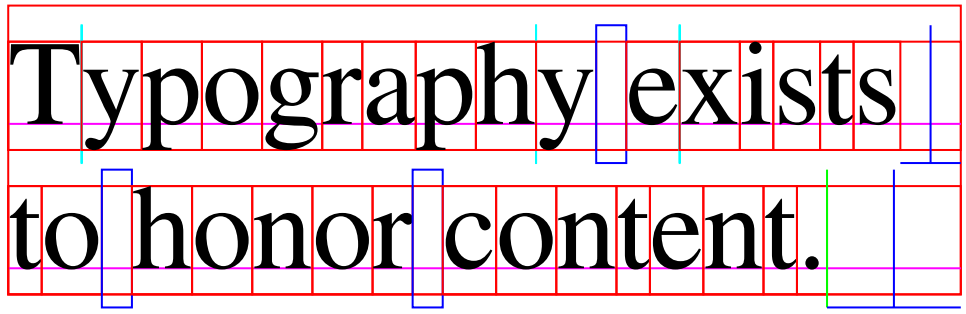
Schnittstelle für zeilenbrechende Algorithmen 346

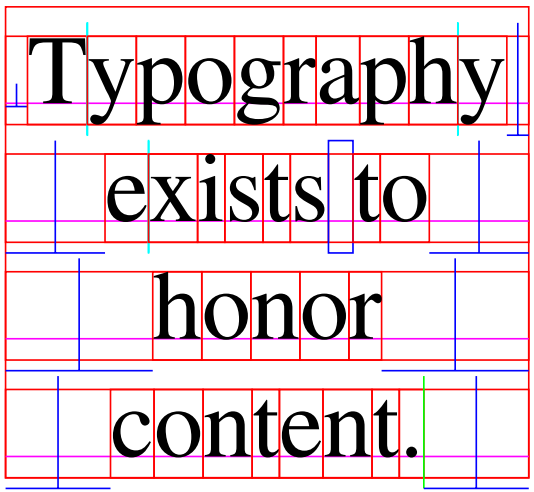
LineBreaker.java

```
public interface LineBreaker {
   public void setLineWrapper(SequenceWrapper wrapper);
   public void setHorizontalBoxWrapper(ItemWrapper hboxwrapper);
   public Item breakParagraph(HorizontalSequence hseq,
      int parwidth, int baselineskip);
} // interface LineBreaker
```

- Da es mehrere zeilenbrechende Algorithmen gibt, ist eine allgemeine Schnittstelle hierfür sinnvoll.
- Der eigentliche Algorithmus wird verpackt durch die Methode *breakParagraph()*, die eine horizontale Sequenz in Zeilen bricht, die einzelnen Zeilen an die vorgegebene Weite anpasst und diese dann mit dem gegebenen Zeilenabstand in einer vertikalen Box übereinander stapelt.
- Optional kann mit *setLineWrapper()* eine Bearbeitung einer Zeilensequenz vor der Anpassung an die vorgegebene Weite eingebaut werden. Ferner ist mit *setHorizontalBoxWrapper()* eine Nachbearbeitung der horizontalen Schachteln für die einzelnen Zeilen möglich.



- Mit *setLineWrapper()* ist es möglich, vom traditionellen Blocksatz abzuweichen.
- Wenn der Wrapper an die horizontalen Sequenzen jeweils eine Dehnfuge mit unendlicher Dehnkapazität anfügt, erhalten wir einen aufgerauhten rechten Rand.
- Der grüne Strich repräsentiert hier eine Sollbruchstelle, die an das Ende des Paragraphen angehängt wurde. Auf diese Weise wird auch das letzte Wort mit einer Trennstelle terminiert, wenn zu Beginn die sonst übliche unendlich dehnbare Dehnfuge am Ende des Paragraphen fehlt.



- Wenn durch den Wrapper für die einzelnen Zeilen jeweils am Anfang und Ende jeweils eine unendlich dehnbare Dehnfuge hinzugefügt wird, erhalten wir eine zentrierte Formatierung.

Basisklasse für zeilenbrechende Algorithmen 349

BasicLineBreaker.java

```
public abstract class BasicLineBreaker implements LineBreaker {
   protected SequenceWrapper wrapper;
   protected ItemWrapper hboxwrapper;

   public BasicLineBreaker() {
      wrapper = null; hboxwrapper = null;
   }

   public void setLineWrapper(SequenceWrapper wrapper) {
      this.wrapper = wrapper;
   }

   public void setHorizontalBoxWrapper(ItemWrapper hboxwrapper) {
      this.hboxwrapper = hboxwrapper;
   }

   // ...

   public abstract Item breakParagraph(HorizontalSequence hseq,
      int parwidth, int baselineskip);
} // class BasicLineBreaker
```

BasicLineBreaker.java

```
protected void add(VerticalBox vbox,
      HorizontalSequence line, int parwidth) {
   if (wrapper != null) {
      line = wrapper.wrap(line);
   }
   HorizontalFitter.fit(line, parwidth);
   Item hbox = new HorizontalBox(line);
   if (hboxwrapper != null) {
      hbox = hboxwrapper.wrap(hbox);
   }
   vbox.add(hbox);
}
```

- Die *add*-Methode wird von den abgeleiteten Klassen aufgerufen, um eine fertiggestellte *HorizontalSequence* an eine *VerticalBox* anzuhängen.
- An dieser Stelle werden die beiden Wrapper aufgerufen: *wrapper.wrap* passt ggf. die Formatierung an (Blocksatz, aufgerauht, zentriert etc) und der *hboxwrapper* dient nur u.U. der Visualisierung des Schachtelsystems.

- Gegeben ist eine horizontale Sequenz, die in einzelne Wörter zerlegt ist.
- Für den Algorithmus wird zusätzlich eine horizontale Sequenz verwaltet, die für die aktuelle Zeile steht. Diese ist zu Beginn leer.
- Für jedes Wort aus der horizontalen Sequenz wird überprüft, ob sich dieses noch auf die aktuelle Zeile quetschen lässt, d.h. ob der bereits vorhandene Zeileninhalt und die dazwischenliegende Trennstelle und das neue Wort bei einer maximalen Stauchung noch in eine Zeile passt.
- Wenn das neue Wort hineinpasst, dann wird das Wort in die aktuelle Zeile mitsamt der davorliegenden Trennstelle hinzugefügt. Die Schleife wird danach fortgesetzt.
- Wenn es nicht hineinpasst, wird die bislang vollendete Zeile zu den fertiggestellten Zeilen hinzugefügt (in eine vertikale Schachtel) und die aktuelle Zeile mit dem neuen Wort initialisiert. Danach geht es in der Schleife weiter.
- Am Ende der Schleife wird die aktuelle Zeile noch hinzugefügt, falls sie nicht leer ist.

FirstFitLineBreaker.java

```
public class FirstFitLineBreaker extends BasicLineBreaker {
   public Item breakParagraph(HorizontalSequence hseq,
         int parwidth, int baselineskip) {
      VerticalBox vbox = new VerticalBox(baselineskip);

      HorizontalSequence line = null; // current line
      Item bp = null; // last breakpoint
      // ... for loop over all words of hseq ...
      if (line != null) {
         line.add(bp);
         add(vbox, line, parwidth);
      }
      return vbox;
   }
} // class FirstFitLineBreaker
```

- Zu Beginn wird eine neue *VerticalBox* erzeugt, dann werden in der **for**-Schleife dieser sukzessive Zeilen hinzugefügt (siehe folgende Folie) und schließlich der Rest, sofern vorhanden, angehängt.

FirstFitLineBreaker.java

```
for (HorizontalSequence word: hseq.getWords()) {
   if (line == null) {
      line = word;
   } else {
      Width sum = new Width(line.getWidth());
      sum.add(bp); sum.add(word.getWidth());
      if (sum.getWidth() > parwidth) {
         if (sum.getWidth() - sum.getShrinkability()
               <= parwidth) {
            line.add(bp); line.add(word);
            add(vbox, line, parwidth); line = null;
         } else {
            add(vbox, line, parwidth); line = word;
         }
      } else {
         line.add(bp); line.add(word);
      }
   }
   bp = word.getFollowingBreakpoint();
}
```

- Der Best-Fit-Algorithmus bemüht sich um lokale Minima bezüglich der Häßlichkeit.
- Die Häßlichkeit wird danach bewertet, wie dramatisch der zur Verfügung stehende Spielraum ausgenutzt wird.
- Ein Wert von 0 bedeutet, dass nichts verändert werden muss. Wenn ein Wert von 1 erreicht wird, dann wird der Spielraum vollständig ausgenutzt. Wenn der Wert von 1 überschritten wird, dann überstrapazieren wir die vorhandene Flexibilität.
- Der Best-Fit-Algorithmus vergleicht alle Trennungskandidaten am nächsten Zeilenende und vergleicht sie anhand des Maßes.
- Best-Fit liefert nicht unbedingt bessere Resultate als First-Fit, da möglicherweise das globale Minimum erst dann erreicht wird, wenn zunächst eine etwas häßlichere Variante gewählt wird.

BestFitLineBreaker.java

```
public class BestFitLineBreaker extends BasicLineBreaker {
   public Item breakParagraph(HorizontalSequence hseq,
         int parwidth, int baselineskip) {
      VerticalBox vbox = new VerticalBox(baselineskip);

      HorizontalSequence line = null; // current line
      HorizontalSequence candidate = null;
      HorizontalSequence nextline = null; // beginning of next line
      double badness = 0; // of candidate
      Item bp = null; // last breakpoint
      // ... for loop over all words ...
      if (line != null) {
         line.add(bp);
         add(vbox, line, parwidth);
      }
      return vbox;
   }
} // class BestFitLineBreaker
```

BestFitLineBreaker.java

```
    for (HorizontalSequence word: hseq.getWords()) {
       if (line == null) {
          line = word;
       } else {
          line.add(bp); line.add(word);
       }
       if (candidate != null) {
          if (nextline == null) {
             nextline = word;
          } else {
             nextline.add(bp); nextline.add(word);
          }
       }
       Width width = line.getWidth();
       // ... analysis ...
       bp = word.getFollowingBreakpoint();
    }
```

- Zu dem aktuellen Kandidaten *candidate* gehört, falls definiert, auch *nextline* (was gehört in die nächste Zeile) und *badness* (gemessener Wert).

BestFitLineBreaker.java

```
if (width.getWidth() - width.getShrinkability() > parwidth) {
   if (candidate == null) {
      add(vbox, line, parwidth); line = null;
   } else {
      add(vbox, candidate, parwidth);
      line = nextline; candidate = null; nextline = null;
   }
} else if (width.getWidth() + width.getStretchability() >= parwidth) {
   double newbadness;
   if (width.getWidth() < parwidth) {
      newbadness = ((double) parwidth - width.getWidth()) /
         width.getStretchability();
   } else {
      newbadness = ((double) width.getWidth() - parwidth) /
         width.getShrinkability();
   }
   if (candidate == null || newbadness < badness) {
      candidate = line.clone(); nextline = null; badness = newbadness;
   }
} else {
   candidate = line.clone(); nextline = null;
   badness = Item.INFINITY;
}
```

Überblick der Struktur einer horizontalen Sequenz nach Knuth

- Eine horizontale Sequenz aus m Elementen kann durch die folgenden sechs Folgen repräsentiert werden:

$t_1 \ldots t_m$	Typ des Elements: *box*, *glue* oder *penalty*.
$w_1 \ldots w_m$	Die Weite des Elements.
$y_1 \ldots y_m$	Der Ausdehnungsspielraum bei Dehnfugen (*glue*), bei anderen Elementen 0.
$z_1 \ldots z_m$	Der Schrumpfungsspielraum bei Dehnfugen (*glue*), bei anderen Elementen 0.
$p_1 \ldots p_m$	Der Strafwert bei Sollbruchstellen, bei anderen Elementen 0.
$f_1 \ldots f_m$	Der Schalter bei Sollbruchstellen, bei anderen Elementen 0.

- Gegeben ist eine Folge $\{l_i\}$ von gewünschten Zeilenlängen. Normalerweise gilt $l_1 = l_2 = \ldots$, aber im Falle von Illustrationen oder anderen ungewöhnlichen Randbedingungen können diese voneinander abweichen.
- Wenn für die Zeile j die Elemente $a_j \ldots b_j$ in Betracht gezogen werden, dann ergeben sich aufsummiert folgende Werte:

$$
\begin{aligned}
L_j &= \sum_{i=a_j}^{b_{j-1}} w_i + \begin{cases} w_{b_j} & \text{falls } t_{b_j} = \mathit{penalty} \\ 0 & \text{sonst} \end{cases} \\
Y_j &= \sum_{i=a_j}^{b_{j-1}} y_i \\
Z_j &= \sum_{i=a_j}^{b_{j-1}} z_i
\end{aligned}
$$

- In Abhängigkeit von L_j und l_j lässt sich ein Justierverhältnis r_j bestimmen:
  - Falls $L_j = l_j$, dann passt die Zeile ganz genau und es sei $r_j = 0$
  - Falls $L_j < l_j$, dann ist die Zeile zu kurz und es sei $r_j = \frac{l_j - L_j}{Y_j}$, falls $Y_j > 0$, und undefiniert andernfalls.
  - Falls $L_j > l_j$, dann hat die Zeile Überlänge und es sei $r_j = \frac{l_j - L_j}{Z_j}$, falls $Z_j > 0$, und undefiniert andernfalls.
- Beispiel: Falls $r_j = \frac{1}{2}$, dann muss der Ausdehnungsspielraum zur Hälfte ausgenutzt werden.
- Die einzelnen Weiten aller Dehnfugen innerhalb der Zeile sind dann auf $w_i + r_j y_i$ zu setzen, falls $r_j \geq 0$ und $w_i + r_j z_i$, falls $r_j < 0$.

- Bereits 1963 schlug C. J. Duncan einen Algorithmus vor, der eine Zerlegung akzeptierte, sobald $|r_j| \leq 1$ galt. In diesem Falle haben wir eine Lösung, die im Rahmen der vorgesehenen Spielräume für Ausdehnungen und Schrumpfungen liegt.
- Da es jedoch häufig mehrere solcher Lösungen gibt, lohnt es sich, diese zu vergleichen. Auch falls es keine Lösung gibt, ist es wohl im Notfall sinnvoll, die am wenigsten schlechte Lösung auszusuchen.
- Knuth hat für TEX folgendes Maß als für sinnvoll befunden, das die Unschönheit einer Zeile bewertet:

$$\beta_j = \begin{cases} \infty, & \text{falls } r_j \text{ undefiniert oder } r_j < -1 \\ \lfloor 100|r_j|^3 + .5 \rfloor & \text{sonst} \end{cases}$$

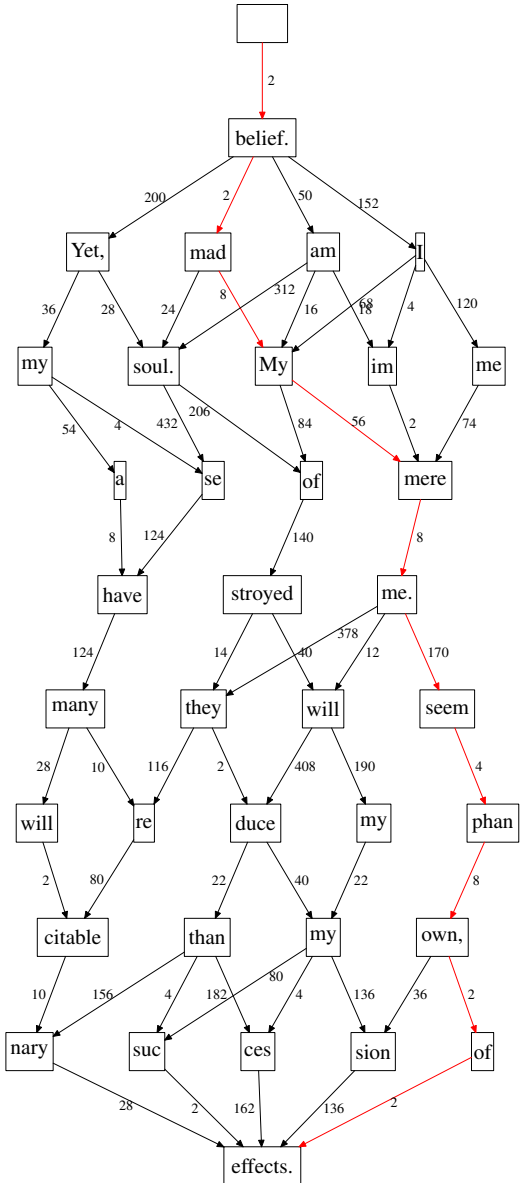
- Bei einer Aggregierung der Unschönheiten der einzelnen Zeilen geht der Strafwert der Sollbruchstelle ein: $\pi_j = p_{b_j}$
- Ferner sei α_j positiv (z.B. 3000), falls die j-te Zeile von zwei Sollbruchstellen eingefasst ist, bei der die Schalter jeweils auf 1 stehen. (Aufeinanderfolgende Trennungen von Wörtern sind zu vermeiden.)
- Darauf aufbauend definiert Knuth folgendes Maß für die Unschönheit einer Zeile:

$$\delta_j = \begin{cases} (1+\beta_j+\pi_j)^2+\alpha_j & \text{falls } \pi_j \geq 0 \\ (1+\beta_j)^2-\pi_j^2+\alpha_j & \text{falls } -\infty < \pi_j < 0 \\ (1+\beta_j)^2+\alpha_j & \text{falls } \pi_j = -\infty \end{cases}$$

- Das Maß für die Unschönheit einer Zerlegung ergibt sich dann aus der Summe der δ_j.

- Knuth versuchte, mit seinem Maß folgende Ziele umzusetzen:
  - Wenn eine einzelne Zeile unvermeidlich schlecht ist, sollen die anderen Zeilen dennoch möglichst schön gehalten werden.
  - Die Unschönheit wächst mit den Strafwerten der Sollbruchstellen π_j. Bei negativen Strafwerten wird ein Bruch an der Stelle gerne gesehen. Allerdings wird bei $\pi_j = -\infty$ dies ignoriert, da hier ein Bruch unvermeidlich ist.
  - Da jeweils 1 dazu addiert wird, werden bei ansonsten annäherungsweise gleich guten Lösungen diejenigen mit weniger Zeilen bevorzugt.

Zerlegung eines Paragraphen als Optimierungsproblem 364



- Optimierungsproblem: Finde die Zerlegung mit der minimalen Summe der Unschönheitsmaße.
- Das Problem kann als gerichteter, antizyklischer Graph (DAG) dargestellt werden, bei dem die möglichen Bruchstellen als Knoten repräsentiert werden und die zugehörigen Unschönheitsmaße der jeweils vorangehenden Zeile als Kostenwert.
- Einen extra Knoten gibt es für den Anfang und ebenso gibt es nur einen Endknoten.
- Das linke Beispiel ist ein zusammengekürzter Graph, bei dem Kanten mit extremem Unschönheiten herausgefiltert wurden.

- Gegeben sei ein Paragraph mit $n+1$ Sollbruchstellen, wobei die letzte Sollbruchstelle unvermeidlich ist.
- Zulässige Zerlegungen sind dann Teilmengen aller Sollbruchstellen, die die letzte Sollbruchstellstelle enthalten.
- Dann gibt es insgesamt $\sum_{i=0}^{n} \binom{n}{i}$ mögliche Zerlegungen.
- Der Aufwand, sämtliche zulässige Zerlegungen zu untersuchen, wäre extrem.
- Wenn es als graphentheoretisches Problem betrachtet wird (Finden des kürzesten Wegs), dann lässt sich der Aufwand unter Verwendung des Algorithmus von Dijkstra auf $O(n^2)$ begrenzen. Dies ist immer noch sehr aufwendig.

- Knuth hat gezeigt, dass sich der Aufwand auf $O(n * m)$ begrenzen lässt, wobei dann m dann der Zahl der zu erwartenden Bruchstellen pro Zeile entspricht.
- Der Wert m lässt sich noch weiter reduzieren, wenn
  - Trennstellen in Wörtern erst dann betrachtet werden, wenn klar ist, dass sie benötigt werden, und
  - Zerlegungen mit extrem hohen Unschönheitsmaßen an einem Zeilenbruch von vorneherein aussortiert werden.
- In der Praxis bedeutet dies, dass der Aufwand sich auf $O(n)$ beschränkt, d.h. mit linearem Aufwand eine Zerlegung eines Paragraphen erfolgen kann. Voraussetzung dafür ist allerdings, dass Paragraphen nicht ungewöhnlich lang werden.

- Der gesamte Paragraph wird in einem Durchgang sequentiell bearbeitet.
- Es gibt eine Menge der Bruchstellen (zu Beginn nur der Anfang des Paragraphen). Eine Teilmenge davon sind die aktiven Bruchstellen (zu Beginn ebenfalls nur der Anfang des Paragraphen).
- Bei jeder Sollbruchstelle beim sequentiellen Durchgang wird die Unschönheit der Zeile gemessen ausgehend von jeder aktiven Bruchstelle bis zur neuen Sollbruchstelle.
- Wenn keine aktive Bruchstelle gefunden wird mit $|r| \leq 1$, dann wird die Sollbruchstelle nicht weiter beachtet.
- Andernfalls wird die Sollbruchstelle in die Liste der aktiven Bruchstellen aufgenommen. Dabei wird notiert, zu welcher vorherigen aktiven Bruchstelle δ minimal war.
- Aktive Bruchstellen, die zur aktuellen Sollbruchstelle einen Wert $r < -1$ haben, werden aus der Liste der aktiven Bruchstellen entfernt.
- Inaktive Bruchstellen, auf die keine aktive Bruchstelle verweist, können entfernt werden (Sackgassen).

First-Fit:

For the most wild, yet most homely narrative which I am about to pen, I neither expect nor solicit belief. Mad indeed would I be to expect it, in a case where my very senses reject their own evidence. Yet, mad am I not--and very surely do I not dream. But to-morrow I die, and to-day I would unburden my soul. My immediate purpose is to place before the world, plainly, succinctly, and without comment, a series of mere household events. In their consequences, these events have terrified--have tortured--have destroyed me. Yet I will not attempt to expound them. To me, they have presented little but horror--to many they will seem less terrible than baroques. Hereafter, perhaps, some intellect may be found which will reduce my phantasm to the commonplace--some intellect more calm, more logical, and far less excitable than my own, which will perceive, in the circumstances I detail with awe, nothing more than an ordinary succession of very natural causes and effects.

Total-Fit:

For the most wild, yet most homely narrative which I am about to pen, I neither expect nor solicit belief. Mad indeed would I be to expect it, in a case where my very senses reject their own evidence. Yet, mad am I not--and very surely do I not dream. But to-morrow I die, and to-day I would unburden my soul. My immediate purpose is to place before the world, plainly, succinctly, and without comment, a series of mere household events. In their consequences, these events have terrified--have tortured--have destroyed me. Yet I will not attempt to expound them. To me, they have presented little but horror--to many they will seem less terrible than baroques. Hereafter, perhaps, some intellect may be found which will reduce my phantasm to the commonplace--some intellect more calm, more logical, and far less excitable than my own, which will perceive, in the circumstances I detail with awe, nothing more than an ordinary succession of very natural causes and effects.

Best-Fit:

For the most wild, yet most homely narrative which I am about to pen, I neither expect nor solicit belief. Mad indeed would I be to expect it, in a case where my very senses reject their own evidence. Yet, mad am I not--and very surely do I not dream. But to-morrow I die, and to-day I would unburden my soul. My immediate purpose is to place before the world, plainly, succinctly, and without comment, a series of mere household events. In their consequences, these events have terrified--have tortured--have destroyed me. Yet I will not attempt to expound them. To me, they have presented little but horror--to many they will seem less terrible than baroques. Hereafter, perhaps, some intellect may be found which will reduce my phantasm to the commonplace--some intellect more calm, more logical, and far less excitable than my own, which will perceive, in the circumstances I detail with awe, nothing more than an ordinary succession of very natural causes and effects.

Total-Fit:

For the most wild, yet most homely narrative which I am about to pen, I neither expect nor solicit belief. Mad indeed would I be to expect it, in a case where my very senses reject their own evidence. Yet, mad am I not--and very surely do I not dream. But to-morrow I die, and to-day I would unburden my soul. My immediate purpose is to place before the world, plainly, succinctly, and without comment, a series of mere household events. In their consequences, these events have terrified--have tortured--have destroyed me. Yet I will not attempt to expound them. To me, they have presented little but horror--to many they will seem less terrible than baroques. Hereafter, perhaps, some intellect may be found which will reduce my phantasm to the commonplace--some intellect more calm, more logical, and far less excitable than my own, which will perceive, in the circumstances I detail with awe, nothing more than an ordinary succession of very natural causes and effects.