Tracing paragraphs

Udo Wermuth

Abstract

The program TeX provides more than a dozen control words for diagnostic and debugging purposes. Some of them are used often, others handle special tasks and are less frequently applied. In the latter case falls the parameter `\tracingparagraphs` that seems to be a hidden gem. This article explains what the parameter triggers if set and how an author can use the trace data to check and improve his text.

1 Introduction

The TeX software, described in *TeX: The Program* [8], implements several control sequences to show information about its work. The commands and parameters form a set of powerful tools to help diagnose errors. TeX itself contains nine primitive integer parameters for tracing ([6, p. 273]) and four primitive show commands ([6, p. 279]). The `plain` format defines additional macros ([6, p. 364]).

The tracing parameters might be classified into different groups: Some look at the settings of the installation, like `\tracingstats`, others are used mainly for developers, like `\tracingmacros`, and some (or all) can be used to get a better understanding how TeX operates. For example, the parameter `\tracingparagraphs` gives detailed insights into the inner workings of TeX's line-breaking algorithm.

Four tracing parameters are optional, i.e., their value might be ignored by a working TeX program ([6, p. 303]) without violating one of the conditions which allow a program to carry the name TeX, called the `TRIP` test [7]. For example, TeX's two control sequences `\tracingstats` and `\tracingparagraphs` are this kind of parameter. They were deemed optional as the code to implement them slows down the program even when they are not in use, i.e., set to their default value 0. That might have been annoying in the early days of TeX, but today with faster machines the effect is small and so most implementations of TeX provide them. Speed is only one aspect as some parameters trigger a huge amount of output to be written in the log file of the run. The setting `\tracingstats=1` adds only eight lines to the log file, but setting `\tracingparagraphs=1` increases the size of the log file significantly as every paragraph in the scope of this parameter is copied at least once into the log file, accompanied by several lines of trace data.

I don't remember how long I played with the parameter `\tracingparagraphs` when I learned TeX. After I gained some experience with TeX I probably read only pages 98–99 of *The TeXbook* [6], a couple of double dangerous-bend paragraphs, and did not see how to benefit from the output in my work. (Somehow I missed the hint of a real world application on page 317 of [6].) Other tracing parameters, for example, `\tracingmacros`, became an important tool to diagnose problems. Years later I read the protocol of a Q&A session with Donald E. Knuth ([12] or [13], Ch. 32) and I realized that the parameter `\tracingparagraphs` is a powerful tool too.

Section 2 gives a high-level overview of TeX's line-breaking algorithm and its parameters and in section 3 the format of the trace data for this algorithm is explained. Section 4 gives examples how I make use of the trace and section 5 shows some aspects of `\looseness` that makes it difficult to extract precisely from the trace data the range of lines in which a paragraph can be typeset without violating the current parameter settings.

All the figures except Figs. 8, 9, 10, and 12 represent data from a previous run of this article; all examples are typeset and traced by TeX during the last compilation.

2 TeX's line-breaking algorithm

This is not the place to present the details of TeX's sophisticated line-breaking algorithm but as the parameter `\tracingparagraphs` creates a trace of this algorithm, an overview will be given.

The procedure tries to break the text into lines of a given length. This length is usually `\hsize` but also `\leftskip`, `\rightskip`, `\hangindent`, or `\parshape` must be considered. Each line gets a *badness* value which is calculated as a function of the change of the available white space, i.e., the glue, which is necessary to place the content of the line into the given length. A *finite* badness is an integer between 0 and 9999, larger values are considered to be *infinite* and are not distinguished anymore ([6], p. 97). The badness is approximately $\min(100 \times r^3, 10000)$ if the ratio of used to available amount of change is called r.

Based on the badness each line is assigned a *fitness class*. A line is called

- C0. *very loose* if the badness value is 100 or more and the glue has to stretch;
- C1. *loose* if the badness value is between 13 and 99 and the glue has to stretch;
- C2. *decent* if the badness value is between 0 and 12;

C3. *tight* if the badness value is 13 or more and the glue has to shrink.

A line break can occur only at certain points; there are five possibilities ([6], p. 96). A line can be broken at

B1. *glue*, i.e., at white space, if a non-discardable item (not glue, kern, penalty or a math switch) appears before the glue;
B2. a *kern*, if it is followed by glue;
B3. *math-off*, i.e., at the end of a formula, if it is followed by glue;
B4. a *penalty* which is either entered directly into the text as an indication how desirable a break at this point is or inserted by TEX automatically, for example, in a formula;
B5. a *discretionary break* when TEX splits a word either at an explicit or an inserted implicit hyphen.

Note that TEX controls white space in math mode; in this mode B1 and B2 are not used.

Certain line breaks get a penalty based on the following parameters, listed here with their default values which the `plain` TEX format sets:

P1. `\exhyphenpenalty` (default 50) which is used if a break occurs after an explicit hyphen, i.e., a hyphen that is present in the input;
P2. `\hyphenpenalty` (default 50) which is applied at an automatically inserted hyphen;
P3. `\binoppenalty` (default 700) which is applied if a formula is broken after a binary operation;
P4. `\relpenalty` (default 300) which is applied if a formula is broken after a relation symbol.

And there is a special penalty called `\linepenalty` (default 10) that is applied to every line.

Finally, each line gets a value called *demerits* by which TEX rates the constructed lines and sets the breakpoints ([6], pp. 97–98). TEX's goal is to select those line breaks that minimize the sum of the line demerits. (TEX's decision can be changed via the integer parameter `\looseness` ([6], p. 103) to select a set of line breaks that might result in a different number of lines for the paragraph.) Demerits combine two aspects: $d = d_1 + d_2$. The first summand d_1 is based on badness and penalty. The formula by which TEX calculates the demerits d_1 follows. It shows the special role that the `\linepenalty`, let's call it l, plays. If b stands for the badness of a line and p for the penalty assigned to the break then

$$d_1 = (l+b)^2 + \begin{cases} \operatorname{sign}(p)\, p^2, & -10000 < p < 10000 \\ 0, & p \le -10000. \end{cases}$$

No line break occurs if $p \ge 10000$, and $p \le -10000$ represents a forced break.

The second aspect d_2 is the sum of fixed values:

D1. `\adjdemerits` (default 10000) is added either to the second line if adjacent lines fall in one of the fitness class pairs (very loose, decent), (very loose, tight), or (loose, tight) or if the first line is very loose.
D2. `\doublehyphendemerits` (default 10000); it is added to the second line if two consecutive lines end with a discretionary break.
D3. `\finalhyphendemerits` (default 5000) which is added to the last line if the second-last line ends with a discretionary break.

Now all but one ingredient of the algorithm has been described. The last item is a limit for the badness which the algorithm uses to decide if a line is acceptable. TEX knows two limits ([6], p. 96):

T1. `\pretolerance` (default 100) which is used as the limit in TEX's attempt to break the paragraph without hyphenation of words (breaks are still allowed at explicit hyphens, i.e., a '`-`' or a '`\-`');
T2. `\tolerance` (default 200) which is used as the limit when hyphenation of words is allowed.

The line-breaking algorithm tries in up to three passes to cut the paragraph into lines whose badness values are less than the current limit. In the first pass the limit T1 counts and no words are given to TEX's hyphenation algorithm. If this pass fails then a second pass with the limit T2 and word hyphenation is made. This pass outputs the paragraph even if it fails, except if the dimen `\emergencystretch` has a positive value. In the first case an overfull line with infinite badness is constructed. In the second case the failed second pass is followed by a third pass with word hyphenation, badness limit T2, and additional stretchability per line given by the value of the dimen `\emergencystretch`. This pass may fail too but then either the value of the dimen must be increased or "the line-breaking task is truly impossible" ([6], p. 107).

3 Format of `\tracingparagraphs`'s output

The TEXbook [6] explains on pages 98–99 the main aspects of the trace data. The full details are contained in *Computers & Typesetting*, Volume B [8] in §§ 813–890 together with the code for general printing routines like §§ 174–175, and 245.

The single assignment `\tracingparagraphs=1` triggers if trace data is written in the log file. Here is an overview of the kind of data that is output:

- *optional header* which identifies for which pass of the line-breaking algorithm the output is written.

It is one of the words `@firstpass`, `@secondpass`, or `@emergencypass`. The headers of the first and second pass are not output if the limit of T1 in section 2, the parameter `\pretolerance`, prevents the first pass, i.e., if it is negative (see § 863). It would be nice to add a hook to the log file in this situation, let's say `@hyphenationpass` to signal in a unique way the start of the trace data. But such a change violates the `TRIP` test [7] although it changes only the log file.

- *break candidates* which are considered by the algorithm as there is a valid way to break the line at this point using a previously found feasible breakpoint. The output has the form "`@<w> via @@<m> b=<x> p=<y> d=<z>`" (§ 856) where the placeholders `<x>`, `<y>`, and `<z>` are the values of the badness of the line, the penalty for the break (see P1–P4) if applicable, and the demerits of the line, whose calculation uses the `\linepenalty` and adds the values of D1–D3 if the conditions are met. The `@@<m>` documents the feasible breakpoint, after which the current line starts (the value 0 stands for the start of the paragraph). The first parameter `<w>` indicates the kind of break: It is empty if the break occurs at glue between words; otherwise it is (see B2–B5) `\kern`, `\math`, `\penalty`, `\discretionary`, or, at the end of the paragraph, `\par`. When the badness values `<x>` are calculated for an emergency pass, the values represent the data that the line-breaking algorithm uses to get the demerits, i.e, one of TeX's input values for its rating function. The "real" values for the badness as "seen in the output" depend heavily on the used stretchability given by the dimen `\emergencystretch`, but these values are not shown in the trace data. See below for an example.

Principally, three types must be distinguished:

- *inter-word breaks* that are line breaks between words or symbols, i.e., the cases B1–B4.
- *discretionary breaks* (case B5) indicated by the word `\discretionary` for `<w>`, which signals that the line break occurs within a word. Then pre-break and post-break information must be considered to construct the contents of the line.
- *end-of-par breaks* indicated by `\par`. This shows that the line break algorithm was able to process the whole paragraph. TeX rates the end of a paragraph as a forced break and assigns therefore a penalty of -10000, which does not add to the demerits (see the formula for the calculation of demerits in section 2).

Note it is possible to have several break candidates at the end of a line for different feasible breakpoints.

- *feasible breakpoint* which gives the best way to break the paragraph up to this point using the current settings of the line-breaking algorithm for one of the break candidates that appear above this feasible breakpoint. The output in the log file is the string "`@@<n>: line <a>.<b><c> t=<d> -> @@<m>`" (§ 846) where `<n>` is the new sequence number of the current feasible breakpoint, and `<m>` states the number of the feasible breakpoint which the new breakpoint needs as the previous line break. The content data between these two breakpoints is then line number `<a>`. It belongs to the fitness class `<b>` (range is 0–3 (§ 817); name is given in C`<b>`), and ends with a hyphen if `<c>` is '`-`'. The value `<d>` states the total demerits of the whole paragraph up to this line, i.e., it is the sum of the `<z>` values of the break candidates for the set of lines ending with this feasible breakpoint.

Note it is not yet determined if this feasible breakpoint will be used to construct the paragraph. The best end-of-par break names the previous feasible breakpoint for the last feasible breakpoint.

The final feasible breakpoints are treated as having a hyphen as the value of `<c>` (§ 829).

- *content data* which is the text seen by the algorithm (§ 857). It is split in small parts as the breakpoints are listed in the output too. In passes that try to hyphenate the words all hyphenation points of TeX's hyphenation algorithm are inserted.

The trace data ends with an empty line (§ 245). Note: Except for the content data all trace lines start with the symbol `@`.

Final remark: Values for the badness are sometimes stated as `*` which means that it is *infinite* according to TeX's rules. For demerits such an asterisk means that the calculation was not performed because of certain forced conditions (§ 856).

The format of the trace lines is rather terse and a lot of trace lines are written even if they do not contribute to the final line breaks. An example will help to understand the above stated description of the data.

Example 1: TeX input

```
\tracingparagraphs=1
\noindent
Note: {\sl pretolerance\/} is \the\pretolerance\
and {\sl tolerance\/} is \the\tolerance, the
{\it hsize\/} is~\the\hsize.
This is a nonsense text to serve as a
constructed example that shows all kind of trace
lines. It contains~inline mathematics and text in
columns. The formula $2\times 2^2 = 8$ is simple
mathematics as well as formula $\root3 \of 8 = 2$
or what do you think? Now a declaration or
```

```
definition for a three columns tabbing
environment is made.
\settabs 3 \columns \+&&End of example:\cr
```

TeX output

Note: *pretolerance* is 100 and *tolerance* is 200, the *hsize* is 225.0pt.

This is a nonsense text to serve as a constructed example that shows all kind of trace lines. It contains inline mathematics and text in columns. The formula $2 \times 2^2 = 8$ is simple mathematics as well as formula $\sqrt[3]{8} = 2$ or what do you think? Now a declaration or definition for a three columns tabbing environment is made.

End of example: ▯

Note: The small rectangle at the end of the previous line indicates the end of an example.

The trace data was written in the log file; here are all trace lines with numbers for identification.

Example 1 continued: Log file contents

```
 1. @firstpass
 2. \ninerm Note: \ninesl pretolerance \ninerm
     is 100 and \ninesl tolerance \ninerm is
     200, the \nineit hsize
 3. @\kern via @@0 b=2 p=0 d=144
 4. @@1: line 1.2 t=144 -> @@0
 5.  \ninerm is 225.0pt.
 6. @\par via @@1 b=0 p=-10000 d=100
 7. @@2: line 2.2- t=244 -> @@1
 8.
 9. @firstpass
10. []\ninerm This is a nonsense text to serve
     as a constructed
11. @ via @@0 b=23 p=0 d=1089
12. @@1: line 1.1 t=1089 -> @@0
13. @secondpass
14. []\ninerm This is a non-sense text to serve
     as a con-structed
15. @ via @@0 b=23 p=0 d=1089
16. @@1: line 1.1 t=1089 -> @@0
17. ex-
18. @\discretionary via @@0 b=27 p=50 d=3869
19. @@2: line 1.3- t=3869 -> @@0
20. am-ple that shows all kind of trace lines.
     It con-tains in-
21. @\discretionary via @@1 b=38 p=50 d=14804
22. @\discretionary via @@2 b=0 p=50 d=12600
23. @@3: line 2.2- t=16469 -> @@2
24. @@4: line 2.3- t=15893 -> @@1
25. line
26. @ via @@2 b=91 p=0 d=10201
27. @@5: line 2.3 t=14070 -> @@2
28. math-e-mat-ics and text in col-umns. The
     for-mula
29. @ via @@3 b=137 p=0 d=31609
30. @ via @@4 b=137 p=0 d=31609
31. @@6: line 3.0 t=47502 -> @@4
32. $2 \ninesy ^^B
33. @\penalty via @@3 b=0 p=700 d=490100
34. @\penalty via @@4 b=0 p=700 d=490100
35. @\penalty via @@5 b=123 p=700 d=517689
36. @@7: line 3.2 t=505993 -> @@4
37.  \ninerm 2[] =
38. @\penalty via @@5 b=10 p=500 d=250400
39. @@8: line 3.2 t=264470 -> @@5
40.  8$ is sim-ple math-e-mat-ics as well as
     for-mula
41. @ via @@6 b=57 p=0 d=4489
42. @@9: line 4.1 t=51991 -> @@6
43. $[][] =
44. @\penalty via @@6 b=72 p=500 d=266724
45. @\penalty via @@7 b=1 p=500 d=250121
46. @@10: line 4.3 t=314226 -> @@6
47.  2$
48. @\math via @@7 b=2 p=0 d=144
49. @\math via @@8 b=130 p=0 d=29600
50. @@11: line 4.0 t=294070 -> @@8
51.  or
52. @ via @@8 b=7 p=0 d=289
53. @@12: line 4.2 t=264759 -> @@8
54. what do you think? Now a dec-la-ra-tion or
55. @ via @@9 b=31 p=0 d=1681
56. @@13: line 5.1 t=53672 -> @@9
57. def-i
58. @\discretionary via @@9 b=4 p=50 d=2696
59. @\discretionary via @@10 b=119 p=50 d=29141
60. @@14: line 5.2- t=54687 -> @@9
61. -
62. @\discretionary via @@9 b=20 p=50 d=13400
63. @\discretionary via @@10 b=82 p=50 d=20964
64. @@15: line 5.3- t=65391 -> @@9
65. ni-
66. @\discretionary via @@10 b=14 p=50 d=13076
67. @\discretionary via @@11 b=106 p=50 d=15956
68. @@16: line 5.0- t=310026 -> @@11
69. tion
70. @ via @@10 b=4 p=0 d=196
71. @ via @@11 b=2 p=0 d=10144
72. @ via @@12 b=107 p=0 d=23689
73. @@17: line 5.0 t=288448 -> @@12
74. for
75. @ via @@12 b=0 p=0 d=100
76. @@18: line 5.2 t=264859 -> @@12
77. a
78. @ via @@12 b=25 p=0 d=1225
79. @@19: line 5.3 t=265984 -> @@12
80. three col-umns tab-bing en-vi-ron-ment is
81. @ via @@13 b=57 p=0 d=4489
82. @@20: line 6.1 t=58161 -> @@13
83. made.
84. @\par via @@14 b=48 p=-10000 d=8364
85. @\par via @@15 b=10 p=-10000 d=5400
86. @\par via @@16 b=0 p=-10000 d=15100
87. @\par via @@17 b=0 p=-10000 d=10100
88. @\par via @@18 b=0 p=-10000 d=100
89. @\par via @@19 b=0 p=-10000 d=100
90. @\par via @@20 b=0 p=-10000 d=100
91. @@21: line 7.2- t=58261 -> @@20
92. @@22: line 6.3- t=63051 -> @@14
93.
```

▯

As expected, the trace starts with `@firstpass` for the first paragraph. Line 2 is the content data preceded by a `\ninerm`, which was added by TeX's routines (§ 174); it was not part of the input. As you see all font switching commands are spelled out explicitly with the control sequence that TeX associates with the requested font. Line 3 outputs the first break candidate; it is a break at the italic correction and so `<w>` is `\kern`. The badness is 2, i.e., the line is decent, penalty is 0 and therefore the demerits are $(10 + 2)^2 = 144$. That the line is decent can been seen in line 4 of the listing as a ".2" appears after the line number (see C2). Line 6 documents an end-of-par break, so the first pass was successful. As explained above the `<c>` is '`-`' merely because of the end of the paragraph; it does not indicate that the final line ends with an hyphen. The penalty is -10000 to mark a forced break; this value is ignored as stated in the formula for the calculation of demerits and therefore the demerits of the second line are $(10 + 0)^2 = 100$. The total demerits are the sum of the line demerits: $144 + 100 = 244$ (see line 7).

The data of the second paragraph is much more interesting. Again the header line of the first pass is printed (line 9), but only one feasible breakpoint is found and output. This means TeX's algorithm was unable to find a second break candidate, so it stops this pass, and starts the second pass, which outputs its header line (line 13). Note however, that the content data in line 10 has in front of the font information, which is inserted as described above, the construction `[]`. This stands for output that TeX cannot show; in this case it is the white space created by the indentation. This is the normal behavior for all non-printable items (see § 175). In the second pass hyphenation is tried so TeX shows all hyphenation points in the words by inserting hyphens. Compare line 14 with line 10: TeX has placed hyphens in the words "nonsense" and "constructed". The breakpoint in the lines 17–19 is a discretionary break. In line 18 the penalty is 50, the value of `\hyphenpenalty`. In line 19 the `1.3-` states that it is a tight line ending in an hyphen.

Let's look at some interesting points without going through all the details of the trace.

- Lines 21 and 22 show that more than one break candidate can occur but they must use different feasible breakpoints after the "via".
- In lines 33–35 and in lines 38 & 44–45 penalty breaks are shown in math mode. In the first set the break occurs after a binary operation and the `\binoppenalty` is applied. The break in the second set is made after a relation and the `\relpenalty` is used.
- An example for a line break after a math-off is given in lines 47–50.
- Lines 57ff. show a discretionary break in the word "definition" which contains the ligature "fi". An implicit discretionary break is used for the ligature with the pre-break text "f-", the post-break text "i" and the no-break text "fi". Both pre-break and post-break text are stated in the content data of line 57. After feasible breakpoint 14 just a hyphen is added to the line.
- The end-of-line break candidates in the lines 84–90 signal the successful completion of the second pass. Lines 88–90 seem to be equivalent judged by the data in the lines, but the path via feasible breakpoint 20 has the lowest total demerits and therefore it is stated in the feasible breakpoint `@@21`.
- Feasible breakpoint `@@22` gives an alternative path via feasible breakpoint `@@14`. Its total demerits are higher than for feasible breakpoint `@@21` therefore it is not used by TeX. But the number of lines is lower and so it would be a valid path if the author states `\looseness=-1` (see example 2 below).
- The trace ends with an empty line. It is shown here but the other examples will omit it.

The paragraph is now built from bottom to top: The last line is between `@@20` and `@@21`, its content is "made" (see line 83). The second last line starts at `@@13` with the concatenation of the content in lines 57, 61, 65, 69, 74, 77, and 80. The third last line begins at `@@9`, the next at `@@6`, then at `@@4`, the second line of the paragraphs starts at `@@1` and the first at the beginning of the paragraph, of course. Their content is built from the lines carrying the content data between the mentioned feasible breakpoints.

So the feasible breakpoints `@@2`, `@@3`, `@@5`, `@@7`, `@@8`, `@@10` to `@@12`, and `@@15` to `@@19` are never used in the line breaks of the paragraph chosen by TeX (`@@14` and `@@22` are used if only six lines are built).

The text entered in the tabbing environment is not repeated in the trace as the line-breaking algorithm is not called and so no trace lines are output.

4 Applications

The description in the previous section makes clear that the task of decoding the tracing information by hand is difficult, or at least time consuming. Moreover, the trace data must be enhanced as example 1 has shown: There are two ways to break the paragraph and the selection involves the knowledge of the current setting of `\looseness`. So its value has to be put in the log file too. The value is reset to 0 after each paragraph; TeX uses the value that it has at the end of the paragraph ([6, p. 349]). The following code writes the parameter to the log file; here it is applied to the second paragraph of example 1.

Example 2: TEX input

```
\let\orglooseness=\looseness
\def\writelooseness{% output looseness to log
  \immediate\wlog{@ looseness \the\orglooseness}}
\def\setlooseness{% enhanced version of looseness
  \afterassignment\writelooseness\orglooseness}
\let\looseness=\setlooseness

\tracingparagraphs=1 \looseness=-1
This is a nonsense text to serve as a ...
```

TEX output

This is a nonsense text to serve as a constructed example that shows all kind of trace lines. It contains in-line mathematics and text in columns. The formula $2 \times 2^2 = 8$ is simple mathematics as well as formula $\sqrt[3]{8} = 2$ or what do you think? Now a declaration or definition for a three columns tabbing environment is made. ▯

The first line of the excerpt from the log file shows the output of the macros; note that the line might not appear directly in front of the start of the data but it is always first. The trace data is nearly identical to the data shown for example 1. All trace lines are present but their sequence is changed and at the end one more feasible breakpoint is added for a six-line paragraph; in this case the ligature in "definition" is resolved and the break is after the 'f'.

Example 2 continued: Log file contents

```
 1. @ looseness -1
 2. @firstpass
 3. []\ninerm This is a nonsense text to serve
      as a constructed
 4. @ via @@0 b=23 p=0 d=1089
 5. @@1: line 1.1 t=1089 -> @@0
 6. @secondpass
...
76. made.
77. @\par via @@19 b=0 p=-10000 d=100
78. @\par via @@18 b=0 p=-10000 d=100
79. @\par via @@17 b=0 p=-10000 d=10100
80. @\par via @@16 b=0 p=-10000 d=15100
81. @\par via @@15 b=10 p=-10000 d=5400
82. @\par via @@14 b=48 p=-10000 d=8364
83. @@21: line 6.2- t=70791 -> @@15
84. @@22: line 6.3- t=63051 -> @@14
85. @\par via @@20 b=0 p=-10000 d=100
86. @@23: line 7.2- t=58261 -> @@20
```
▯

The integer parameter looseness influences the line-breaking algorithm and makes it select paragraph lines that are not the optimum. Most often this price is worth being paid to improve the page breaking. It would be useful to inform an author about the possibilities to shorten or to lengthen a paragraph. Section 5 discusses this topic in more detail.

In the following I do not discuss lines with overfull boxes, etc. These are reported by TEX during the run. My recommendation is that an author react to these messages. Moreover, only the multiline paragraphs are handled in the figures. This article contains many single-line paragraphs through the verbatim listings but they are not discussed. The following tasks are addressed:

1. Find hyphenated words.
2. Find the longest sequence of hyphenated lines.
3. Find paragraphs that contain visually incompatible lines.
4. Find sequences of lines starting or ending with the same word.
5. Use statistics to learn about the overall appearance of the text.
6. Perform actions to eliminate detected problems.

List of hyphenated words. In a Q&A session Donald E. Knuth was asked why TEX does not provide a way to generate a list of hyphenated words of a text. He answered that a little filter program can do the work if all relevant tracing is switched on ([12], p. 365 or [13], pp. 620–621).

There are several ways to get the hyphenated words, for example, one could set `\hbadness=-1` and check the output for lines ending in a hyphen. See for example, exercise 28 of [11]. (Note: Such lines often start with the words "loose" and "tight" but that does not refer to C1 and C3, resp. See [6], p. 302.) A problem might be that the part of a word at the beginning in the last line is not output. Another approach is to filter the output of `dvitype` [10]. I decided to use the trace data of the command `\tracingparagraphs`. Of course, that meant writing the "little filter program" for efficient extraction of data. Such a program reads the trace data, chooses the right final feasible breakpoint, goes back through the chain of feasible breakpoints looking at the content data and saves all hyphenated words. The script might output a list like the one in Fig. 1.

It is much easier to check such a file for bad hyphenation than to go through the DVI output and check every end of line. Changes to this list between runs can be analyzed by a diff command. And more is possible: As TEX hyphenates all the words in certain passes they can be collected in a list, supplied with corrected hyphenation points if necessary, and saved in a database. With time this database

```
A '=' marks the hyphenation point.

   1: con=trol
   2: pa=ram-e-ters
   3: di-ag=nose
      ...
 160: Stan=ford
 161: Pro=gram
```

Figure 1: List of hyphenated words

```
Show line-break statistics based on the log file of a TeX run.

texput.log analyzed on 05/11/16, 14:48:16

   Hsize: 225.0pt; Parindent: 20.0pt; Parfillskip: 168.75pt plus 13.49945pt minus 168.75pt

Parameter settings for line breaking:

  Pretolerance: 100; Tolerance: 200; Emergencystretch: 0.0pt
   Linepenalty: 10; Exhyphenpenalty: 49; Hyphenpenalty: 50
  Binoppenalty: 700; Relpenalty: 500
   Adjdemerits: 10000; Doublehyphendemerits: 10000; Finalhyphendemerits: 5000

Single-line paragraphs            :   570
                  vloose  loose decent  tight
Line categories:       0      0    563      7


Multiline Paragraphs              :   201
with total lines                  :  1155
and lines pro par between         :     2 and 25
with demerits between             :  -156 and 545991

Demerits in range   >1,000,000 200,000 50,000 20,000 10,000  5,000      0
   Positive values:          0       2     17     46     25     26     84
   Negative values:          0       0      0      0      0      0      1
```

Figure 2: General information about this article

represents an author's active vocabulary and the list of all hyphenated words can be checked against the database to find wrong hyphenation points that can be given as exceptions using the control word `\hyphenation`. And new entries enlarge the database (for this article by 56 words). The list can also be generated by `\pretolerance=-1` (no first pass), set `\emergencystretch=\hsize` (allow awful lines), apply `\looseness=1000` to every paragraph, and run `dvitype` [10] to find all hyphenated words.

A wrong hyphen in a word must be corrected, of course. Either declare the word at the beginning of the document as a hyphenation exception, or add `\-` to the word at the right place, or put a short word into an `\hbox` to avoid the hyphen. The last two methods are useful if the word occurs only a few times in the text. For this text three hyphenation exceptions are specified: `Eng-lish`, `stretch-ability` and `Mas-sa-chu-setts`.

Note: In order to distinguish between explicit, i.e., author entered, and implicit, i.e., TEX inserted, hyphens I subtract 1 from `\exhyphenpenalty` if it equals `\hyphenpenalty` (see Fig. 2). This changes the calculations of TEX during the tracing compared to the normal run, but the impact is usually small, at most 99 demerits for a break at an explicit hyphen with the defaults of `plain` TEX.

```
13% lines with implicit hyphen    :   161
 0% lines with explicit hyphen    :     2
    with longest sequence         :     3
    and hyphenated final lines    :    22
```

Figure 3: Global statistics about hyphenated lines

Martin Budaj wrote a script in Perl [1] that finds the hyphenated words in the trace data and outputs a list similar to Fig. 1. A LuaTEX solution is described in [4]. Its author, Patrick Gundlach, developed also the package [3] for LuaLATEX to show all hyphenation points using tiny vertical marks inside the text similar to the triangles in Fig. 1 of [5].

Counting consecutive hyphenated lines. As all the lines are checked for discretionary breakpoints, overall statistics can be collected to give information on the longest sequence of consecutive lines that are hyphenated. For example, the report for an early draft of this article showed that the longest sequence of hyphenated lines was 5, which is too long according to [2], 3.11: *When four or more lines end with a hyphen or the same word, word spacing should be adjusted to prevent such "stacks."* The current count for this article is shown in Fig. 3.

The length of the longest sequence of hyphenated lines is valuable and easily output by the script. The author decides if this length is acceptable or not. How can the hyphen stack be reduced? An author has several possibilities:

A0. change the words of the paragraph;
A1. increase the penalties and demerits that have to do with hyphenation for this paragraph;

A2. lower the `\tolerance` and use a positive value for `\emergencystretch` in this paragraph;
A3. try to make the paragraph a line longer (or shorter) using `\looseness` (maybe supported by a positive value of `\emergencystretch`);
A4. improve the chance of the first pass by increasing `\pretolerance` for this paragraph;
A5. put the third or fourth hyphenated word in an hbox if it is a short word.

Action A0 is always an option and it is guaranteed to be successful; the other actions might fail. Actions A1, A2, and A4 should not be made for the whole text; apply the parameter setting only to the "bad" paragraph; see below.

Relationships between lines. Instead of counting and printing numbers of lines of a certain type the relationships between lines can be documented. As an example I use the data of the fitness classes C0–C3.

```
  4% very loose lines              :    53
 21% loose lines                   :   243
 62% decent lines                  :   713
 13% tight lines                   :   146
```

Figure 4: Distribution of fitness classes

Figure 4 reports on the distribution of the lines into the four classes. The distribution shows that less than two-thirds of all lines in multiline paragraphs are decent. Michael F. Plass and Donald E. Knuth gave in [5] some results for the second volume of *The Art of Computer Programming*, a book with 702 pages, 5526 paragraphs, and 21057 lines. (I refer to this article through the reprint in [13].) The article was published in 1981 one year before TeX82 was released and the algorithm was changed a little bit for TeX82, see [8, § 813]. But the algorithm is close enough that the data can serve as an example. In [13], p. 125, Fig. 19, Donald E. Knuth shows the distribution of lines into the fitness classes. A rough measurement of this data together with the definitions on page 112 of [13] gives the distribution (2, 14, 79, 5)% for (very loose, loose, decent, tight). So the data for the present article is worse. On the other hand the book has an hsize of 468 pt compared to the 225 pt of this column; the line-breaking job is easier for the book.

The numbers for transitions from one class to another provides additional insights (see Fig. 5). It gives the volume of visually incompatible lines. Only a few lines are incompatible, about 2.9%. In summary the data looks acceptable to me. Only a few jumps from very loose or loose to tight occur. The majority of transitions is listed on the "diagonal",

```
From / To: vloose  loose decent  tight
vmode           1      2    169     29
vloose         18     11     12      3
loose          15     57    109     15
decent         16    157    339     75
tight           3     16     84     24
```

Figure 5: Transitions between fitness classes

so lines of the same class follow most often. One intentionally bad paragraph is the second paragraph in the introduction. There, lines 3 and 4 are very loose, lines 5 and 7 are tight, and line 6 is loose.

To improve a paragraph with excessive transitions between visually incompatible lines the abovementioned action A2 seems to be the best choice. Usually it reduces the number of very loose lines if the parameters are chosen carefully; see below.

Distribution of demerits. When the topic "distribution" is considered, the idea of showing the data of TeX's rating values, the demerits, comes to mind. Of course, a script can easily document them and I use a set of ranges for positive and negative values to categorize the data points. The scripts calculate the distribution as shown in Fig. 2. Negative values can occur as an author can specify them via `\penalty`. (The paragraphs with negative demerits appear in the references where URLs are broken by macros.)

The problem that I have with the data is the lack of a trigger for action. A twenty-line paragraph with lines all having badness 10 and no hyphenations has the same demerits as a two-line paragraph with lines having badnesses of 10 and 0 and a hyphen after the first line. Which one is better? What can be done to improve the situation? Is there a problem at all?

But the statistic is useful to give a general overview. When the default values of TeX are active, paragraphs with demerits in the range 0–5000 can have only one hyphenated word, which is not at the end of the second last line: The penalty for a hyphenated word is $50^2 = 2500$ so there cannot be two if the total demerits are at most 5000 and, of course, the value of `\finalhyphendemerits` was not applied. Similarly the paragraphs in the range 5001–10000 have no stack of two hyphens and no visually incompatible lines. The next range might have just one of such things but only once.

So one can concentrate on the few paragraphs that have very high values of demerits. But let me state again: A high value does not imply a problem. In this article most paragraphs with high demerits appear in examples. The paragraph above the description of the fitness classes on the first page is an exception (see Fig. 11): It has the highest demerits (see Fig. 2) because of 13 lines, three have a badness above 50, one more than 100, one `\binoppenalty` and one `\hyphenpenalty` are charged, and it has one pair of visually incompatible lines. But only the break in the formula might trigger a change.

```
number of words in paragraphs      :  8417
max. words in one line             :    14
one-word lines (multiline par)     :    35

Repeat word >4 chars at start      :     3
                  or end of line   :     2
Repeat shorter word at start       :    10
                  or end of line   :     7
Longest sequence at start of line  :     2
Longest sequence at end of line    :     2
```

Figure 6: Several global counts

More global statistics. Other counts can be calculated. For example, I use an experimental count of lines that start or end with the same word or syllables as the previous line (see Fig. 6). In this text most occurrences of stacks of words with at least five letters appear in the examples. As the longest sequence is two, there is no problem according to [2]. Otherwise an author should tie one of the words to the previous or the following word if the stack appears at the start or the end of the line, resp. It seems best to connect the tie to the shortest possible word or syllable. The number of hyphenated lines might increase or an overfull line is created if the stack appears near the beginning of the paragraph; then the text must be rewritten to avoid it.

Other general statistics can be generated and they may trigger actions by an author although the interpretation is more complicated. Such statistics do not point to a certain situation in the input which might be changed.

```
 50% successful in the first pass  :   100
 43% successful in second pass     :    86
  2% successful without first pass :     5
  5% needed an emergency pass      :    10
```

Figure 7: Global statistics about passes

Figure 7 shows the distribution of passes for this article. But how can this data be interpreted? The emergency pass is used in several examples and the list of references where a positive value for the `\emergencystretch` is specified. The main question is: Is the shown distribution between first and second pass OK? In [13], p. 123, Donald E. Knuth writes that in the second volume of *The Art of Computer Programming* (TAOCP) only 5% of all paragraphs needed a second pass and only 2.26% lines ended in a hyphen. So compared to these data the values are

```
 55% successful in the first pass  :   108
 44% successful in second pass     :    87
  0% successful without first pass :     0
  1% needed an emergency pass      :     2

  5% very loose lines              :    49
 20% loose lines                   :   197
 64% decent lines                  :   643
 11% tight lines                   :   110

 11% hyphenated lines              :   109
 11% lines with implicit hyphen    :   105
  0% lines with explicit hyphen    :     4
     with longest sequence         :     3
     and hyphenated final lines    :    18
```

Figure 8: Global statistics of another article [15]

bad. On the other hand the columns of this journal are much smaller than the `\hsize` of the book. It needs some judgment to decide if the values are acceptable or if some parameters should be changed. As I have written another article for this journal [15] its data can be used for a comparison. Figure 8 shows its values for the data shown in Figs. 7, 4, and 3. The values of the present article are worse. But I do not change anything as, for example, some paragraphs have been *designed* to make the first pass fail.

What options does an author have to respond to unwanted global statistics? Of course, when an author has to use a given format with given parameters often only rewriting of the text is possible. The distributions are useful to detect problems in the general setup. If a small percentage is seen for the

```
\newtoks\TRsavedLBparameters
\newif\ifprotectLBparameters
\def\SaveallLBparameters{% store 11 parameters
 \ifprotectLBparameters
 \else\protectLBparameterstrue
  {\edef\saveparameters{%
    \global\TRsavedLBparameters=\expandafter
     {\the\emergencystretch:\the\pretolerance:%
      \the\tolerance:\the\linepenalty:%
      \the\hyphenpenalty:\the\exhyphenpenalty:%
      \the\binoppenalty:\the\relpenalty:%
      \the\adjdemerits:\the\doublehyphendemerits:%
      \the\finalhyphendemerits}}\saveparameters}\fi}
% usage:\afterassignment\RestoreLBparameters
\def %  \emergencystretch=\the\TRsavedLBparameters
\RestoreLBparameters:#1:#2:#3:#4:#5:#6:#7:#8:#9:{%
  \pretolerance=#1 \tolerance=#2 \linepenalty=#3
  \hyphenpenalty=#4 \exhyphenpenalty=#5
  \binoppenalty=#6 \relpenalty=#7
  \adjdemerits=#8 \doublehyphendemerits=#9
  \finalhyphendemerits=}
% #1: i - s c o ii i- ... (is for --) or \hskip<x>pt
\def\setEMstr(#1){SaveallLBparameters
  \setbox0=\hbox{#1}\emergencystretch=\wd0 }
```

Figure 9: Support macros for Fig. 10

```
#Pars  P Demerits Breakpoints Lines opt  L Lines-found B-inf B-min B-max vloos loose decnt tight hyphen seq last
================================================================================================================
    1: 1      100           1    1    1  0           1           0     0                 1
    2: 1      100           1    1    1  0           1           0     0                 1
    3: 1      100           1    1    1  0           1           0     0                 1
    4: 2    15554      (0)/25    8    8  0     (0)/8-9           0    69           1     5     2   1/ 1
    5: 1      100           1    1    1  0           1           0     0                 1
    6: 2    40159      (2)/15    8    8  0       (2)/8           0    79           3     5         3/ 3   3
    7: 2   120270      (2)/13    8    8  0       (2)/8           3   178     2     2     1     3   2/ 2   2
    8: 2    44163      (5)/52   21   21  0      (5)/21           0    89           4    13     4   3/ 3
    9: 2    63786      (1)/46   13   13  0   (1)/13-14           0    86           3     6     4   6/ 6   3    Y
   10: 2    18559      (2)/16    9    9  0       (2)/9           0    30           4     4     1   3/ 3        Y
   11: 2     3565       (0)/9    4    4  0       (0)/4           0    14           1     3         1/ 1
   12: 1      100           1    1    1  0           1           0     0                 1
   13: 2     4290       (1)/4    4    4  0       (1)/4           0    29                 2     2   1/ 1
   14: 2   545991     (14)/31   13   13  0     (10)/13           0   146     1     3     8     1   1/ 1
```

Figure 11: Information about the first several paragraphs of this article

first pass check that (1) no non-breakable items like large hboxes or verbatim strings (as in this article) are present, (2) `\pretolerance` is not too low, and (3) the `\hsize` is appropriate, i.e., not too small for justified text. In such situations where the hyphenation passes appear in the expected amount but the number of hyphenated lines is high, check additionally that (4) the values of the parameters for hyphenation are set reasonably.

A few examples. The actions that change some line-breaking parameters should apply that change only for a single paragraph. The technically named `try`... macros in Fig. 10 (supported by those in Fig. 9) change parameters and with the command `\defaultlinebreaking` after an empty line or a `\par` the old parameters are reset. In most cases action A0, i.e., change the wording, is the best solution; only if this is not possible the actions A2, A3, or A4 should be tried.

In the first example the action A2, i.e., a lower `\tolerance` with `\emergencystretch`, is applied to the second paragraph of the introduction. The parameter to the macro is a string to specify the length of the `\emergencystretch`; an "`\hskip<dimen>`" is

```
\def\defaultlinebreaking{% reset parameters
 \ifprotectLBparameters\protectLBparametersfalse
  \afterassignment\RestoreLBparameters
  \emergencystretch=\the\TRsavedLBparameters\fi}
\def\tryonlyfirstpass{\SaveallLBparameters
 \pretolerance=125 }
\def\tryonlysecondpass{\SaveallLBparameters
 \pretolerance=-1 }
\def\trythirdpassD(#1){\setEMstr(#1)%
 \tolerance=125 \ignorespaces}
\def\trylesshyphens{\SaveallLBparameters
 \hyphenpenalty=75 \doublehyphendemerits=20000
 \exhyphenpenalty=55 \finalhyphendemerits=7500 }
```

Figure 10: Set of useful macros

possible too. The characters "i-sco" cover the range of 5–9 basic units of width in `cmr10`; see [9]. This makes the additional stretchability individual to the paragraph.

Example 3: TeX input

```
\trythirdpassD(oo)
The tracing parameters might be classified ...
```

TeX output

The tracing parameters might be classified into different groups: Some look at the settings of the installation, like `\tracingstats`, others are used mainly for developers, like `\tracingmacros`, and some (or all) can be used to get a better understanding how TeX operates. For example, the parameter `\tracingparagraphs` gives detailed insights into the inner workings of TeX's line-breaking algorithm. ▯

The paragraph is one line longer; two lines are loose, five decent, then follows a tight and a decent line. No visually incompatible pair is reported. The documented badness values are 91, 43, 1, 7, 5, 8, 3, 19, and 0 instead of 3, 25, 171, 178, 37, 21, 41, and 28. Six lines have a lower value than before. But the reporting does not include the additional stretchability. The true badness values can be seen using `\hbadness=-1`. Then the line badnesses are 1019, 239, 11, 69, 18, 41, 53, 19, and 0; two visually incompatible pairs are present. The first line is so bad that TeX reports an underfull hbox. An author should expect looser lines if a paragraph is lengthened.

```
\def\trylongerparD(#1){\setEMstr(#1)%
 \finalhyphendemerits=0 \adjdemerits=5000
 \looseness=1 \ignorespaces}
```

Figure 12: Stronger than `\looseness=1`

The macros can be combined and more macros are possible, for example, Fig. 12 increases forces

```
--#Par---(#Lines [#per pass Looseness+# for min demerits])---
    #Line   Badness  Penalty Demerits FitClass  -? ST #w
========================================================
--1---(1 [1 L0+1])-----------------------------------
        1:        0   -10000      100   decent         2 Tracing paragraphs
--2---(1 [1 L0+1])-----------------------------------
        2:        0   -10000      100   decent         2 [] Udo Wermuth
--3---(1 [1 L0+1])-----------------------------------
        3:        0   -10000      100   decent         1 Abstract
--4---(8 [(0)/8-9 L0+8])-----------------------------
        4:        0       50     2600   decent   Y     9 The pro-gram T[]X pro-vides more than a dozen con-
        5:       29        0     1521     loose        6 trol words for di-ag-nos-tic and de-bug-ging pur-poses.
        6:        0        0      100   decent         9 Some of them are used of-ten, oth-ers han-dle spe-cial
        7:        9        0      361   decent         9 tasks and are less fre-quently ap-plied. In the lat-ter
        8:        1        0      121   decent         5 case falls the pa-ram-e-ter \tracingparagraphs that
        9:       57        0     4489  tight          10 seems to be a hid-den gem. This ar-ti-cle ex-plains what
       10:        1        0      121   decent        10 the pa-ram-e-ter trig-gers if set and how an au-thor can
       11:       69   -10000     6241  tight          10 use the trace data to check and im-prove his text.
--5---(1 [1 L0+1])-----------------------------------
       12:        0   -10000      100   decent         1 1 Introduction
--6---(8 [(2)/8 L0+8])-------------------------------
       13:        0        0      100   decent         8 The T[]X soft-ware, de-scribed in T[]X : The Pro-gram
```

Figure 13: Information about the lines of the first paragraphs of the article

to lengthen a paragraph, i.e., it implements action A3. Note that the paragraph does not change with a simple `\looseness=1`.

Example 3 continued: TeX input

```
\trylongerparD(i)
The tracing parameters might be classified ...
```

TeX output

The tracing parameters might be classified into different groups: Some look at the settings of the installation, like `\tracingstats`, others are used mainly for developers, like `\tracingmacros`, and some (or all) can be used to get a better understanding how TeX operates. For example, the parameter `\tracingparagraphs` gives detailed insights into the inner workings of TeX's line-breaking algorithm. ▯

The typical pattern of `\looseness=1` appears: The new line contains only the last word or a part of it. But the result looks better than before.

Next, stacks of hyphens are removed.

Example 4: TeX input

```
\trylesshyphens\noindent
The program \TeX, described in \TP\ [...
```

TeX output

The TeX software, described in *TeX: The Program* [8], implements several control sequences to show information about its work. The commands and parameters form a set of powerful tools to help diagnose errors. TeX itself contains nine primitive integer parameters for tracing ([6, p. 273]) and four primitive show commands ([6, p. 279]). The `plain` format defines additional macros ([6, p. 364]). ▯

The original text appears as the first paragraph of section 1. It is one of the paragraphs with the longest sequence of hyphenated lines in this text (see Fig. 11). The best solution is to insert "the book" after "in" in the first line, but here `\trylesshyphens` is also successful. Sometimes this command does not work, for example, if the stack is at the beginning of the paragraph; more penalties and demerits might not change the first line break. The macro of Fig. 12 might help; next it is applied to a statement in [12], p. 358, where `\trylesshyphens` is not successful. I use `\trylongerparD(Ar)` for the second paragraph.

Example 5: TeX output

So instead, I worked only at Stanford, at the Artificial Intelligence Laboratory with the very primitive equipment there. We did have television cameras, and my publisher, Addison-Wesley, was very helpful — they sent me the original press-printed proofs of my book, from which *The Art of Computer Programming* had been made. The process in the 60s ...

So instead, I worked only at Stanford, at the Artificial Intelligence Laboratory with the very primitive equipment there. We did have television cameras, and my publisher, Addison-Wesley, was very helpful — they sent me the original press-printed proofs of my book, from which *The Art of Computer Programming* had been made. The process in the 60s ... ▯

Details for paragraphs and lines. How to find the word or section in the text which is responsible for a bad value in the statistics? My solution involves the scripts creating two additional files, one with the data about the paragraphs, the other listing the values of all lines (see Figs. 11 and 13).

Figure 11 shows the following information for each paragraph in one line: a sequential number, the number of passes, the number of breakpoints in each pass, the number of lines used, the optimal number of lines, the active looseness, lines found in each pass, number of lines with infinite badness, the minimal badness, the maximal badness, the number of lines that are very loose, loose, decent, and tight, the number of hyphens and implicit hyphens, the longest sequence of hyphenated lines, and a flag to indicate if the second last line is hyphenated. Entries in parentheses stand for failed passes, slashes separate the data of the passes. Figure 13 lists all the details about the lines separated by dashed lines that repeat some data of the paragraph. The dashed lines show the number of the paragraph, the number of lines created (for all passes), lines found per pass, the active looseness, and the optimal number of lines. For each line the line number, the line badness, the penalty at the end of the line, the line demerits, the fitness class, three flags for a hyphen at the end of the line, stacks at the start or end of the line, an approximation to the number of words and the content of the line is output.

To locate, for example, paragraphs that have the longest sequence of lines that end with a hyphen, check the column "seq" in the list of paragraphs (Fig. 11) and go to the entries for this paragraph in the list of lines (Fig. 13) to see the text. In some cases the list of lines can be consulted directly. For example, in the column "FitClass" the word "tight" is moved to the left and the word "loose" to the right. This helps to find visually incompatible lines.

My REXX scripts output probably too much data. Everything that the trace data shows is printed. At least it serves as educational material.

5 Remarks about `\looseness`

Let's look a little bit closer at the integer parameter `\looseness` and how it influences the line-breaking algorithm and as a consequence also the trace data output. It would be nice to inform an author about the number of lines his paragraph can have.

Example 2 has shown that the looseness does not force the algorithm to make a second pass. Only when a pass cannot provide the desired number of lines does TeX start the next pass because the previous pass counts as failed. Therefore, some statements in [14] are wrong in general.

In this section the following facts are shown.

1. In different passes a paragraph can have different number of lines.
2. The use of looseness might result in the execution of a second or third pass.
3. A possible emergency pass is not executed if a previous pass is successful.
4. A possibility to shorten a paragraph with the same pass is not always reported explicitly in the trace data.
5. Similarly, a possibility to lengthen a paragraph might not be reported.
6. The use of looseness might result in different line breaks even if no additional pass is run.
7. This can also happen with a "neutral" parshape.

Different number of lines in the passes. When TeX has successfully finished a pass, it builds from the feasible breakpoints the paragraph with the lowest total demerits. During this process the best number of lines N for the paragraph is also determined. A non-zero looseness forces TeX's algorithm to go again through the feasible breakpoints but this time it picks those that change the number of lines by the given value of `\looseness`. If this is not possible, the pass fails and, if it is not the final pass, the next pass is executed. The last pass outputs the paragraph even in a failed situation. The number of lines is then the best approximation that TeX has found to the sum of lines for lowest total demerits and the looseness value. Note that the number N is determined individually for each pass. The value for the second pass might be equal to the value of a successful first pass. But other cases are possible too, as the following example shows.

Example 6: TeX input

```
Hi \TeX. Tell me how is the following long
word hyphenated: ‘antidisestablishmentarianism’?
Now do it.

\noindent Hi! \TeX! Tell me: How is the
following long word broken
’pneumonoultramicroscopicsilicovolcanoconiosis’?
I am sure that you are an expert in hyphenation,
right \TeX?

\smallskip \pretolerance=-1

Hi \TeX. Tell me how is the following long ...
```

TeX output

Hi TeX. Tell me how is the following long word hyphenated: 'antidisestablishmentarianism'? Now do it.
Hi! TeX! Tell me: How is the following long word broken 'pneumonoultramicroscopicsilicovolcanoconiosis'? I am sure that you are an expert in hyphenation, right TeX?

Hi TeX. Tell me how is the following long word hyphenated: ‘antidisestablishmentarianism’? Now do it. Hi! TeX! Tell me: How is the following long word broken ’pneumonoultramicroscopicsilicovolcanoconiosis’? I am sure that you are an expert in hyphenation, right TeX? ▯

The first two paragraphs are typeset in the first pass. When that pass is suppressed, as in the third and fourth paragraphs, the pass which tries hyphenation is the only available pass. As the output shows, instead of three lines once two and once four are built. Larger differences between passes are possible as [14] points out. In the case that the second pass creates N lines and the first pass $N + 2$, a `\looseness=1` which the first pass cannot fulfill but the second can result in a shorter paragraph — and TeX claims success. The shortest paragraph with this property that I was able to construct with text in `cmr9` and an `\hsize` of 225 pt has $N = 39$.

Therefore, if `\looseness=-1` is applied to the second paragraph of example 6 with three lines the result is a successful (based on TeX’s rating) second pass that shortened a four line paragraph to three lines. Even if `\emergencystretch` has a positive value TeX does not run a third pass. The paragraph looks identical to the output of the first pass.

An emergency pass is made if the second pass fails to create the requested number of lines.

Example 7: TeX input

```
\pretolerance=-1

\looseness=1 \noindent Hi! \TeX! Tell me:
How is the following long word broken
’pneumonoultramicroscopicsilicovolcanoconiosis’?
I am sure that you are an expert in hyphenation,
right?

\emergencystretch=6.75pt

\looseness=1 \noindent Hi! \TeX! Tell me: ...
```

TeX output

Hi! TeX! Tell me: How is the following long word broken ’pneumonoultramicroscopicsilicovolcanoconiosis’? I am sure that you are an expert in hyphenation, right?
Hi! TeX! Tell me: How is the following long word broken ’pneumonoultramicroscopicsilicovolcanoconiosis’? I am sure that you are an expert in hyphenation, right? ▯

The pass can build only three lines, so TeX executes for the second paragraph an emergency pass as the `\emergencystretch` is positive. The next example specifies a positive value for `\emergencystretch` but no emergency pass is executed.

Example 8: TeX input

```
\tracingparagraphs=1 \emergencystretch=4.5pt

\looseness=1
This is a short paragraph and two words can
have a hyphen in it. The rest of the text
is made up of short words only. Well, I
think the first sentence is wrong. Wait
then one more must be wrong. Two are wrong.
```

TeX output

This is a short paragraph and two words can have a hyphen in it. The rest of the text is made up of short words only. Well, I think the first sentence is wrong. Wait then one more must be wrong. Two are wrong. ▯

As the trace data proves in line 17 the line-breaking algorithm is successful in the first pass. It creates a paragraph of four lines. In order to increase this number TeX performs a second pass.

Example 8 continued: Log file contents

```
 1. @firstpass
 2. []\ninerm This is a short paragraph and two
      words can have
 3. @ via @@0 b=0 p=0 d=100
...
16. Wait then one more must be wrong. Two are
      wrong.
17. @\par via @@4 b=0 p=-10000 d=100
18. @@5: line 4.2- t=2509 -> @@4
19. @secondpass
20. []\ninerm This is a short para-graph and two
      words can have
21. @ via @@0 b=0 p=0 d=100
...
39. Wait then one more must be wrong. Two are
40. @ via @@5 b=12 p=0 d=10484
41. @@7: line 4.2 t=60914 -> @@5
42. wrong.
43. @\par via @@6 b=0 p=-10000 d=100
44. @@8: line 4.2- t=2509 -> @@6
45. @\par via @@7 b=0 p=-10000 d=100
46. @@9: line 5.2- t=61014 -> @@7
```
▯

No information in the trace data. The information in the trace of example 1 that the paragraph could be typeset with six instead of seven lines was part of the construction of the example. Here are some examples which demonstrate that this is not always reported. The first example uses a negative value for `\looseness`.

Example 9: TeX input

```
\tracingparagraphs=1

This is a short paragraph and two words can
have a hyphen in it. The rest of the text
is made up of short words only. I think the
first sentence is wrong. Wait then the next
one must be wrong too. Two are wrong, or?
```

TeX output

This is a short paragraph and two words can have a hyphen in it. The rest of the text is made up of short words only. I think the first sentence is wrong. Wait then the next one must be wrong too. Two are wrong, or? ▯

The line-breaking algorithm finds seven feasible breakpoints and the reported breaks are for a paragraph of five lines. It is not reported that the paragraph can be set in four lines.

Example 9 continued: Log file contents

```
 1. @firstpass
 2. []\ninerm This is a short paragraph and two
      words can have
 3. @ via @@0 b=0 p=0 d=100
 4. @@1: line 1.2 t=100 -> @@0
 5. a
 6. @ via @@0 b=19 p=0 d=841
 7. @@2: line 1.3 t=841 -> @@0
 8. hyphen in it. The rest of the text is made
      up of short
 9. @ via @@1 b=0 p=0 d=100
10. @ via @@2 b=4 p=0 d=196
11. @@3: line 2.2 t=200 -> @@1
12. words only. I think the first sentence is
      wrong. Wait
13. @ via @@3 b=24 p=0 d=1156
14. @@4: line 3.1 t=1356 -> @@3
15. then
16. @ via @@3 b=89 p=0 d=9801
17. @@5: line 3.3 t=10001 -> @@3
18. the next one must be wrong too. Two are
      wrong,
19. @ via @@4 b=1 p=0 d=121
20. @@6: line 4.2 t=1477 -> @@4
21. or?
22. @\par via @@5 b=0 p=-10000 d=100
23. @\par via @@6 b=0 p=-10000 d=100
24. @@7: line 5.2- t=1577 -> @@6
```

Nevertheless, the setting `\looseness=-1` succeeds in the first pass and a four line paragraph is output.

Example 9 continued: TEX input

```
\tracingparagraphs=1

\looseness=-1
This is a short paragraph and two words can ...
```

TEX output

This is a short paragraph and two words can have a hyphen in it. The rest of the text is made up of short words only. I think the first sentence is wrong. Wait then the next one must be wrong too. Two are wrong, or?

The log file contains only one additional line, the feasible breakpoint for a shorter paragraph in line 23.

Example 9 continued: Log file contents

```
 1. @firstpass
 2. []\ninerm This is a short paragraph and two
      words can have
 3. @ via @@0 b=0 p=0 d=100
...
21. or?
22. @\par via @@5 b=0 p=-10000 d=100
23. @@7: line 4.2- t=10101 -> @@5
24. @\par via @@6 b=0 p=-10000 d=100
25. @@8: line 5.2- t=1577 -> @@6
```

The next example sets the looseness parameter to 1, i.e., the number of lines for the paragraph should be one more than the optimum.

Example 10: TEX input

```
\tracingparagraphs=1

Let's look at another example. We saw that
$\root3\of8=2$ and $2^3=8$. What happens when 2
and 3 are switched? The equal sign is wrong! So
write $\root2\of8\neq3$ and $3^2\neq8$.

\looseness=1
Let's look at another example. We saw that ...
```

TEX output

Let's look at another example. We saw that $\sqrt[3]{8}=2$ and $2^3=8$. What happens when 2 and 3 are switched? The equal sign is wrong! So write $\sqrt[2]{8}\neq 3$ and $3^2\neq 8$.

Let's look at another example. We saw that $\sqrt[3]{8}=2$ and $2^3=8$. What happens when 2 and 3 are switched? The equal sign is wrong! So write $\sqrt[2]{8}\neq 3$ and $3^2\neq$ 8.

Again only one additional feasible breakpoint appears in the trace for the longer paragraph (see line 38). In both cases only the first pass is executed.

Example 10 continued: Log file contents

```
 1. @firstpass
 2. []\ninerm Let's look at another example. We
      saw that $[][] =
 3. @\penalty via @@0 b=0 p=500 d=250100
 4. @@1: line 1.2 t=250100 -> @@0
 5.  2$
 6. @\math via @@0 b=73 p=0 d=6889
 7. @@2: line 1.3 t=6889 -> @@0
 8.  and $2[] = 8$. What happens when 2 and 3
      are switched?
 9. @ via @@1 b=53 p=0 d=3969
10. @ via @@2 b=0 p=0 d=100
11. @@3: line 2.2 t=6989 -> @@2
12. The equal sign is wrong! So write $[][]
      \ninesy 6\ninerm = 3$ and $3[] \ninesy
      6\ninerm =
13. @\penalty via @@3 b=23 p=500 d=251089
14. @@4: line 3.1 t=258078 -> @@3
15.  8$.
16. @\par via @@3 b=0 p=-10000 d=100
17. @\par via @@4 b=0 p=-10000 d=100
18. @@5: line 3.2- t=7089 -> @@3
19.
20. @firstpass
...
35. @\par via @@3 b=0 p=-10000 d=100
36. @@5: line 3.2- t=7089 -> @@3
37. @\par via @@4 b=0 p=-10000 d=100
38. @@6: line 4.2- t=258178 -> @@4
```

If we want more information in the trace data, we have to find a way to have TEX report feasible breakpoints without setting `\looseness`. Unfortunately, this is not possible. A non-zero `\looseness` does two things: a) it changes the number of "easy" lines to TeX's maximum and b) it forces the execution of a slightly more complicated loop to find breakpoints. The code of this loop is shown in § 875 of [8] and it is only executed if the looseness parameter has a non-zero value (§ 873). But as both examples show, the possibility to shorten or lengthen a paragraph seems to be indirectly included in the end-of-par break candidates. The line number of the feasible breakpoints associated with an end-of-par break candidate can simply be increased by one and that value gives a possible alternative.

A digression. The change in item a) can be simulated and it has an interesting side effect: TeX might change the output of a paragraph with a non-zero `\looseness` even if the looseness command cannot be obeyed.

Example 11: TeX input

```
\tracingparagraphs=1 \pretolerance=-1

A one! Or two! Oh! A one! A two! A three!
It is a lovely day and I've got a feeling!
A new feeling! Yes it's a sunny day! Good
day! Sunshine! Sunshine! Sun! I'm in ---
hey, the text of the song sounds familiar.

\looseness=-1
A one! Or two! Oh! A one! A two! A three! ...
```

TeX output

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar.

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar. □

The line-breaking algorithm uses an internal counter to mark certain lines as "easy." TeX's algorithm saves space and time by the fact that after a certain point all lines have the same length ([8, § 818]) and this point is given by that counter. As stated in a), the counter is set to its maximum if `\looseness` is used. In example 2 we observed that the sequence of breakpoints in the trace output was changed compared to example 1. This effect has to do with the counter for easy lines (see [8, § 819]). As TeX picks the first break candidate that minimizes the total demerits the sequence is important.

The effect is seen not only when `\looseness` is non-zero, as the internal counter for easy lines is also set by `\hangindent` and `\parshape` (see [8, §§ 848–849]). A "neutral" `\parshape` specification — all lines have length `\hsize` and there are more lines than the paragraph will have — increases the counter for easy lines high enough to stimulate the same output as a non-zero value for `\looseness`. A five-line parshape outputs the paragraph in the style of the second paragraph above.

Example 12: TeX input

```
\def\fivelineparshape{\parshape 5 0pt \hsize
0pt \hsize 0pt \hsize 0pt \hsize 0pt \hsize }
\tracingparagraphs=1

A one! Or two! Oh! A one! A two! A three! ...

\fivelineparshape
A one! Or two! Oh! A one! A two! A three! ...
```

TeX output

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar.

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar. □

Let's look at the trace data. In this example two different sets of line breaks produce exactly the same total demerits, but in the first the line badnesses are 2, 1, 0, and in the second, 0, 2, 1.

Example 12 continued: Log file contents

```
 1. @firstpass
 2. []\ninerm A one! Or two! Oh! A one! A two!
      A three!
 3. @ via @@0 b=57 p=0 d=4489
 4. @@1: line 1.1 t=4489 -> @@0
 5. It
 6. @ via @@0 b=7 p=0 d=289
 7. @@2: line 1.2 t=289 -> @@0
 8. is
 9. @ via @@0 b=0 p=0 d=100
10. @@3: line 1.2 t=100 -> @@0
11. a
12. @ via @@0 b=2 p=0 d=144
13. @@4: line 1.2 t=144 -> @@0
14. lovely day and I've got a feeling! A new
      feeling!
15. @ via @@1 b=1 p=0 d=121
16. @ via @@2 b=40 p=0 d=2500
17. @@5: line 2.1 t=2789 -> @@2
18. @@6: line 2.2 t=4610 -> @@1
19. Yes
20. @ via @@2 b=3 p=0 d=169
21. @ via @@3 b=2 p=0 d=144
22. @ via @@4 b=25 p=0 d=1225
23. @@7: line 2.1 t=1369 -> @@4
24. @@8: line 2.2 t=244 -> @@3
25. it's
26. @ via @@3 b=64 p=0 d=5476
27. @ via @@4 b=1 p=0 d=121
28. @@9: line 2.2 t=265 -> @@4
29. @@10: line 2.3 t=5576 -> @@3
30. a
31. @ via @@4 b=64 p=0 d=5476
32. @@11: line 2.3 t=5620 -> @@4
33. sunny day! Good day! Sunshine! Sunshine!
34. @ via @@5 b=8 p=0 d=324
35. @ via @@6 b=8 p=0 d=324
```

```
36. @@12: line 3.2 t=3113 -> @@5
37. Sun!
38. @ via @@7 b=1 p=0 d=121
39. @ via @@8 b=1 p=0 d=121
40. @ via @@9 b=62 p=0 d=5184
41. @ via @@10 b=62 p=0 d=15184
42. @@13: line 3.1 t=5449 -> @@9
43. @@14: line 3.2 t=365 -> @@8
44. I'm
45. @ via @@9 b=0 p=0 d=100
46. @ via @@10 b=0 p=0 d=100
47. @ via @@11 b=5 p=0 d=225
48. @@15: line 3.2 t=365 -> @@9
49. in
50. @ via @@11 b=1 p=0 d=121
51. @@16: line 3.2 t=5741 -> @@11
52. --- hey, the text of the song sounds
     familiar.
53. @\par via @@12 b=2 p=-10000 d=144
54. @\par via @@13 b=0 p=-10000 d=100
55. @\par via @@14 b=0 p=-10000 d=100
56. @\par via @@15 b=0 p=-10000 d=100
57. @\par via @@16 b=0 p=-10000 d=100
58. @@17: line 4.2- t=465 -> @@15
59.
60. @firstpass
61. []\ninerm A one! Or two! Oh! A one! A two!
     A three!
62. @ via @@0 b=57 p=0 d=4489
63. @@1: line 1.1 t=4489 -> @@0
64. It
...
74. @ via @@2 b=40 p=0 d=2500
75. @ via @@1 b=1 p=0 d=121
76. @@5: line 2.1 t=2789 -> @@2
77. @@6: line 2.2 t=4610 -> @@1
78. Yes
...
112. @\par via @@16 b=0 p=-10000 d=100
113. @\par via @@15 b=0 p=-10000 d=100
114. @\par via @@13 b=0 p=-10000 d=100
115. @\par via @@14 b=0 p=-10000 d=100
116. @\par via @@12 b=2 p=-10000 d=144
117. @@17: line 4.2- t=465 -> @@14
```

▯

Look at lines 53–58 and 112–117: the best final feasible breakpoints select different previous feasible breakpoints, as the order of the break candidates is not the same. In other places of the trace this happens too but without consequence.

References

[1] Martin Budaj, `findhyph`, V3.4, 18.10.2015 `http://ctan.org/pkg/findhyph`

[2] *The Chicago Manual of Style*, 15th edition, Chicago, Illinois: University of Chicago Press, 2003

[3] Patrick Gundlach, `showhyphens`, V0.5c, 19.02.2016 `http://ctan.org/pkg/showhyphens`

[4] Patrick Gundlach, `lua-check-hyphen`, V0.4, 02.04.2016 `http://ctan.org/pkg/lua-check-hyphen`

[5] Donald E. Knuth and Michael F. Plass, "Breaking paragraphs into lines," *Software — Practice and Experience* **11** (1981), 1119–1184; reprinted with an addendum as Chapter 3 in [13], 67–155

[6] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984

[7] Donald E. Knuth, "A torture test for TEX," Stanford Computer Science Report *STAN-CS-84-1027*, Stanford, California: Stanford University, 1984

[8] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986

[9] Donald E. Knuth, *Computer Modern Typefaces*, Volume E of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986

[10] Donald E. Knuth, "The DVItype processor," in *TEXware*, Stanford Computer Science Report *STAN-CS-86-1097*, Stanford, California: Stanford University, 1986 (David R. Fuchs designed the first program, Peter Breitenlohner helped with the latest revisions) `http://ctan.org/pkg/dvitype`

[11] Donald E. Knuth, "Exercises for TEX: The Program", *TUGboat* **11**:2 (1990), 165–170; answers are given in: *TUGboat* **11**:4 (1990), 499–511; reprinted together as Chapter 10 in [13], 197–223 (exercise 28 is in the reprint exercise 25) `http://tug.org/TUGboat/tb11-2/tb28knut.pdf` `http://tug.org/TUGboat/tb11-4/tb30knut-exercises.pdf`

[12] Donald E. Knuth, "CSTUG, Charles University, Prague, March 1996: Questions and Answers with Prof. Donald E. Knuth," *TUGboat* **17**:4 (1996), 355–367; reprinted as Chapter 32 in [13], 601–624 `http://tug.org/TUGboat/tb17-4/tb53knuc.pdf`

[13] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999

[14] Frank Mittelbach, "`\looseness` on the loose," *TUGboat* **29**:2 (2008), 334; also published in *Die TEXnische Komödie* **19**:4 (2007), 41; and in: "Pearls of TEX programming," *TUGboat* **26**:3 (2005), 256–263, as "`\looseness` not so loose" (p. 259) `http://tug.org/TUGboat/tb29-2/tb92mitt.pdf`

[15] Udo Wermuth, "Typesetting the 'Begriffsschrift' by Gottlob Frege in plain TEX", *TUGboat* **36**:3 (2015), 243–256 `http://tug.org/TUGboat/tb36-3/tb114wermuth.pdf`

◇ Udo Wermuth
Babenhäuser Straße 6
63128 Dietzenbach
Germany
`u dot wermuth (at) icloud dot com`