 8 nodi radicati in un nodo figlio di uno dei nodi appartenenti alla stringa; Λ non cambia.
- $k=2;\ n/2^k = 4$: i nodi i cui sottoalberi hanno almeno 4 nodi sono c e d; la sequenza cd viene suddivisa in blocchi di $2^1=2$ nodi (perciò in questo caso non viene suddivisa); a è il padre di c e viene marcato; vengono inseriti in Λ prima di a i contorni sinistri di c e d: ngc e qhd;

$$\Lambda = i\ e\ b\ n\ g\ c\ q\ h\ d\ \underline{a}$$

- $k=3;\ n/2^k = 2$: non ci sono sottoalberi con almeno 2 nodi; Λ non cambia.
- $k=4;\ n/2^k = 1$: vengono selezionati i nodi $l\ m\ f\ o\ p\ r$ e posti in quest'ordine; questa sequenza viene divisa in blocchi di $2^2=4$ nodi : $l\ m\ f\ o\ |\ p\ r$; viene marcato e perché padre di l e g perché padre di p; in Λ vengono aggiunti i contorni sinistri radicati nei nodi nella sequenza (in questo caso vengono aggiunti i nodi stessi, essendo tutti foglie);

$$\Lambda = i\ l\ m\ f\ o\ \underline{e}\ b\ n\ p\ r\ \underline{g}\ c\ q\ h\ d\ \underline{a}$$

Postprocessing:
si scorre Λ da sinistra a destra e viene marcato ogni nodo preceduto da $n^{1/2} - 1=3$ nodi non marcati:

$$\Lambda = i\ l\ m\ \underline{f}\ o\ \underline{e}\ b\ n\ p\ \underline{r}\ \underline{g}\ c\ q\ h\ \underline{d}\ \underline{a}$$

Infine, si scorre da destra a sinistra Λ assegnando a ogni nodo v il valore $\lambda(v) = i$ se esistono i nodi marcati che seguono v in Λ.

Λ = i l m f o e b n p r g c q h d a
λ = 6 6 6 5 5 4 4 4 4 3 2 2 2 2 1 0

Teorema. *L'etichettatura λ prodotta dall'algoritmo precedente è un upward layering di T tale che $H(\lambda)=O(\sqrt{n})$ e $W(\lambda)=O(\sqrt{n})$.*

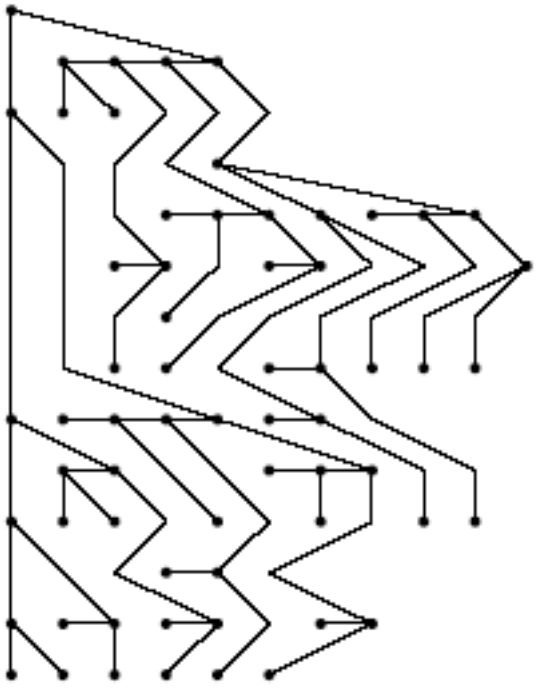
Parte di dimostrazione: Ogni nodo è certamente inserito in Λ. In particolare, dopo il passo k, vengono inseriti in Λ tutti i nodi nel cammino sinistro di qualche sottoalbero con almeno $N/2^k$ nodi. Quindi, ogni nodo sarà etichettato con un valore e tutti i figli di un nodo v precedono v in Λ, per cui $\lambda(v)$ è minore o uguale al λ di tutti i suoi figli. Inoltre, se u è figlio di v, ma non il più a sinistra, allora $\lambda(u) > \lambda(v)$ poiché o v è marcato, o esiste un altro nodo tra u e v in Λ marcato, grazie all'ordinamento dei nodi selezionati. Se ne conclude che λ è un'etichettatura upward layering.

E' immediato vedere che la larghezza dell'etichettatura sia $n^{1/2}$, in virtù della fase di post-processing. Viene omessa la dimostrazione per l'altezza del disegno. **CVD**

Dal teorema segue che l'area del disegno è $H(\lambda) \times W(\lambda)=O(\sqrt{n}) \times O(\sqrt{n})=O(n)$.

Concludiamo questa sezione mostrando, in Figura 3.12, un esempio di esecuzione dell'algoritmo su un albero di 63 nodi.

Figura 3.12.

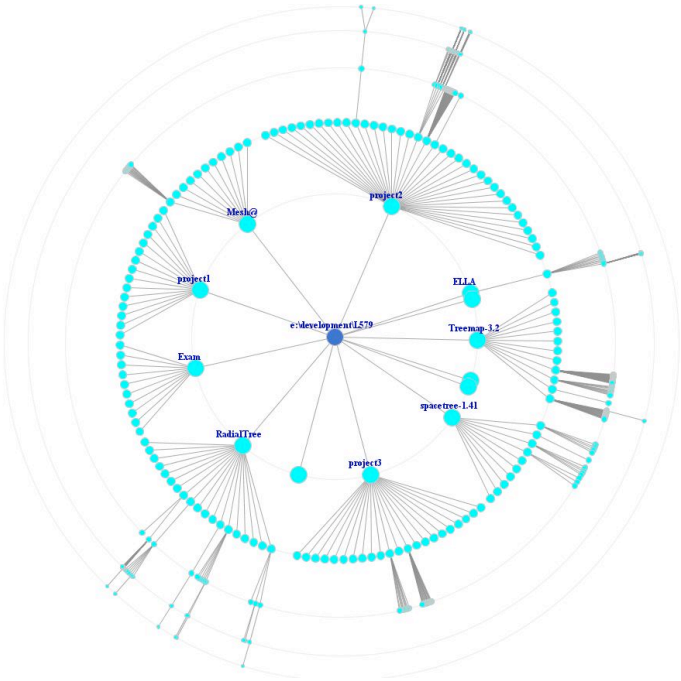


3.6. Un algoritmo di disegno radiale

Riferimenti: [Dal99] pp 52-55

Il disegno radiale è una variante del disegno layered in cui i livelli vengono rappresentati su cerchi concentrici e la radice dell'albero è posta al centro del disegno (vedi Figura 3.13).

Figura 3.13.



E' possibile dedurre degli algoritmi per costruire disegni radiali a partire dagli algoritmi che rappresentano gli alberi come livellati. Nel disegno radiale, infatti, il livello i dell'albero T viene disegnato sulla circonferenza C_i, di raggio $\rho(i)$ (dove $\rho(i)$ è una funzione crescente); per distribuire il disegno su tutta l'area del disegno, sembra ragionevole che ciascun sottoalbero radicato in v venga disegnato all'interno di un settore circolare W_v il cui angolo sia proporzionale al numero di foglie $l(v)$ nel sottoalbero; questa scelta può però provocare degli incroci tra archi, se un arco con estremi in W_v si estende al di fuori del settore circolare andando ad incrociare un altro arco (vedi Figura 3.14 parte sinistra): per prevenire ciò, e garantire la planarità, anziché assegnare al sottoalbero radicato in v l'intero W_v se ne assegna una porzione convessa (si ricorda che una figura si dice *convessa* se il segmento che congiunge due qualsiasi punti al suo interno, o sul suo bordo, si trova tutto all'interno della figura) definita come segue. Il vertice v sia disegnato sulla circonferenza C_i e si consideri la tangente a C_i passante per v; siano a e b i punti in cui tale tangente incontra la circonferenza C_{i+1} (vedi Figura 3.14 parte destra): la regione F_v delimitata dal segmento ab e dai raggi passanti per a e per b è convessa ed è contenuta in W_v. Si restringa ad F_v, con angolo τ_v, la regione assegnata al sottoalbero radicato in v. I figli di v vengono posizionati sulla circonferenza C_{i+1} in base al numero di foglie nei rispettivi sottoalberi; per determinare l'ampiezza β_u del settore W_u assegnato a un figlio u di v si utilizza la formula:

$$\beta_u = min \{ l(u)\, \beta_v / l(v) , \tau_u \}$$

Trovato l'angolo di W_u si posiziona u sulla circonferenza C_{i+1} al centro di W_u.

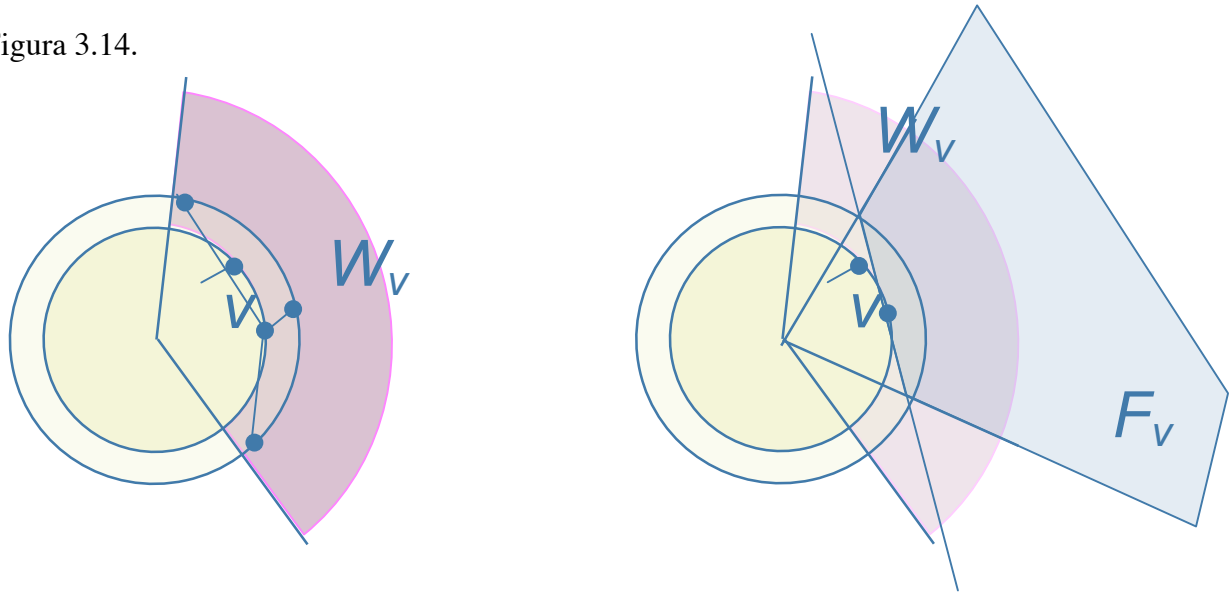
Si noti che β_u è ben definito poiché la somma degli angoli assegnati ai settori dei figli di v è minore o uguale all'angolo del settore assegnato a v infatti, ricordando che il numero di foglie nel sottoalbero radicato in v è pari alla somma del numero di foglie nei sottoalberi radicati ai figli di v, cioè $l(v) = \sum_{i=1..m} l(u_i)$, dove u_i, $i=1, \ldots, m$, è figlio di v, si ha:

$$\sum_{i=1..m} (\beta_{ui}) \leq \sum_{i=1..m} (l(u_i)\, \beta_v / l(v)) = \beta_v / l(v) \sum_{i=1..m} (l(u_i)) = (\beta_v / \cancel{l(v)}) \cancel{l(v)} = \beta_v.$$

Dalla descrizione precedente e dagli algoritmi per disegni livellati si può facilmente derivare un algoritmo che produca un disegno planare radiale che venga eseguito in tempo lineare.

Si definiscano le seguenti regole di risoluzione:

Figura 3.14.



- la minima distanza fra due nodi sia 1;
- $\rho(i)$ sia definita in modo che $\rho(i+1) - \rho(i) = k \ \forall\, i$, dove $k=\rho(1)$, cioè la distanza fra due cerchi consecutivi sia uguale alla distanza fra il centro e il primo cerchio.

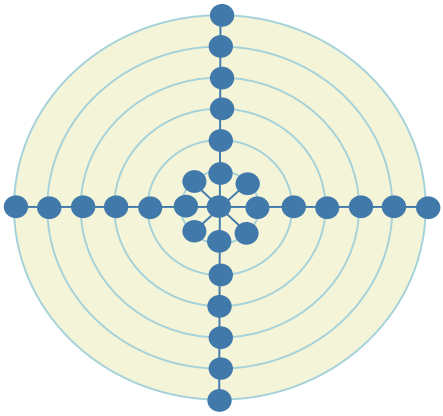
Con questi vincoli, l'area del disegno prodotto da un tale algoritmo è almeno $O(h^2m^2)$, dove h è l'altezza dell'albero, m è il massimo grado dell'albero, e si suppone che la radice abbia grado $O(m)$. Per dimostrarlo, si consideri il perimetro di C_1, pari a $2\pi\rho(1)$, sul quale è necessario posizionare tutti gli $O(m)$ figli della radice, che devono essere a mutua distanza almeno 1; ne segue che il perimetro è $O(m)$, il che significa che $\rho(1)=O(m)$; dalla definizione di $\rho(i)$, il raggio $\rho(h)$ dell'ultimo cerchio è $h\rho(1) = O(hm)$, che implica $A=O(h^2m^2)$, essendo l'area proporzionale al quadrato del raggio. Tutto ciò è corretto se ogni livello i dell'albero contiene $O(im)$ nodi. Se, invece, l'albero è tale che esista almeno un livello i che contiene più di $O(im)$ nodi (ciò è possibile, infatti in un albero m-ario completo sul livello i giacciono $m(m-1)^i$ nodi), allora C_i deve essere grande abbastanza da contenere tutti i nodi del suo livello a distanza almeno 1 l'uno dall'altro. Nel caso peggiore (albero m-ario completo) avremo che $\rho(h)=O((m-1)^h)$, per cui l'area risulterà esponenziale in h, essendo $A=O((m-1)^{2h})$.

Per rendere i cammini dalla radice alle foglie più facili da seguire, è preferibile che essi seguano il più possibile linee rette; sperimentalmente si osserva che, anche se i cammini non sempre sono perfettamente rettilinei, l'algoritmo precedente garantisce che i cammini lunghi siano "abbastanza" rettilinei.

Il disegno radiale è utilizzato per disegnare gli *alberi liberi*, alberi senza una radice prefissata. Infatti, per questi alberi si può sempre individuare un nodo radice fittizio, ed – affinché il disegno sia più equilibrato – tale nodo dovrebbe essere tale che l'altezza dell'albero radicato risultante sia minima. Si dimostra facilmente che un albero può avere un unico o due nodi adiacenti siffatti e questa radice può essere individuata in modo molto semplice con un algoritmo ricorsivo di potatura, tagliando ad ogni passo tutte le foglie finché non rimangano al massimo 2 nodi; se la radice è unica, essa viene piazzata al centro del disegno; se invece le radici sono due, l'arco che le unisce viene disegnato come un segmento orizzontale di lunghezza $2\rho(1)$ il cui punto medio è posizionato in corrispondenza del centro del disegno.

Si conclude questo argomento con l'osservazione che il disegno radiale non è, in generale, efficiente, come mostra la Figura 3.15: se l'altezza dell'albero ed il suo grado massimo sono molto elevati rispetto al numero dei nodi ($\Theta(n)$) si ottiene un disegno di area dell'ordine di n^4.

Figura 3.15.



3.7. Un algoritmo di disegno HV

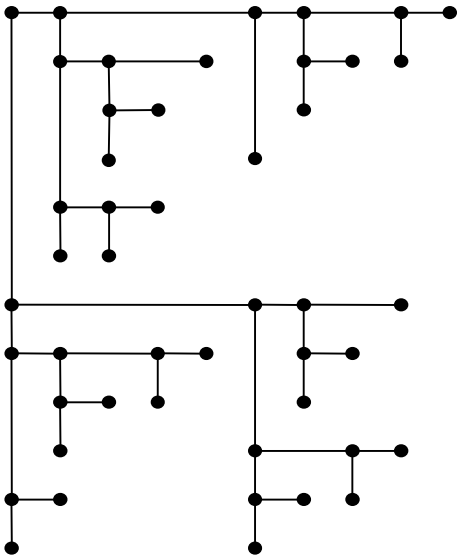
Riferimenti: [Dal99] pp 56-60.

Si definisce *disegno HV (Horizontal Vertical)* di un albero binario un disegno ortogonale, su griglia, rettilineo e downward tale che, per ogni nodo *u* nell'albero:

- un figlio di *u* sia allineato rispetto a *u* orizzontalmente o verticalmente;
- le porzioni di disegno riservate ai sottoalberi radicati ai figli di *u* abbiano intersezione vuota.

Un esempio di un disegno HV è mostrato nella Figura 3.16. Si nota facilmente che il disegno è piano, rettilineo, ortogonale e downward, ma non strettamente downward per la presenza di archi orizzontali.

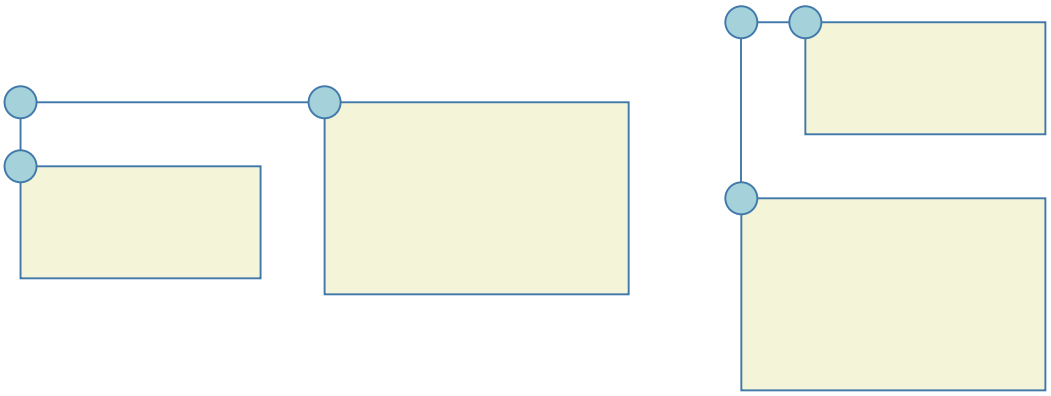
Figura 3.16.



Dato il generico nodo u dell'albero, si considerino i suoi due sottoalberi sinistro e destro e se ne individui quello contenente meno nodi, T_m, rispetto all'altro, T_M. Nella *combinazione orizzontale* il sottoalbero T_m viene posizionato sotto il vertice u, con distanza tra u e la radice di T_m unitaria, mentre T_M viene posizionato a destra di u, con la radice avente ordinata uguale a quella di u e ascissa pari all'ascissa del nodo più a destra di T_m aumentata di 1. La *combinazione verticale* è simmetrica, cioè T_m viene posizionato alla destra di u orizzontalmente a distanza unitaria, mentre T_M viene posizionato sotto u, e la sua radice ha la stessa ascissa di u e ordinata pari a quella del nodo più profondo in T_m aumentata di 1 (vedi Figura 3.17).

Uno schema generale che utilizzi la tecnica del divide et impera, per il disegno HV, è il seguente:

Figura 3.17.



Input: un albero binario T radicato in r;
Output: un disegno HV di T;
passo divide: costruisci ricorsivamente un disegno HV per i sottoalberi destro e sinistro del nodo corrente u;
passo impera: esegui una combinazione orizzontale o verticale dei sottoalberi di u.

Questo algoritmo preserva l'ordinamento dei nodi se le combinazioni orizzontali o verticali al passo impera vengono scelte assecondando le dimensioni dei sottoalberi, cioè, se si sceglie una combinazione orizzontale quando il sottoalbero sinistro è minore di quello destro, e se ne sceglie una verticale altrimenti.

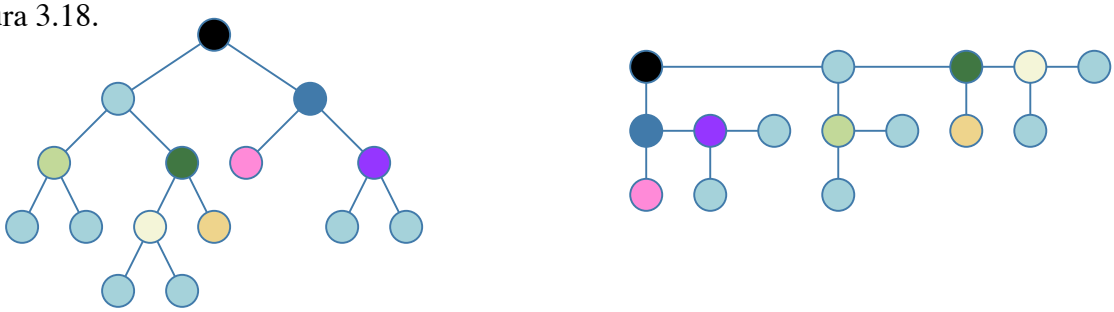
Teorema. *L'area di un disegno HV è* $O(n^2)$.

Dimostrazione. E' sufficiente osservare che tutte le righe e le colonne contengono almeno un nodo, ne segue che sia l'altezza che la larghezza del disegno sono al più $n-1$. CVD

E' possibile ridurre l'area ad $O(n \log n)$ utilizzando una variante dell'algoritmo precedente, denominata Right-Heavy, in quanto essa utilizza solo combinazioni orizzontali e quindi posiziona sempre il sottoalbero più grande a destra (vedi Figura 3.18):

Per come lavora l'algoritmo, è facile notare che, in generale, il disegno prodotto non preserva l'ordine dell'albero.

Figura 3.18.



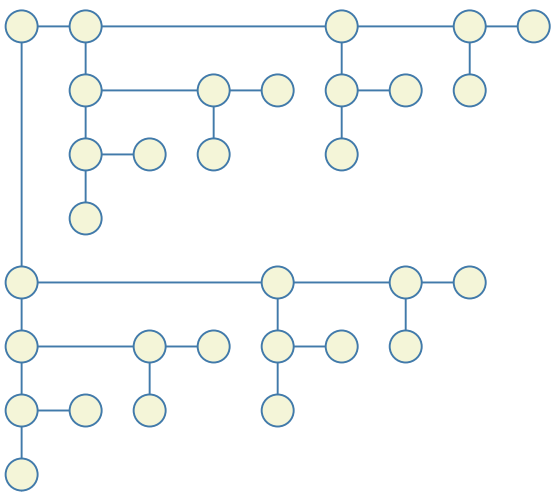
Teorema. *Sia T un albero binario con n vertici. L'algoritmo Right-Heavy produce un disegno HV (e quindi piano, downward, su griglia, rettilineo e ortogonale) di T di area al più (n-1) × log n.*

Dimostrazione. E' immediato accertarsi che l'output dell'algoritmo sia un disegno HV.

La larghezza del disegno è ≤ *n-1* per ragioni analoghe a quelle del teorema precedente. Rimane dunque da dimostrare che l'altezza è *log n*. Supponiamo che *w* sia il vertice più in basso nel disegno di *T*. Poiché sono state usate esclusivamente combinazioni orizzontali, ogni arco verticale ha lunghezza unitaria, ed in particolare avranno lunghezza unitaria gli archi sul cammino dalla radice a *w*. Sapendo che i sottoalberi posizionati a destra hanno dimensione maggiore o uguale di quella dei sottoalberi posizionati in basso, ogni volta che - percorrendo il cammino da *w* a *r* - attraversiamo un arco *(u,v)*, la dimensione del sottoalbero radicato in *u* (padre di *v*) sarà almeno il doppio della dimensione del sottoalbero radicato in *v*; ciò significa che, risalendo da *w* a *r*, verranno attraversati al più *log n* archi verticali di lunghezza unitaria. Ne segue che l'altezza del disegno è O*(log n)*. CVD

A fronte di una buona limitazione sull'area, l'algoritmo Right-Heavy è penalizzato da una cattiva aspect ratio, pari a (*Ω(n/logn)*). Per migliorarla, è stata proposta un'ulteriore variante dell'algoritmo originario per disegni HV: essa prevede l'alternanza di combinazioni orizzontali e verticali. Più precisamente, si utilizzano combinazioni orizzontali per sottoalberi radicati in vertici con profondità dispari e combinazioni verticali per sottoalberi radicati in vertici di profondità pari. Il disegno di un albero binario *completo* realizzato in questo modo ha area *O(n)* e aspect ratio costante (vedi Figura 3.19). Per dimostrarlo, si definiscano *l(h)* e *w(h)* le dimensioni del disegno dell'albero di altezza *h* e si osservi che *l(h)=l(h-1)+1* se *h* è dispari ed *l(h)=2l(h-1)+1* se *h* è pari; inoltre, *l(1) =1*. Le ricorrenze per *w(h)* sono analoghe. Risolvendo, si ottiene che $l(h)=w(h)=2^{h/2}+h$. Poiché l'albero binario è completo, l'area ottenuta è *O(n)*.

Figura 3.19.

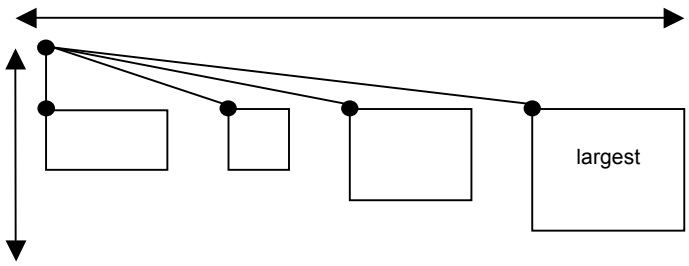


La tecnica dell'algoritmo Right-Heavy può essere estesa anche ad alberi k-ari (Figura 3.20): nel posizionare i sottoalberi del nodo corrente, si mette il più grande a destra; questo fa sì che si perda

la proprietà del mantenimento dell'ordine; inoltre il disegno risultante non è piu ortogonale ma semplicemente rettilineo.

I risultati ottenuti si possono riassumere nel seguente:

Figura 3.20.



Teorema. *Sia T un albero k-ario radicato con n nodi. L'algoritmo Right-Heavy costruisce un disegno Γ di T in O(n) tempo tale che:*

- *Γ è downward, piano, su griglia, rettilineo;*
- *la larghezza di Γ è al più n-1 e la sua altezza al più log n, quindi l'area è O(n log n);*
- *sottoalberi semplicemente ed assialmente isomorfi hanno disegni congruenti.*

3.8. Visualizzazione di gerarchie

Riferimenti:[C04] pp 89-95, 190-193; [BSW02] pp 833-837; [RMC91] pp 189-191.
Per approfondire: [BSW02] tutto; [RMC91] tutto.

Un approccio all'organizzazione di un gran numero di elementi, siano essi files, impiegati o libri, è di raggrupparli ricorsivamente, ottenendo così delle *gerarchie*. Le gerarchie, tipicamente espresse tramite alberi, vengono usate comunemente per l'astrazione di strutture dell'informazione: la struttura organizzativa di un file system può essere rappresentata come una gerarchia, la struttura di un sistema di classificazione è una gerarchia, ed anche la classificazione biologica di tutti gli animali è una gerarchia. Visualizzare gerarchie è dunque uno degli obiettivi fondamentali nel campo della visualizzazione dell'informazione. In tal caso la struttura ad albero ha un significato più ampio di quello visto fin ora, poiché fornisce un significato addizionale cruciale nel rappresentare una struttura più complessa. Se è abbastanza semplice rappresentare gerarchie medio-piccole trasmettendo con successo le informazioni che esse contengono, è decisamente più difficile estrarre informazioni da gerarchie di grandi dimensioni.

Esistono molti metodi per visualizzare gerarchie. Il primo, e più immediato, consiste nell'utilizzare uno degli algoritmi studiati per disegnare alberi in 2D, ma essi – se evidenziano la struttura gerarchica - non producono disegni che sfruttano lo spazio al meglio; se sono ottimi dal punto di vista dell'area non garantiscono nemmeno un disegno strettamente downward. Nel caso di alberi di grandi dimensioni, perciò la loro applicabilità è limitata.

Un metodo alternativo consiste nell'utilizzo delle *tree maps*, descritto da Shneiderman [S92], che ricopre tutto lo spazio disponibile tramite rettangoli e permette, quindi, di rappresentare gerarchie ben più grandi. Il lato positivo delle tree maps è che esse naturalmente portano un'informazione aggiuntiva data dalla dimensione dei rettangoli. Un concetto correlato è quello delle *finestre elastiche* [KS96], in cui lo schermo è ricoperto da tutte le finestre aperte, in varie dimensioni, in

modo che l'utente possa riportare in primo piano la finestra desiderata. Una variante della tree map è il *sun burst*, in cui si cerca di fondere l'idea della tree map con quella del disegno radiale.

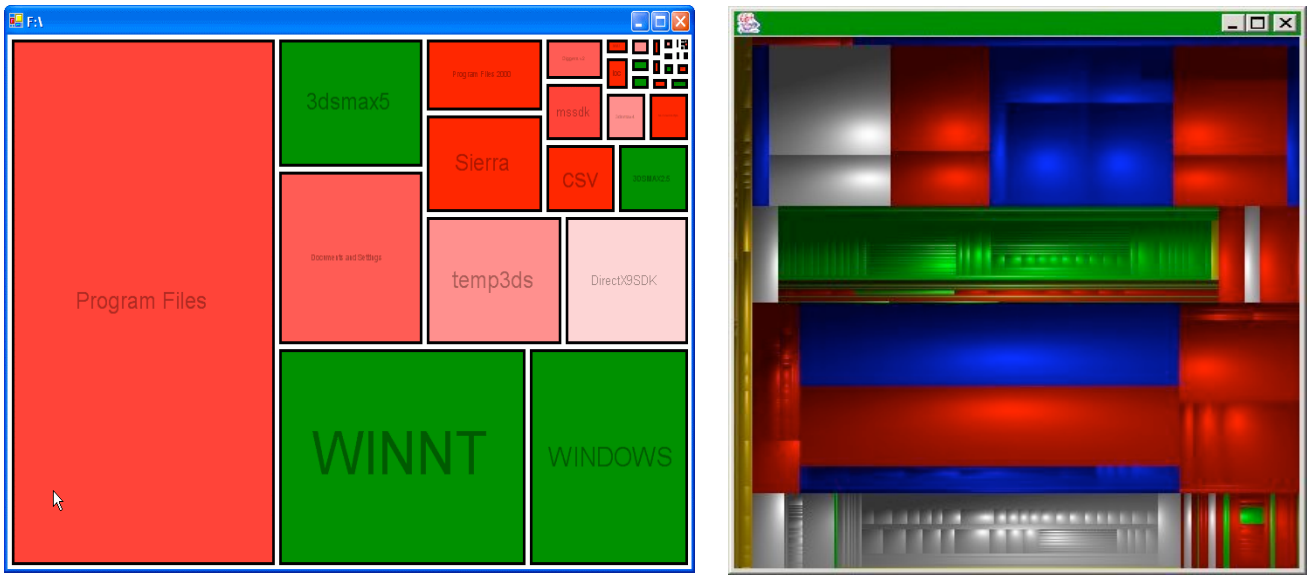
Un altro metodo di visualizzazione è quello di passare dal 2D al 3D; questo consentirebbe di avere "più spazio" fornito dalla terza dimensione. Tuttavia questo approccio porta notevoli problemi, dati dall'occlusione visiva degli oggetti che si trovano dietro ad altri. Un metodo che ovvia a questi inconvenienti è quello che si serve dei *cone trees*, che integra il disegno 3D con le trasparenze e l'animazione. Questa tecnica fu sviluppata per la prima volta da Robertson et al. [RMC91], e l'idea ebbe un profondo impatto sulla visualizzazione dell'informazione, venendo adottata per la progettazione di molti sistemi che visualizzano strutture gerarchiche. Il problema di queste strutture è che riescono a visualizzare ancora un numero relativamente basso si informazioni. Un tentativo decisamente originale di ovviare a tutti i problemi precedenti è costituito dagli *alberi botanici* [KWW??].

Nel seguito saranno esposti gli algoritmi per costruire queste strutture.

3.8.1. Tree Maps

La tecnica di visualizzazione della tree map è una tecnica classica che supporta la navigazione nella gerarchia. Le tree maps utilizzano un algoritmo di riempimento dello spazio che riempie ricorsivamente aree rettangolari con componenti della gerarchia (vedi Figura 3.21).

Figura 3.21.



Uno dei fattori principali a favore delle tree maps è la loro continua crescita nell'utilizzo, dalla loro ideazione in poi. Molti approcci sono stati tentati, in una filosofia incentrata sull'utente. In generale, un algoritmo per disegnare una tree map lavora dividendo l'area visualizzata in una sequenza annidata di rettangoli le cui aree corrispondono ad un attributo dell'insieme dei dati, combinando gli aspetti positivi dei diagrammi di Venn (inclusione) con quelli dei diagrammi a torta (suddivisione di aree). Le tree maps scalano bene e possono essere usate per visualizzare insieme fino ad un milione di elementi.

L'algoritmo *slice-and-dice* dell'articolo originario sui tree maps [S92] usa linee parallele per dividere un rettangolo che rappresenta un certo elemento in rettangoli che rappresentano i suoi figli. Ad ogni livello della gerarchia l'orientamento delle linee - orizzontale o verticale - viene cambiato. Sebbene semplice da implementare, questo algoritmo produce disegni difficili da leggere, per via dei rettangoli troppo sproporzionati e quindi difficili da selezionare, confrontare in dimensione, ed etichettare.

Altri autori hanno quindi tentato di risolvere alcuni dei problemi identificati nelle prime versioni: se i rettangoli molto sottili tendono a causare problemi all'utente, un rettangolo con una forma che tende al quadrato è più facile da osservare. Perciò, si è proposto di bilanciare la aspect ratio dei rettangoli in modo da evitare rettangoli troppo sottili. Tuttavia questo approccio introduce nuovi problemi: innanzi tutto, i cambiamenti nell'insieme dei dati possono causare dei cambiamenti enormi nel disegno prodotto, mentre l'output dell'algoritmo slice-and-dice cambia in modo continuo rispetto alla variazione dell'input; i grandi cambiamenti sono particolarmente sgradevoli in quest'ambito perché spesso le tree maps vengono impiegate per descrivere dati che cambiano istante per istante (ad esempio per monitorare i portafogli in borsa), ma anche quando i dati dovessero cambiare solo occasionalmente uno sconvolgimento della visualizzazione creerebbe problemi nel ritrovamento degli elementi da una visualizzazione all'altra. Inoltre, gli algoritmi che mirano a migliorare l'aspect ratio difficilmente preservano l'ordine delle strutture, mentre l'ordine intrinseco dei dati è spesso un punto importante (si pensi all'ordine cronologico per l'analisi delle azioni, che permette di valutare correttamente gli interessi, o al semplice ordine alfabetico utile in molti casi). Per risolvere questo problema è stato proposto un algoritmo per produrre una tree map ordinata, che mantenga l'ordine originario ma eviti i rettangoli molto sottili [SW01].

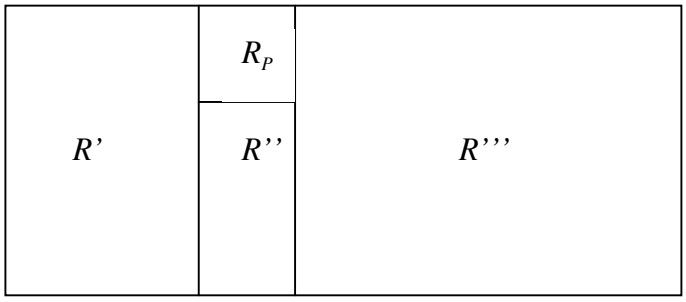
Lo schema generale di un tale algoritmo si basa su un semplice processo ricorsivo, in parte ispirato dall'idea di trovare l'analogo bidimensionale dell'algoritmo di Quick-Sort. Nel primo passo viene scelto un elemento speciale, il *pivot*, che va posizionato a contatto del bordo del rettangolo che contiene i rettangoli considerati; nel secondo passo gli elementi rimasti vengono assegnati a 3 grandi rettangoli che permettono di ricoprire il resto dell'area; infine, l'algoritmo viene applicato ricorsivamente a questi 3 rettangoli.

L'algoritmo può essere descritto come segue:

Input: Rettangolo R da suddividere;
Lista di elementi con area L_1, L_2, ..., L_n;
Output: Rettangoli R_1, R_2, ..., R_n corrispondenti agli elementi.

1. Se $n<=4$, posiziona i rettangoli secondo Figura 3.22
2. Sia il pivot P l'elemento con area maggiore
3. Dividi R nelle 4 aree mostrate in Figura 3.22
4. Posiziona P nel rettangolo R_P
5. Dividi gli elementi della lista diversi da P in 3 liste, L', L'' ed L''' da posizionare in R', R'' ed R''' rispettivamente. L' consiste di tutti gli elementi con indice minore di P nell'ordinamento; Dividi gli elementi rimanenti in L'' ed L''' in modo tale che gli elementi in L'' abbiano indice minore degli elementi di L''' e l'aspect ratio di R_P sia più vicino possibile ad 1.
6. Ricorsivamente inserisci L', L'' ed L''' in R', R'' ed R'''.

Figura 3.22.



Questo algoritmo può essere considerato anche con delle piccole variazioni, che consistono sostanzialmente nel modificare la scelta del pivot. L'algoritmo descritto infatti sceglie il pivot come l'elemento di area maggiore con l'idea che un elemento più è grande e più è difficile da inserire.

Approcci alternativi consistono nel selezionare l'elemento centrale della lista ordinata in modo da creare in qualche modo un disegno bilanciato, oppure nel selezionare P in modo da creare le liste L' ed L''' approssimativamente della stessa area totale, calcolando per ciascun elemento quale sarebbe l'area di L' ed L''' se esso fosse il pivot; anche qui lo scopo è quello di generare dei disegni bilanciati.

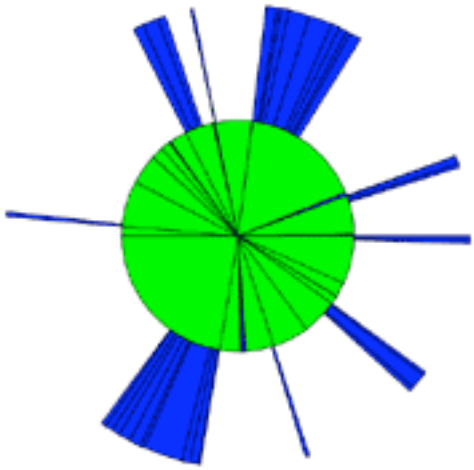
Tutte queste varianti hanno comunque la proprietà di creare disegni che preservano approssimativamente l'ordine dell'indice degli elementi se si va da sinistra a destra e dall'alto verso il basso.

Sono state condotte delle sperimentazioni per valutare la bontà delle varianti ora descritte [BSW02].

3.8.2. Sun bursts

La struttura di *sun burst* è simile a quella di tree map, ma è circolare: la radice della gerarchia è al centro della visualizzazione, mentre i livelli successivi sono irraggiati nei confini definiti dall'angolo assegnato al padre (si noti in questo la similitudine con il disegno radiale). Essa ha diversi punti in comune con il tree map; uno di questi è che entrambe possono essere considerate come due strutture volte al riempimento dello spazio: la tree map riempie completamente un'area rettangolare mentre il sun burst parzialmente un'area circolare (Figura 3.23).

Figura 3.23.



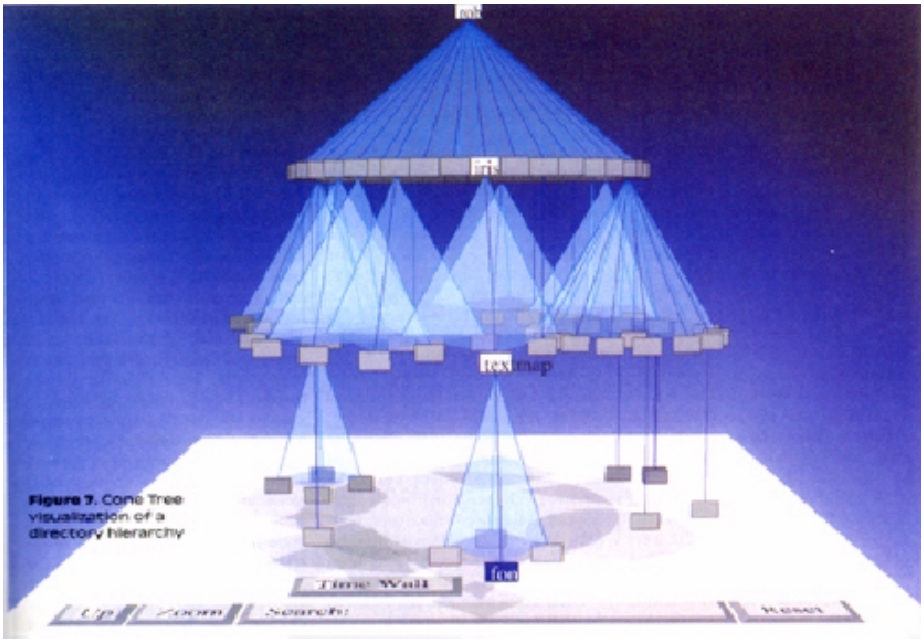
3.8.3. Cone trees

I *cone trees* sono delle strutture tridimensionali interattive, realizzate tramite ombreggiature, trasparenze e animazioni (Figura 3.24).

Essi sono stati introdotti nel '91 [RMC91] proprio per visualizzare strutture gerarchiche, che vengono così rappresentate uniformemente nello spazio tridimensionale. I nodi sono rappresentati tramite rettangoli di dimensione fissa in modo da poter contenere delle etichette; la radice della gerarchia è posta in corrispondenza del centro della faccia superiore del parallelepipedo che contiene il disegno, ed è l'apice di un cono lungo la cui base sono uniformemente posti i figli. Il secondo livello di nodi viene rappresentato sotto il primo, con i figli sul bordo della base di altri coni. L'aspect ratio dell'albero è fissata in modo da farlo sempre entrare nello spazio a disposizione. I diametri di base dei coni su ciascun livello sono ridotti in una progressione che assicuri che l'ultimo livello entri nella larghezza dello spazio a disposizione. I coni sono ombreggiati e

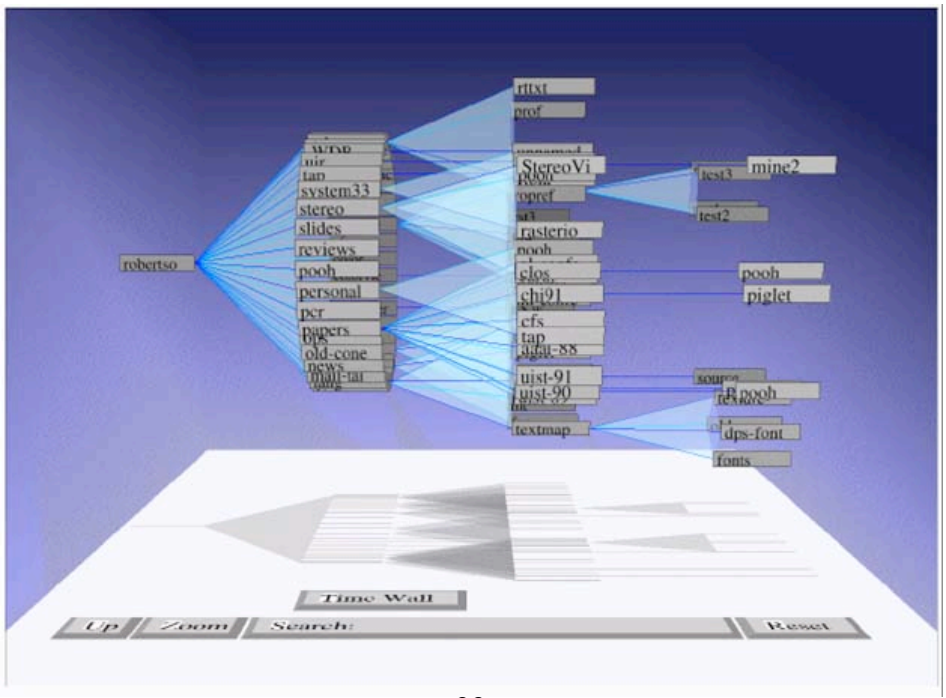
trasparenti in modo da non percepirli come blocchi che coprono la visuale di quello che c'è dietro di loro.

Figura 3.24.



Quando viene selezionato un nodo con il mouse, l'intero cone tree ruota in modo che il nodo selezionato ed ogni nodo sul cammino da quel nodo alla radice si trovino sul davanti del disegno. Le rotazioni di tutti i coni sono condotte in parallelo, scegliendo l'angolo di rotazione minore, e sono animate, in modo che l'utente possa seguire la trasformazione ad una velocità tale che il suo sistema di percezione possa tracciare. Si osservi che l'animazione è fondamentale affinché l'utente comprenda la modifica del disegno: senza animazione l'utente avrebbe bisogno di diversi secondi per comprendere come si è modificato il disegno mentre con l'animazione questo tempo si riduce ad un secondo. Poiché spesso il rettangolo per l'etichetta non è di dimensione sufficiente, essa viene mostrata solo sul cammino corrispondente al nodo selezionato. Alternativamente, è possibile posizionare l'albero in orizzontale in modo che le etichette risultino più lunghe; in tal caso la struttura prende il nome di *cam tree* (vedi Figura 3.25).

Figura 3.25.

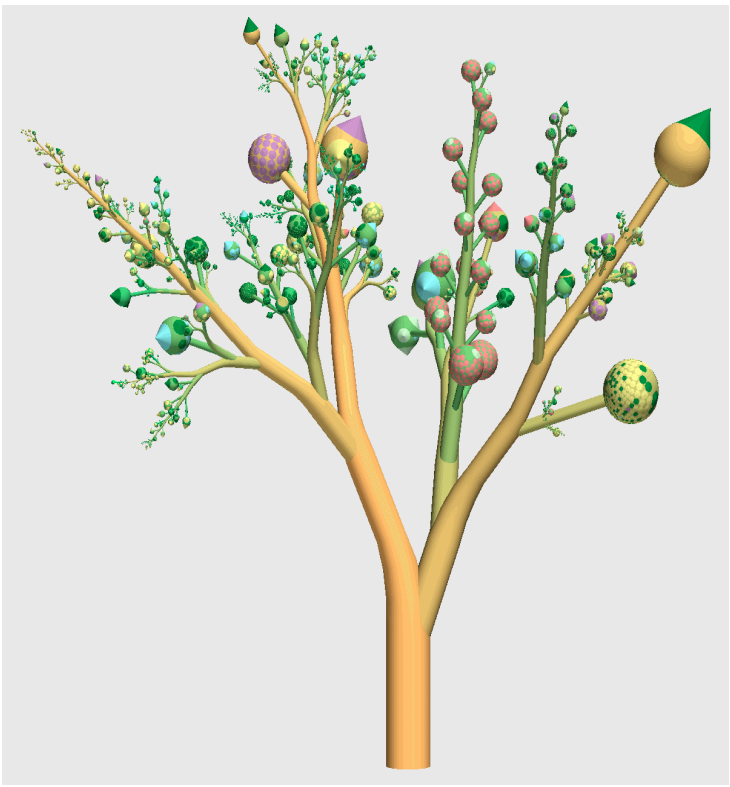


La gerarchia viene presentata in 3 dimensioni per massimizzare l'uso effettivo dello schermo e rendere possibile la visualizzazione dell'intera struttura. Infatti, una rappresentazione 2D della stessa struttura non entrerebbe nello schermo, e l'utente dovrebbe o usare la scroll bar o accontentarsi di una vista ridotta dell'albero. Poiché molte gerarchie tendono ad essere larghe ma non molto profonde, il cone tree sembra funzionare meglio. Per capire analiticamente questo effetto, si consideri l'aspect ratio di un albero 2D: se ci sono l livelli ad ogni nodo ha b figli, la larghezza della base è b^{l-1} e l'aspect ratio è b^{l-1}/l, cioè cresce esponenzialmente, e peggiora tanto più quanto più cresce b. Invece, la aspect ratio del cone tree viene fissata in funzione dello spazio a disposizione, aggiustando l'altezza di ogni livello e il diamentro dei coni. Sebbene fissare l'aspect ratio significhi introdurre una limitazione sul numero di livelli che possono effettivamente essere visualizzati (circa 10), ciò rende i cone trees indipendenti dal numero dei nodi, dal loro grado e dal numero dei livelli (fintanto che non si raggiunga il limite).

3.8.3. Alberi botanici

Recentemente [KWW??], è stata ideata una struttura di visualizzazione botanica per gerarchie molto grandi – ad esempio file systems, detta *albero botanico* (Figura 3.26). L'idea generale è quella di disegnare un albero in cui si distinguano facilmente grandi files (frutti a cono, il cui colore dà indicazioni sul tipo), directories con grandi e/o tanti files (frutti a puntini), e directories con grande contenuto (rami grossi). Gli alberi botanici hanno un'estetica interessante, ma non è ancora chiaro quali siano le applicazioni reali in cui essi siano veramente utili.

Figura 3.26.



RIFERIMENTI BIBLIOGRAFICI

[BSW02] B.B. Bederson, B. Shneiderman and M. Wattenberg. Ordered and Quantum Trremaps: Making Effective Use of 2D Sapce to Display Hierarchies. ACM Trans. Graph. 21(4), 833-854, 2002.

[BK80] Brent, Kung: >> H-tree. Trovare riferimento preciso

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

[CDP92] P. Crescenzi, G. Di Battista, A. Piperno: A Note on Optimal Area Algorithms for Upward Drawings of Binary Trees. *Comput. Geom. Theory Appl.*, 2, 187-200, 1992.

[CP95] P. Crescenzi, A. Piperno: Optimal-Area Upward Drawings of AVL-Trees. *Proc. Of DIMACS International Workshop on Graph Drawing (GD'94), Lecture Notes in Coimputer Science 894*, 307-317, 1995.

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

[GGT96] A. Garg, M.T. Goodrich, R. Tamassia: Planar Upward Tree Drawings with Optimpal Area. *Internat. J. Comput. Geom. Appl.*, 6, 333-356, 1996.

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

[KWW??] E. Kleiberg, H. van de Wetering and J.J. van Wijk. Botanical visualization of Huge Hierarchies. .....

[KS96] E. Kandogan, B. Shneiderman: Elastic windows: Improved spatial layout and rapid multiple window operations. *Proc. of Advanced Visual Interfaces Conference 1996.*

[L80] C. E. Leiserson: Area-efficient graph layouts (for VLSI), Proc. *21th IEEE Symp. On Found. Of Compu. Sci (FOCS'80)*, 270-281, 1980.

[RT81] E. Reingold, J. Tilford: Tidier Drawing of Trees. *IEEE Trans. Softw. Eng.*, SE-7, no. 2, 223-228, 1981.

[RMC91] G.G. Robertson, J.D. Mackinlay and S.K. Card. Cone Trees: animated 3D visualizations of hierarchical information. *ACM ???* , 189-194, 1991.

[S92] B. Shneiderman. Tree Visualization with treemaps: A 2-D space-filling approach. *ACM Trans. Graph.* 11(1), 92-99, 19992.

[SW01] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. Proc. of *IEEE Information Visualization (InfoVis 01)* 73-78, 2001.

[SR83] K.J. Supowit, E.M. Reingold: The Complexity of Drawing Trees Nicely. *Acta Inform.* 18, 377-392, 1983.

[T81] J.S. Tilford: Tree Drawing Algorithms. Tech. Rep. UIUCDCS-R-81-1055, Dept. Of Computer Science Univ. Of Illinois at Urbana-Champaign, 1981.

[V81] L. Valiant: Universality Considerations in VLSI Circuits. *IEEE Trans. Comput.*, C-30, no. 2, 135-140, 1981.

[W90] I.Q. Walker: A node-positioning Algorithm for General Trees. *Softw. – Pract. Exp.* 20(7), 685-705, 1990.