Composizione di uno *Stemma codicum* con Ti*k*Z

Jerónimo Leal, Gianluca Pignalberi

Sommario

Sebbene LaTeX abbia ottimi pacchetti dedicati alla composizione di uno *stemma codicum* più o meno complesso, Ti*k*Z fa ottenere risultati almeno altrettanto buoni senza bisogno di introdurre maggior complessità notazionale.

Abstract

While LaTeX has very good packages to typeset a more or less complex *stemma codicum*, Ti*k*Z allows to reach at least results as good without requiring harder notational complexity.

Premessa

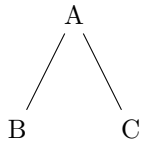
Gli autori di questo articolo, pensato poche settimane dopo la pubblicazione del loro libro (Leal e Pignalberi, 2012), lo hanno completato solo dopo la presentazione dell'articolo di Matteo Fadini presente su questo stesso numero di ArsTeXnica. Quell'articolo ha convinto gli autori a esplorare ulteriormente le capacità di Ti*k*Z e a tentare di realizzare gli stessi esempi complessi.

I risultati confutano i timori espressi da Fadini in una conversazione privata (*TikZ potrebbe non essere in grado di comporre gli stessi schemi complessi di* xytree).

Questa premessa è anche un esplicito ringraziamento allo stesso Fadini per aver ispirato e discusso con gli autori le parti dell'articolo non previste (e neanche pensate) in origine.

1 Introduzione

L'elaborazione di uno *stemma codicum* richiede strumenti adeguati per restituire chiaramente le dipendenze dei manoscritti. L'articolo di Matteo Fadini (pubblicato alla pagina **??** e seguenti, Fadini (2012)) descrive synttree (van Zuijlen, 2009; Pignalberi e Bini, 2008) e xytree (Un, 2006). Mentre il primo è adatto a costruire solo schemi molto semplici quali sono gli alberi,[1] il secondo è in grado di costruire schemi più complessi come i grafi in cui un nodo può derivare da più genitori. La notazione imposta da xytree non ha la stessa semplicità di quella di synttree e richiede all'utente di avere già uno schema mentale esatto da riprodurre:




```
\begin{tikzpicture}
\node{A}
  child{node{B}}
  child{node{C}}
;
\end{tikzpicture}
```

Figura 1: Albero con due sole foglie (B e C) figlie della radice (A).

mentre synttree chiede solo di elencare ogni nodo dell'albero seguito dai suoi figli (*organizzazione logica*) per generare da sé l'albero, xytree chiede l'esatta posizione spaziale di ogni nodo in un'ipotetica griglia (o tabella) e l'elenco di tutti i collegamenti, o archi (*organizzazione fisica*). Dunque l'utente è l'unico responsabile dell'aspetto finale del grafo.

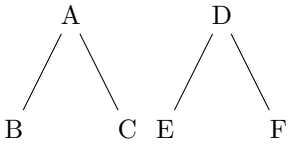
Ti*k*Z (Tantau, 2010) offre un'interfaccia intermedia: dobbiamo indicare l'elenco dei nodi seguiti dai figli, cosa che genera direttamente l'albero, dopodiché possiamo specificare una serie di collegamenti supplementari applicabili solo ai nodi precedentemente etichettati. Nel corso di questo breve articolo chiariremo tutti i dettagli notazionali ricorrendo ad alcuni esempi concreti tratti da una tesi del dottorando Fabrizio Tiddia in preparazione alla Pontificia Università della Santa Croce e consistente nell'edizione critica della *Passio Sancti Alexandri*.

Il paragrafo 2 introduce la produzione di uno o più alberi con Ti*k*Z. Il paragrafo 3 spiega come segnalare le contaminazioni tra i manoscritti mediante l'aggiunta di archi arbitrari. Infine il paragrafo 4 espone come comporre con Ti*k*Z alcuni degli *stemmata codicum* realizzati da Matteo Fadini nel suo articolo Fadini (2012).

2 Costruzione di alberi semplici con Ti*k*Z

Un albero è una struttura comunemente usata per rappresentare delle strutture gerarchiche: è costituito da *nodi* e *archi*, dove i nodi sono dei simboli contenenti una didascalia, cioè un'informazione, mentre gli archi sono dei segmenti che uniscono due nodi di due livelli adiacenti. Un albero contie-

1. Un albero è una struttura dati formata da nodi — elementi con un contenuto — e da archi — collegamenti tra i nodi. Ogni nodo di un albero può appartenere a uno e uno solo dei nodi al livello superiore, cioè avere un solo genitore. L'unica eccezione è la radice: occupa il livello supremo.




```
\begin{tikzpicture}
\node{A}
  child{node{B}}
  child{node{C}}
;
\node at (2,0) {D}
  child{node{E}}
  child{node{F}}
;
\end{tikzpicture}
```

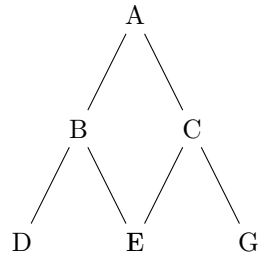
Figura 2: Due alberi nello stesso ambiente tikzpicture.

ne sempre una *radice*, un nodo senza "genitore", da cui possono dipartire diversi rami, quindi con eventuali *figli*. I figli che non hanno a loro volta dei figli sono definiti *foglie*.

La sintassi di Ti*k*Z ricalca la nomenclatura esposta, come vediamo nell'esempio mostrato nella figura 1: dentro l'ambiente tikzpicture apriamo (radichiamo) l'albero col comando `\node` a cui diamo una didascalia (da non confondere con l'etichetta, come vedremo più avanti), in questo caso "A". Quindi elenchiamo tutti i suoi figli dal primo (quello più a sinistra) all'ultimo (quello più a destra): generiamo ognuno dei figli con l'attributo `child`. Questo prende tra parentesi graffe l'attributo `node`, la didascalia (anch'essa tra parentesi graffe), e l'eventuale lista dei figli del nodo corrente. La descrizione dell'albero termina con un ';'.

Niente ci vieta di disegnare due o più alberi nello stesso ambiente tikzpicture, come vediamo nella figura 2. Dobbiamo solo avere l'accortezza di specificare la posizione della radice degli alberi successivi al primo per non averli tutti sovrapposti a esso.

Naturalmente non siamo vincolati a produrre alberi con una profondità predeterminata dato che possiamo aggiungere tutti livelli che servono. La figura 3 mostra lo stesso albero della figura 1 in cui però le vecchie foglie ora hanno dei figli. Noterete che i rami relativi alle foglie "E" e "F", provenienti da due *parenti* (due o più nodi allo stesso livello) adiacenti, sembrano puntare alla stessa foglia. In realtà le foglie sono due sovrapposte perché Ti*k*Z, a differenza di synttree, non tiene conto automaticamente degli ingombri, che devono essere corretti manualmente specificando la distanza tra parenti. Vediamo come applicare la correzione nella figura 4. Per ognuno dei livelli da "ritoccare" alteriamo uno degli attributi di `style`: `sibling distance`. Se avessimo specificato, ad esempio, solo l'attributo del primo livello (`level 1`), tutti i successivi livelli avrebbero ereditato il suo valore, quindi anche i




```
\begin{tikzpicture}
\node{A}
  child{node{B}
    child{node{D}}
    child{node{E}}}
  child{node{C}
    child{node{F}}
    child{node{G}}}
;
\end{tikzpicture}
```

Figura 3: Albero con una radice, due figli (nodi intermedi) e quattro foglie (nodi terminali). Le due foglie denominate "E" e "F" sono sovrapposte.

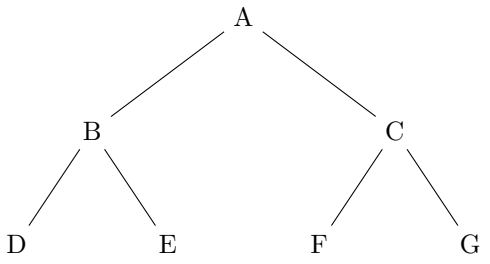
nodi del secondo livello avrebbero avuto una distanza tra essi di 40 mm anziché 20. Ogni livello è separato dal successivo da una virgola e tutte le descrizioni sono passate come parametro opzionale all'ambiente tikzpicture.

Dopo aver visto alcuni casi didattici vediamo un caso concreto tratto dalla tesi citata. La figura 5 mostra uno *stemma codicum* il cui disegno non pone problemi: non ci sono due parenti adiacenti che abbiano figli, quindi non ci sono possibilità di sovrapposizioni e dunque non c'è la necessità di inserire correzioni di sorta.

Ricordiamo che quello che abbiamo finora chiamato "radice" sarà più correttamente definito *archetipo*. Abbiamo già discusso la sintassi di un albero in Ti*k*Z, quindi non ci soffermeremo in ulteriori descrizioni. L'unico appunto che possiamo fare riguarda la stesura del codice: usiamo l'indentazione solo per comodità umana. Il compilatore non necessita di spazi particolari ma solo del codice conforme alle regole del linguaggio.

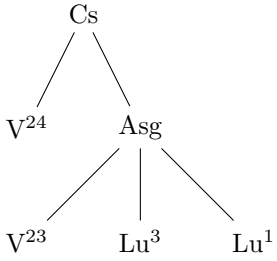
Passare a uno *stemma codicum* con un maggior numero di livelli non cambia la difficoltà (sebbene servirà maggiore attenzione nello scrivere tutte le parentesi correttamente). Lo stemma mostrato nella figura 6 è notevolmente più grande di quelli visti finora, didattici o no. Se è vero che la maggior grandezza è motivo di maggior scrittura di codice, è anche vero che il disegno non presenta alcuna difficoltà, visto che non ci sono sovrapposizioni da correggere né altre caratteristiche che discuteremo nel paragrafo 3: siamo ancora nel caso di alberi.

Quello che piuttosto possiamo notare è che nel codice alcuni nodi hanno una lettera tra parentesi tonde dopo l'attributo `node`. Questa è un'etichet-



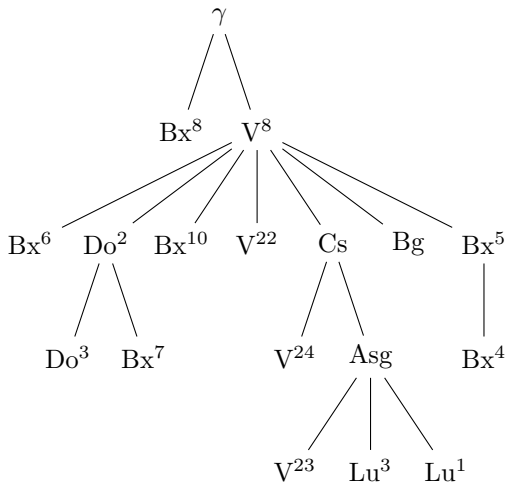

```
\begin{tikzpicture}
[level 1/.style={sibling distance=40mm},
 level 2/.style={sibling distance=20mm}]
\node{A}
  child{node{B}
    child{node{D}}
    child{node{E}}}
  child{node{C}
    child{node{F}}
    child{node{G}}}
;
\end{tikzpicture}
```

Figura 4: Albero con una radice, due figli (nodi intermedi) e quattro foglie (nodi terminali) dopo la correzione manuale degli ingombri.




```
\begin{tikzpicture}
\node {Cs}
    child { node {V$^{24}$} }
    child { node {Asg}
      child { node {V$^{23}$} }
      child { node {Lu$^{3}$} }
      child { node {Lu$^{1}$} }
    };
\end{tikzpicture}
```

Figura 5: Uno *stemma codicum* concreto.




```
\begin{tikzpicture}
[level 1/.style={sibling distance=1cm}]
\node (g) {$\gamma$}
  child { node (E) {Bx$^{8}$} }
  child { node {V$^{8}$}
    child { node {Bx$^{6}$} }
    child { node {Do$^{2}$}
      child { node {Do$^{3}$} }
      child { node (B) {Bx$^{7}$} }
    }
    child { node {Bx$^{10}$} }
    child { node {V$^{22}$} }
    child { node {Cs}
      child { node {V$^{24}$} }
      child { node {Asg}
        child { node (v23) {V$^{23}$} }
        child { node {Lu$^{3}$} }
        child { node {Lu$^{1}$} }
      }
    }
    child { node {Bg} }
    child { node {Bx$^{5}$}
      child { node {Bx$^{4}$} }
    }
  }
;
\end{tikzpicture}
```

Figura 6: Uno *stemma codicum* più profondo (cinque livelli).

ta che serve per futuri riferimenti ai nodi quando aggiungeremo degli archi, cioè dei percorsi non previsti nell'albero standard. Ovviamente, un albero che non rispetti più le regole costruttive e costitutive di tale struttura, come avere archi che uniscano nodi arbitrari, non sarà più un albero ma un grafo. Proprio di grafi ci occuperemo nel paragrafo 3 in cui introdurremo le *contaminazioni*.

Abbiamo già visto come possiamo variare la distanza tra i nodi dello stesso livello. Ti*k*Z ci permette di variare anche la distanza tra i livelli grazie all'attributo `level distance`. La sintassi è uguale a quella vista per `sibling distance`: basta specificare il valore dell'attributo come opzione all'apertura dell'ambiente tikzpicture. Anche in questo caso l'attributo è ereditario e viene propagato agli altri livelli, a meno che non specifichiamo un valore diverso per ogni livello.

Se vogliamo imporre che tra un livello e l'altro la distanza sia di 3 cm dobbiamo scrivere

```
\begin{tikzpicture}[level 1/.style={level
distance=3cm}]
```

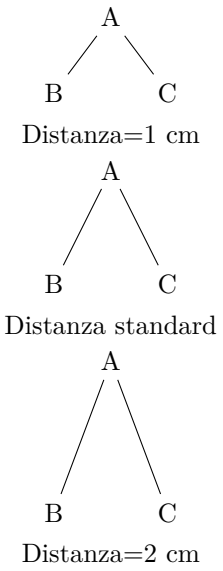


FIGURA 7: Alberi con i livelli a distanze differenti.

La figura 7 mostra lo stesso albero con diverse distanze tra i livelli.

3 Segnalazione delle contaminazioni

Abbiamo detto che Ti*k*Z ci permette di etichettare i nodi per disegnare degli archi non previsti dalla costruzione standard dell'albero. Per comodità etichetteremo solo i nodi di interesse, cosa già vista nella figura 6, e aggiungeremo il codice relativo al disegno degli archi *dopo* il ';' che chiude la costruzione dell'albero:

```
\draw[dashed,-] (B) -- (E);
\draw[dashed,-] (B) -- (g);
\draw[dashed,-] (E) -- (v23);
\draw[dashed,-] (g) -- (v23);
```

Il comando `\draw` serve a disegnare qualcosa; nel caso specifico disegna un segmento rettilineo (`--`) tra due nodi (le lettere tra parentesi). Come argomento opzionale di `\draw` specifichiamo lo stile del segmento (qui *dashed*, cioè tratteggiato) e il tipo di terminazione. In questo caso non c'è alcuna terminazione perché il segno `-` indica un segmento senza terminazioni. avremmo potuto omettere l'informazione essendo questo il caso di default. Vediamo il relativo risultato nella figura 8.

Quanto visto finora relativamente alla connessione dei nodi può essere applicato al caso di due alberi separati senza alcuna difficoltà: la figura 9 mostra proprio un caso del genere.

Questo espediente ci permetterà di collegare lo stemma concreto a un archetipo α posto alla sua sinistra. Aggiungeremo il comando

```
\node (a) at (-5,0) {$\alpha$}
;
```

prima dell'albero iniziale o dopo, o anche dopo gli archi delle contaminazioni.

Sempre a causa del fatto che Ti*k*Z non tiene conto degli ingombri, le linee tratteggiate delle contaminazioni attraversano dei nodi rendendo confusa la lettura dello *stemma codicum*. Possiamo modificare la posizione dei nodi o la curvatura degli archi in diversi modi: quello più immediato è spostare leggermente la posizione del contenuto dei nodi, magari introducendo dello spazio vuoto prima o dopo il contenuto. Spazi forzati, *box*, `\phantom` o altro sono tutti espedienti rientranti in questa categoria e utilizzabili con successo.

Non ci concentriamo però su questi metodi ma sugli strumenti propri di Ti*k*Z, un potentissimo pacchetto che fornisce gli strumenti necessari per curvare gli archi. Invece di usare un arco rettilineo (indicato con il simbolo `--` tra due punti o due etichette nel comando `\draw`), useremo un arco curvilineo indicato con il simbolo `..` e uno o due *punti di controllo*. I punti di controllo indicano le coordinate verso cui puntano le tangenti al punto iniziale e al punto finale del segmento da disegnare. Se non specifichiamo il secondo punto, Ti*k*Z lo assumerà uguale al primo. Ci troviamo quindi di fronte a una curva di Bézier quadratica (un punto di controllo) o cubica (due punti) (WIKIPEDIA, 2012; BECCARI, 2012). Separiamo i due punti di controllo con `and`. Ad esempio,

```
\draw[dashed,-] (g) .. controls (.1,-2.5)
  and (.3,-2) .. (v23);
```

Risulta chiaro che i punti di controllo servono solo a definire una curvatura; in generale la curva non passerà per essi.

Nel caso qui esposto abbiamo scelto i punti di controllo nei pressi dei nodi attraversati. Ne troviamo facilmente le coordinate tenendo conto che la radice a cui non abbiamo specificato `at` viene posta in (0,0) (origine). Le coordinate seguono la convenzione del piano cartesiano: la prima è positiva per i punti a destra dell'origine e negativa per quelli alla sua sinistra; la seconda è positiva per i punti sopra l'origine e negativa per quelli sotto di essa.

Ammettiamo che l'utenza diversa dagli scienziati (fisici, matematici, informatici, chimici), dagli ingegneri e da chi ha un buon retroterra matematico e geometrico può rimanere atterrita dalla spiegazione precedente. Può essere un incubo trovare i punti di curvatura per chi non "mastica" correntemente i numeri e le coordinate cartesiane. Ti*k*Z soccorre questi malcapitati mettendo a disposizione una griglia numerata come aiuto alla determinazione delle coordinate. Purtroppo dobbiamo disegnarla e numerarla da noi, ma è un'operazione alla portata di tutti.

In un ambiente tikzpicture disegneremo per prima cosa la griglia, grigia e con le linee fine per maggior contrasto col disegno principale e un po'

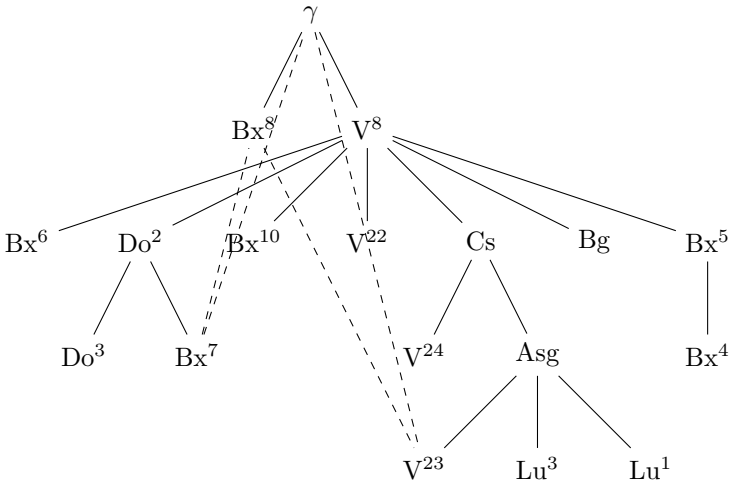
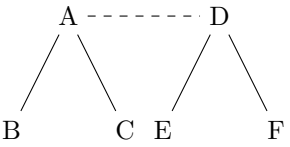


FIGURA 8: *Stemma codicum* con alcune contaminazioni segnate con archi rettilinei. Capita che questi archi si sovrappongano ad alcuni nodi.




```
\begin{tikzpicture}
\node(A){A}
  child{node{B}}
  child{node{C}}
;
\node at (2,0) (D){D}
  child{node{E}}
  child{node{F}}
;
\draw [dashed] (A) -- (D);
\end{tikzpicture}
```

FIGURA 9: Due alberi separati con le radici connesse.

più grande dello stemma così da contenerlo completamente. Il comando `\draw` accetta il punto iniziale e finale divisi dall'attributo `grid`. Opzionalmente possiamo dare il *passo* della griglia (la distanza tra due linee adiacenti), il colore e lo spessore delle linee. Questo comporterà alcune prove iniziali ma renderà più facile il lavoro; cancelleremo la griglia appena il grafo sarà definitivo. Dopo le prime volte che avremo usato questo "bastone d'appoggio" arriveremo addirittura a farne a meno per sempre. Nel nostro caso la griglia adatta è

```
\draw[step=1cm,gray,thin] (-5,-7)
 grid (6,1);
```

Possiamo rendere la griglia millimetrata anziché centimetrata aggiungendo

```
\draw[step=1mm,gray,very thin] (-5,-7)
 grid (6,1);
```

al precedente comando. Teniamo presente che possiamo anche sostituire i due attributi `gray` e `very thin` con `help lines` ottenendo un effetto simile.

A questa griglia dovremo aggiungere i numeri. Potremmo scriverli uno per uno ma, nel caso la griglia abbia molte linee, la cosa diventerebbe impraticabile. Sfruttiamo quindi il comando per i cicli di Ti*k*Z: `\foreach`. Con un ciclo scriviamo i numeri sull'asse verticale e con un altro ciclo scriviamo i numeri sull'asse orizzontale:

```
\foreach \y in {-7,...,1}
  \node at (-5.5, \y)
    {\makebox[1pc][r]{\y}}; % numeri
      % imbandierati a destra
\foreach \x in {-5,...,6}
  \node at (\x, 1.5) {\x};
```

Dopo il comando `\foreach` specifichiamo un contatore (nel nostro caso sono due: `\x` e `\y`) che assumerà di volta in volta uno dei valori possibili, elencati tra parentesi graffe. Siccome i valori possono essere molti e sarebbe impraticabile elencarli tutti, possiamo specificare solo i punti di inizio e fine separati dalle virgole e da `...`. Questo definisce un *intervallo* di valori. In questo caso semplice i valori dell'intervallo differiscono di 1: il primo contatore assumerà valori -7, -6, -5 e così via fino a 1, mentre il secondo assumerà i valori -5, -4, -3 e così via fino a 6. L'unica istruzione eseguita da ogni ciclo è il disegno di un nodo composto da una scritta: il valore del contatore tra parentesi graffe. La posizione del nodo è data dal valore del contatore e da una coordinata fissa, spostata di poco rispetto agli assi di riferimento. La figura 10 mostra il risultato della griglia con dentro lo *stemma codicum* da modificare: ora è facilissimo capire dove prendere i punti di controllo per curvare gli archi tratteggiati.

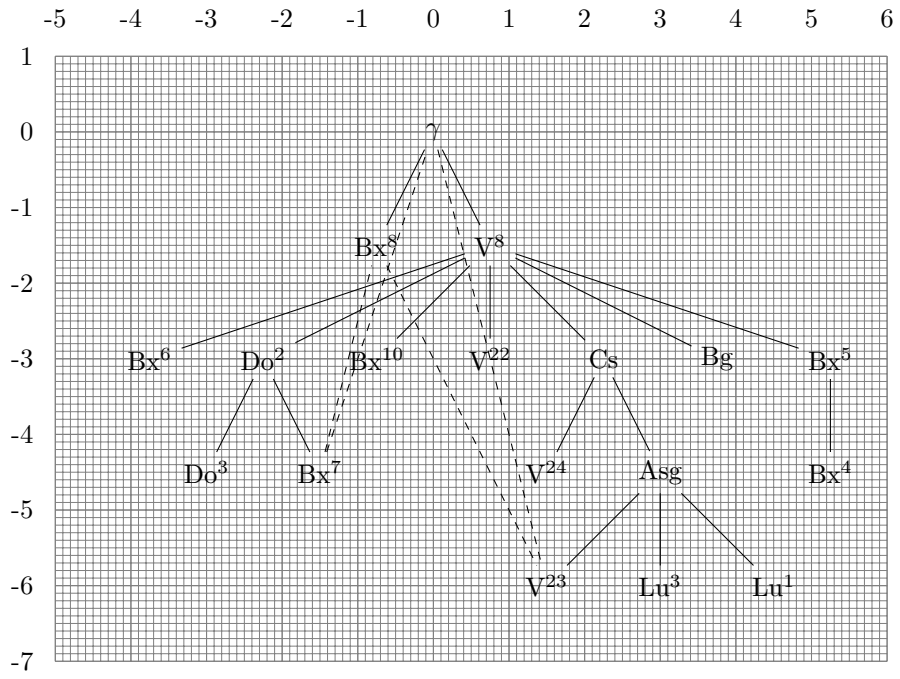


FIGURA 10: La griglia sullo sfondo dello *stemma codicum* rende più agevole la determinazione dei punti di controllo per curvare gli archi sovrapposti ai nodi.

Quello che faremo, ad esempio, è sostituire i seguenti comandi:

```
\draw[dashed,-] (B) ..
  controls (-1.2,-2.75) .. (E);
\draw[dashed,-] (B) ..
  controls (-1.25,-1) .. (g);
\draw[dashed,-] (E) -- (v23);
\draw[dashed,-] (g) ..
  controls (.1,-2.5) .. (v23);
\draw (a) -- (g);
```

a quelli usati per disegnare lo stemma della figura 8. Il risultato è visibile nella figura 11. L'ultimo `\draw` serve a unire "α" e "γ".

4 Contro xytree

Siamo arrivati alla "sfida" Ti*k*Z contro xytree, sfida limitata a tre degli esempi mostrati da Matteo Fadini in FADINI (2012), riprodotti nella figura 12 e affiancati all'equivalente versione composta con Ti*k*Z. Vediamo di seguito i sorgenti di entrambe le versioni, ricordando che le versioni xytree sono riprodotte in accordo con il loro autore e indentate diversamente in virtù dei diversi spazi.

Stemma 1

Il codice xytree è il seguente:

```
\xytree{ & \xynode[-1,0,1]{O}
 \xyconnect[.](D,D){0,2} &   &
\xynode[0]{O\ap{1}} \\
\xynode{A} & \xynode{B} &
\xynode{C} & \xynode{D}
}
```

mentre il codice Ti*k*Z è questo:

```
\begin{tikzpicture}
\node(O){O}
  child{node {A}}
  child{node {B}}
  child{node {C}}
;
\node at (3,0) (O1){O$^1$}
  child{node {D}}
;
\draw[dotted] (O) -- (O1);
\end{tikzpicture}
```

Stemma 2

Il codice xytree è il seguente:

```
\xytree{  &
\xynode[-1,0,1]{O}
 \xyconnect[.](D,D){0,2} &  &
\xynode[0]{O\ap{1}}\\
\xynode[0]{A} & \xynode[0]{B} &
\xynode{C} & \xynode[0]{D}
 \xyconnect[.](D,U){1,-2} \\
\xynode{E} & \xynode{F}
 \xyconnect[.>][^](U,D){-1,1} &   &
\xynode{G}\\
}
```

mentre il codice Ti*k*Z è questo:

```
\node(O){O}
  child{node {A}
    child{node {E}}}
  child{node {B}
    child{node (F){F}}}
```

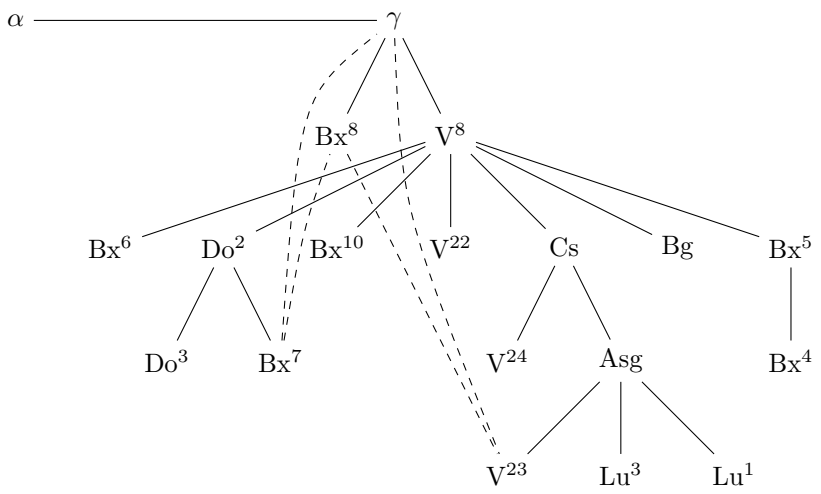


FIGURA 11: *Stemma codicum* con alcune contaminazioni segnate con archi curvilinei. In questo modo evitiamo la sovrapposizione degli archi con i nodi. Troviamo anche il collegamento tra i due capostipiti α e γ.

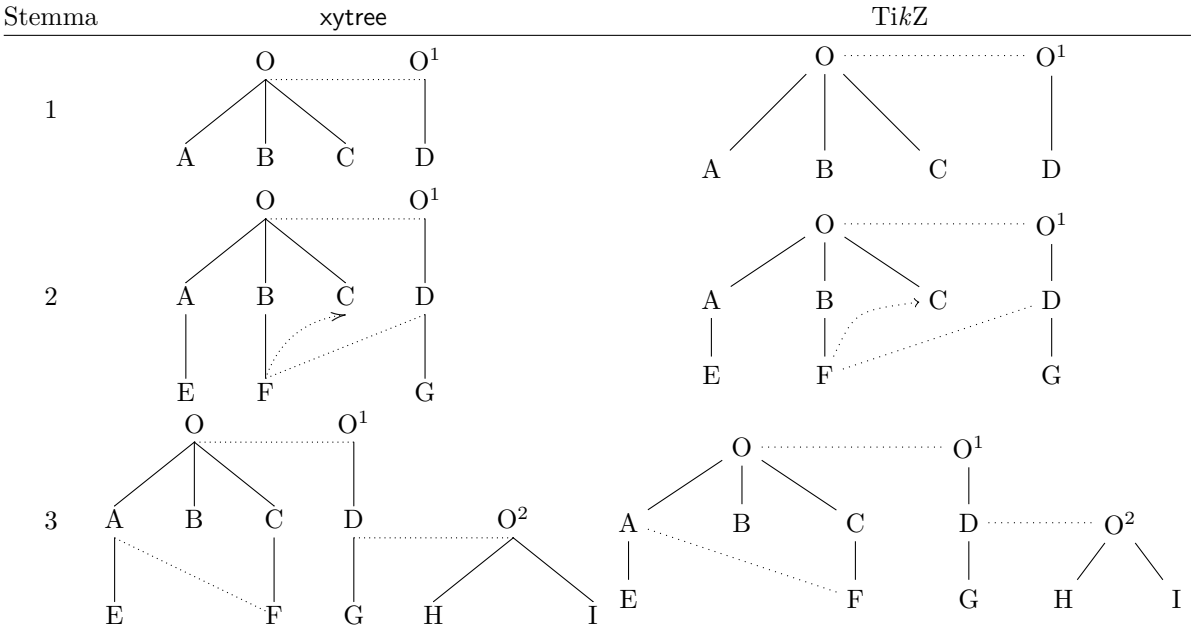


FIGURA 12: *Stemmata codicum* a confronto: a sinistra quelli realizzati con xytree e a destra quelli realizzati con Ti*k*Z. A parte le ovvie differenze estetiche (facilmente modificabili quelle di Ti*k*Z per riprodurre quelle di xytree) gli stemmi sono uguali.

```
  child{node (C){C}}
;
\node at (3,0) (O1){O$^1$}
  child{node (D){D}
    child{node {G}}}
;
\draw[dotted] (O) -- (O1);
\draw[dotted] (F) -- (D);
\draw[dotted,->] (F) ..
 controls +(.3,1.05) .. (C);
\end{tikzpicture}
```

Notiamo che l'arco tra i nodi "C" e "F" è curvo e terminato da una freccia, come stabilito nell'argomento opzionale dal comando `->`.

Stemma 3

Il codice xytree è il seguente:

```
\xytree{ &
\xyconnect[.](D,D){0,2}
 \xynode[-1,0,1]{O} &%
&
\xynode[0]{O\ap{1}} & & & \\
\xyconnect[.](D,L){1,2} \xynode[0]{A} &
\xynode{B} & \xynode[0]{C} &
\xynode[0]{D} \xyconnect[.](D,D){0,2} & &
\xynode[-1,1]{O\ap{2}} &\\
\xynode{E}&  &
\xynode{F} & \xynode{G} & \xynode{H} &  &
\xynode{I} \\
}
```

mentre il codice Ti*k*Z è questo:

```
\begin{tikzpicture}
\node(O){O}
  child{node (A){A}
    child{node {E}}}
  child{node {B}}
  child{node {C}
    child{node (F){F}}}
;
\node at (3,0) (O1){O$^1$}
  child{node (D){D}
    child{node {G}}}
;
\node at (5,-1.5) (O2){O$^2$}
  child{node {H}}
  child{node {I}}
;
\draw[dotted] (O) -- (O1);
\draw[dotted] (A) -- (F);
\draw[dotted] (D) -- (O2);
\end{tikzpicture}
```

Dai codici appena visti è evidente quanto detto nell'introduzione: disegnare uno *stemma codicum* con xytree è veramente come comporre una tabella con un nodo per cella, connessioni a parte. Invece, disegnarlo con Ti*k*Z è più simile a comporre un albero con synttree poiché Ti*k*Z mantiene la sintassi di un albero (con in più la complessità e e possibilità del suo linguaggio) a cui vanno poi aggiunte le eventuali connessioni supplementari.

Quale dei due metodi sia migliore, però, dipende da ogni autore e dal suo stile di ragionamento, di organizzazione spaziale e di preferenza visiva. Ognuno può vedere i risultati a confronto (codici e disegni) e decidere per sé quale metodo usare caso per caso o definitivamente.

5 Conclusioni

Gli utenti LaTeX hanno a disposizione diversi modi per comporre uno *stemma codicum*. Il più semplice di essi, il pacchetto synttree, permette la generazione di un albero a partire da un codice di descrizione dell'albero stesso (organizzazione logica). Tranne i casi più semplici, uno *stemma codicum* concreto sarà ben più complesso di un albero: dovrà mostrare tutte le contaminazioni sia tra diversi archetipi che tra documenti successivi. Ciò trasforma un albero in un grafo. Purtroppo synttree non è in grado di gestire questa complessità perché serve a comporre degli alberi mentre xytree è più adatto in casi del genere. Tuttavia la sua sintassi è più simile a quella di una tabella che non a quella di un grafo (organizzazione fisica). A tale proposito Ti*k*Z offre una soluzione più adatta poiché la sintassi dei suoi comandi per generare alberi e grafi è più simile a quella di synttree, con l'ulteriore possibilità di personalizzare l'aspetto finale del disegno.

L'articolo ha mostrato come comporre degli *stemmata codicum* con Ti*k*Z, ha esposto come sia possibile connettere più alberi e come segnalare le contaminazioni etichettando i nodi dell'albero e connettendoli *ad hoc*. Ha inoltre spiegato come aggiustare le distanze e come deformare gli archi per evitare sovrapposizioni. Infine ha descritto come replicare alcuni risultati permessi dal pacchetto xytree. Tutto ciò ha trovato e trova tutt'ora riscontro in casi concreti quali gli *stemmata codicum* della tesi di dottorato di Fabrizio Tiddia.

Riferimenti bibliografici

Beccari, C. (a cura di) (2012). *Introduzione all'arte della composizione tipografica con LaTeX*. Versione A4-0.98t.

Fadini, M. (2012). «*Stemma codicum* con LaTeX». *ArsTEXnica*, (14).

Leal, J. e Pignalberi, G. (2012). *Edizioni Critiche. Guida alla composizione con il proprio computer*. TEXnologie. CompoMat, Rieti, Italia.

Pignalberi, G. e Bini, E. (2008). «LaTeX e grammatiche (la faccia triste dell'informatica)». *ArsTEXnica*, (6), pp. 76–85. URL `http://www.guitex.org/home/images/ArsTeXnica/AT006/LATEX%20e%20grammatiche.pdf`.

Tantau, T. (2010). *The Ti*k*Z and pgf Packages*. URL `http://www.ctan.org`. Lo leggete dando `texdoc tikz` al prompt del terminale o di DOS.

Un, K. (2006). *xytree.sty — A LaTeX package for drawing syntsctic trees*. URL `http://www.ctan.org`. Lo leggete dando `texdoc xytree` al prompt del terminale o di DOS.

van Zuijlen, M. (2009). *The synttree package for typesetting syntactic trees*. URL `http://www.ctan.org`. Lo leggete dando `texdoc synttree` al prompt del terminale o di DOS.

Wikipedia (2012). «Curva di Bézier». URL `http://it.wikipedia.org/wiki/Curva_di_Bézier`.

▷ Jerónimo Leal
Dipartimento di Storia della Chiesa
Pontificia Università
della Santa Croce
Piazza Sant'Apollinare 49
00186 Roma
`jleal at pusc dot it`

▷ Gianluca Pignalberi
`gianluca dot pignalberi at alice dot it`