Zeichnen von Bäumen

Lina Wolf

1. Was ist ein Baum?

Ein Baum ist ein zusammenhängender, azyklischer Graph. In der Regel hat ein Baum T einen Knoten $r \in T$, der Wurzel genannt wird.
Man geht davon aus, das alle Kanten von der Wurzel weg gerichtet sind. Die Wurzel ist der einzige Knoten ohne ankommende Kanten. Es gibt von der Wurzel zu jedem Knoten genau einen Weg.
Für eine gerichtete Kante (u,v) heißt u Mutter und v Kind. Kein Knoten kann mehr als eine Mutter haben.
Ein Knoten heißt Blatt, wenn er keine Kinder hat.
Die Tiefe eines Knotens ist die Anzahl der Kanten, um ihn von der Wurzel aus zu erreichen.
In einem m-ären Baum hat jeder Knoten maximal m Kinder.
In einem geordneten Baum ist die Ordnung der Kinder eines Knotens entscheidend.
In einem geordneten Binärbaum ist jedes Kind eines Knotens entweder ein rechtes oder linkes Kind. Das heißt, wenn ein Knoten nur ein Kind hat, muss dieses nach rechts oder links gezeichnet werden(Man denke an binäre Suchbäume!)
In einem m-ären Baum wird ein einziges Kind direkt unter die Mutter gezeichnet.
Ein Teilbaum mit Wurzel v ist der Teilgraph mit allen Knoten, die auf gerichteten Wegen von v aus erreicht werden können.
Ein freier Baum (ohne spezifizierte Wurzel) kann wie ein gewurzelter Baum gezeichnet werden, indem man eine Wurzel wählt. Ideal ist es die Wurzel so zu wählen, dass die Höhe des Baumes minimal wird:
hat der Baum 1 oder 2 Knoten, so ist das die Mitte.
sonst entferne alle Blätter und wiederhole.

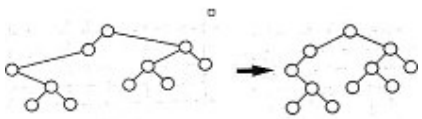
2. Generelle Wünsche

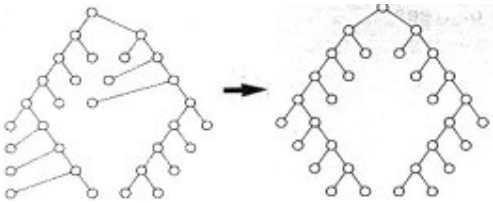
- Planare Zeichnungen: keine zwei Kanten kreuzen
- Gitter Zeichnungen: Punkte haben integer Koordinaten
- Gerade- Linien- Zeichnungen: jede Kante ist eine Gerade
- (streng) aufsteigende Zeichnungen: ein Kind muss (streng) unterhalb der Mutter plaziert sein
- Ordnungerhaltende Zeichnungen: die Kante von der Mutter zum linkesten (rechtesten) Kind muss monoton fallend(steigend) sein. Die Kanten aller Kinder sind nach Winkel von links nach rechts sortiert.
- Aufgeräumte Zeichnung(tidy drawing): möglichst wenig Platztverbrauch bei ästhetischem Endergebnis.

3. Ästhetische Regeln (*nach Tilford und Reingold*)

1. Knoten gleicher Tiefe sollen auf gerade Linie liegen. Diese Linien sollen parallel sein.
2. Ein linkes Kind soll links von seiner Mutter liegen und ein rechtes Kind rechts. Dieser Wunsch gilt natürlich nur in Binärbäumen

3. Die Mutter soll über ihren Kindern zentriert werden.





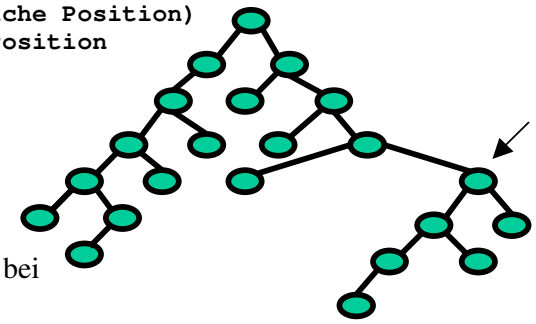
4. Ein Baum und sein Spiegelbild sollen gleiche Zeichnungen, die nur gespiegelt sind, erzeugen (Isomorphismus). Ein Teilbaum soll immer gleich gezeichnet werden, egal wo er im Baum erscheint.

4. Geschichtete Zeichnungen (Layered Drawings)
Als Y- Koordinate des Knotens wird seine Tiefe verwendet. Das garantiert die 1. ästhetische Regel (Knoten selber Tiefe auf gerader Linie).
Daher müssen sich Algorithmen nur noch um die Positionierung auf der X-Achse kümmern.
Alle im folgenden vorgestellten Algorithmen für geschichtete Zeichnungen laufen nach dem selben Schema ab, das hier für alle erklärt wird:
In einem Schritt, den ich firstWalk() nenne, werden allen Knoten vorläufige Koordinaten zugewiesen. War es in diesem Zuge notwendig einen Teilbaum zu verschieben, so wurde diese Verschiebung nur in seiner Wurzel gespeichert.
In einem zweiten Durchgang wird der Baum pre-order traversiert, dabei werden die aufsummierten Verschiebungen nach unten durchgereicht, so dass jeder Knoten um die Summe der Verschiebungen aller seiner Vorgänger verschoben wird:

```
void secondWalk(Node knoten, int tiefe, int summe){
      knoten.x_Koordinate = knoten.x_Koordinate + summe;
      knoten.y_Koordinate = tiefe;
      für(alle Kinder){
            secondWalk(kind, tiefe+1, summe+knoten.verschiebung);
      }
}
```

4.1 Algorithmus WS
Der Algorithmus von Wetherell und Shannon funktioniert nur mit Binärbäumen. Er beachtet nicht die 4. ästhetische Regel (Isomorphismus)
Der Algorithmus produziert zum Teil Ergebnisse, deren Platzanforderung nach den 3 beachteten ästhetischen Regeln nicht minimal ist (Bsp siehe unten).

```
firstWalk{
      für (alle Knoten in post-Order){
            Falls (Knoten hat n Kinder){
                  n == 0: Knoten.x <- nächst mögliche Position
                  n == 1: Knoten.x <- 1 Einheit rechts/links von Kind
                  n == 2: zentriere Knoten über Kindern
            }
            wenn (Knoten.x < nächst mögliche Position)
            Knoten.x <- nächst mögliche Position
            Teilbäume geschifftet
      }
}
```



Wäre der Knoten, auf den der Pfeil zeigt, weiter links, wäre der Platzbedarf des Baumes kleiner bei weiterer Beachtung aller Regeln.

4.1.1 verbesserter WS-Algorithmus
Wetherell und Shannon schlagen einen neuen Algorithmus vor, der das Problem des Platzbedarfs verbessern soll. Dabei geben sie allerdings die 3. ästhetische Regel(Mutter zentriert über Kindern) auf, weshalb dieser Algorithmus nicht direkt eine Verbesserung des

gesammten Problems ist. Man möchte aufgeräumte Zeichnungen (minimaler Platzbedarf bei Beachtung der ästhetischen Regeln) haben.

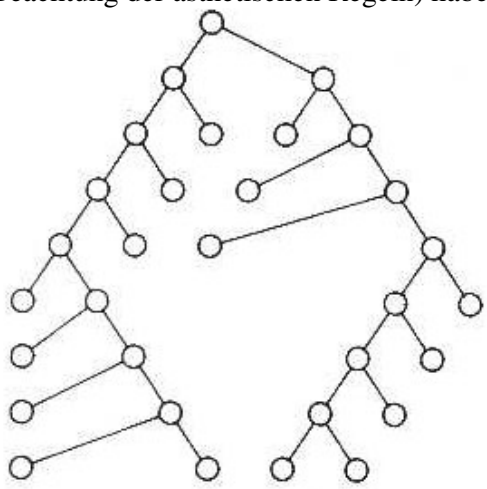


Fig. 3. Example tree drawn by a modified Algorithm WS.

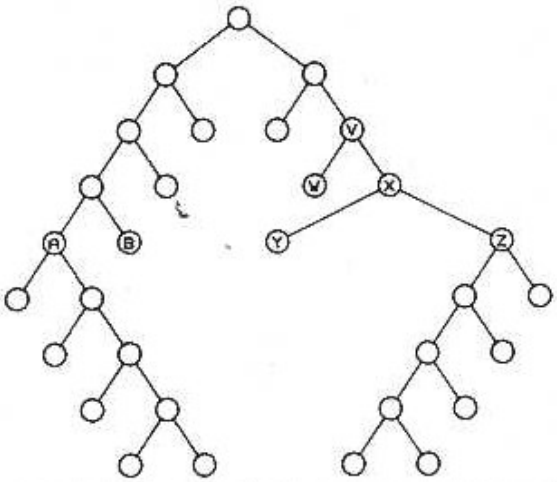


Fig. 1. Final positioning of example tree as drawn by Algorithm WS.

4.2 Algorithmus TR

Reingold und Tilford führen die 4. ästhetische Regel ein. Walker zeigt, dass nicht alle Bäume, die der TR-Algorithmus produziert isomorph sind, aber Teilbäume werden gleich gezeichnet unabhängig von Knoten außerhalb des Teilbaumes.

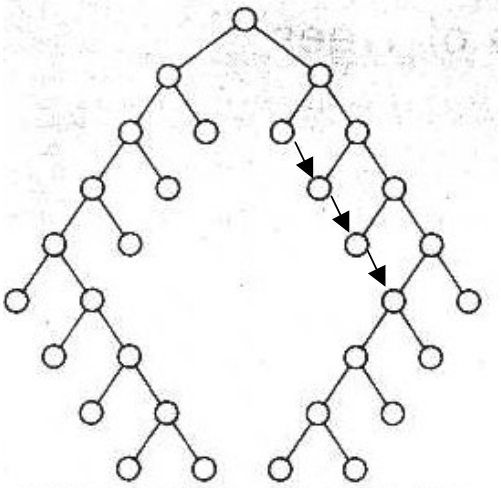
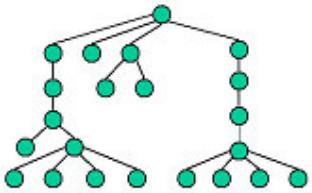


Fig. 2. Example tree as drawn by Algorithm TR.

Im firstWalk wird der Baum in post-order traversiert. Dabei werden jeweils die beiden Teilbäume rekursiv erzeugt und dann so nah wie möglich beieinander plaziert. Wie man im Beispiel sehen kann, reicht es nicht aus einfach der rechten(linken) Kante zu folgen.

Entweder es gibt eine Kante, der man folgen kann, oder der Knoten ist ein Blatt. Wenn der Knoten ein Blatt ist, hat er unbenutzte Pointer, diese werden verwendet, um auf den nächsten Knoten zu zeigen (Pfeile).

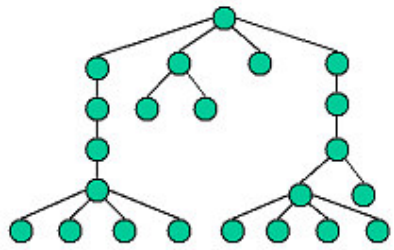
4.2.1 Erweiterung des TR-Algorithmus für m-äre Bäume



Der TR-Algorithmus lässt sich relativ leicht für m-äre Bäume erweitern. Dann tritt jedoch das sogenannte links-rechts-kleben Problem (left-right-gluing problem) auf. Wenn man ein neuer, größerer Teilbaum immer weiter nach rechts geschoben werden muss, öffnet sich eine Lücke zwischen ihm und zuvor korrekt eingefügten kleineren Teilbäumen. Diese scheinen sich dann allerdings nach links zu häufen. Auch die 4. Regel, Isomorphismus, ist jetzt verletzt.

4.3 Algorithmus von Walker

Walker stellt einen verbesserten Algorithmus zum geschichteten Zeichnen von m-ären Bäumen nach den 4 Ästhetischen Regeln vor (ohne Regel 2 linkes/ rechtes Kind).



Laut Walker (!) beträgt die Laufzeit O(n), aber es wurde nachgewiesen, dass die Laufzeit in schlechten Fällen $\Omega(n^{3/2})$ wird.

Wenn bei einem Binärbaum ein Knoten nur ein Kind hat, so wird dieses direkt unter seiner Mutter plaziert(kein linkes/rechtes Kind).

```
firstWalk(knoten, level){
      knoten.x_Koordinate = 0;
      wenn(knoten.hatLinkeSchwester){
            knoten.x_Koordinate = linkeSchwester.x_Koordinate +1;
      }
      wenn(!knoten.istBlatt){
            für(alleKinder){
                  firstWalk(kind, level+1);
            }
            mitte =(linkesKind.xKoordinate +
                  rechtesKind.x_Koordinate)/2;
            wenn(knoten.hatLinkeSchwester){
                  knoten.verschiebung = knoten.x_Koordinate + mitte;
                  apportion(knoten, level);
            } sonst {
                  knoten.x_Koordinate = mitte;
      }
}
```

Apportion errechnet den Abstand zwischen dem rechtesten und linkesten Teilbaum und verteilt diesen gleichmäßig auf alle Schwestern. Dadurch wird links-rechts-kleben verhindert

```
void apportion(knoten, level){
      linkestes = knoten.erstesKind();
      nachbar = linkerNachbar(linkestes);
      tiefe = 1;
      solange(linkestes != null && nachbar != null){
            rechtePosition = linkestes.position(tiefe);
            linkePosition = nachbar.position(tiefe);
            vorfahrNachbar = nachbar.Vorfahr(tiefe);
            abstand = rechtePosition + TEILB_TRENNUNG - linkePosition;
            wenn(abstand > 0){
                  linkeSchw = knoten.zählLinkeSchwestern(vorfahrNachbar);
                  wenn(knoten.hatLinkeSchwester(vorfahrNachbar)){
                        teil = abstand/ linkeSchw;
                        für(alle Schwestern von knoten bis
                                    vorfahrNachbar){
                              schwester.x += abstand;
                              schwester.verschiebung += abstand
                              abstand = abstand-teil;
                        }
                  }
            }
            wenn(linkestes.istBlatt()){
                  linkestes = knoten.holLinkestes(knoten, 0, tiefe);
            } sonst {
                  linkestes = linkestes.erstesKind();
            }
```

```
        }
}
```

4.3.1 verbesserter Walker

Buchheim, Jünger und Leipert weisen nach, dass Walker in worst-case $\Omega(n^{3/2})$ ist. Sie schlagen einen anderen Algorithmus vor, der mit einem veränderten apportion angeblich in linearer Zeit arbeitet.

Verbesserungen:

-den Konturen folgen und Verschiebungen aufsummieren:

es wird das Verfahren von Reingold und Tilford mit den Pointern von den Blättern verbessern

-Vorfahren finden:

der rechte Vorfahr ist sowieso bekannt, der linke wird mit dem Algorithmus von Schieber und Vishkin gefunden.

-Teilbäume zählen:

einfach nr linker - nr rechter +1

verschieben kleiner Teilbäume:

es werden alle Teilbäume in einer Traversierung verschoben. inzwischen werden nur die gewünschten Verschiebungen gespeichert.

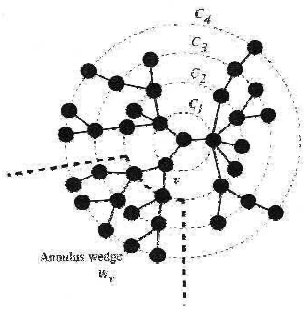
5. Andere Algorithmen

5.1 Radiale Zeichnungen

Radiale Zeichnungen werden gerne für freie Bäume verwendet.

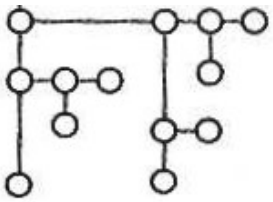
Sie werden mit ähnlichen Algorithmen wie geschichtete Zeichnungen gezeichnet. Die (errechnete) Wurzel wird im Mittelpunkt plaziert, die anderen Knoten auf Kreisbahnen.

Ein Teilbaum darf dabei allerdings nicht in einem beliebig großen Kreisabschnitt gezeichnet werden, sonst können sich seine Kanten mit denen von Nachbarbäumen schneiden. Teilbäume müssen auf konvexe Kreisabschnitte beschränkt bleiben.



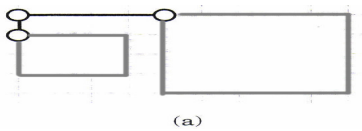
5.2 HV-Zeichnungen

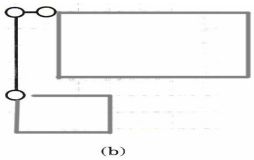
HV(= horizontal -vertikal)-Zeichnungen sind nur für Binärbäume geeignet.



Divide: Rekursiv linken und rechten Teilbaum konstruieren

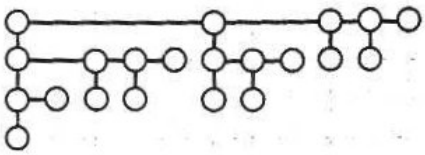
Conquer: horizontale (a) oder vertikale (b) Kombination





5.2.1 rechtsschwere HV- Zeichnungen

Ordnung der Kinder bleibt nicht erhalten die Höhe der Zeichnung ist maximal log(n) bei n Knoten des Baumes



Divide: Rekursiv Teilbäume konstruieren

Conquer: mit horizontaler Kombination größerer Teilbaum rechts von anderem