le — A Text-line Extraction Program Based on the Area Voronoi Diagram MEMO

Koichi Kise
Dept. of Computer & Systems Sciences,
Osaka Prefecture University
`kise@cs.osakafu-u.ac.jp`

October 15, 1999

1 Changes

1. 991015
   - some minor modifications for this distribution.
2. 990126
   - the files `read_image.c` and `read_image.h` in le–981218.tgz and dl–981218.tgz were modified to fix the bug as follows: images whose width is not a multiple of 8 could not be correctly read.
   - The description of **4.2** in the memo was modified.
3. 981218
   - first version

2 Files

The archive `le.tgz` contains the following files.

`Makefile`	makefile

`const.h`
`defs.h`
`extern.h`
`function.h`
`read_image.h`
	constants and default values of parameters.
	data types used in the program.
	declaration of global variables.
	prototype declaration of some functions.
	header file for read_image.c

`main.c`	main routine for the program `le`
`read_image.c`	utility functions for reading a binary image. Sun rasterfiles and TIFF files are supported.

`img_to_site.c`
 	
  `2dch.c`
  `label_func.c`
  `sampling_points.c`
	extraction of generators (points) for Voronoi diagrams from an input binary image.
	extraction of convex hulls of connected components (CC's).
	functions for handling a labeled image.
	functions for recording sampling points on a CC.

`component.c`	utility functions for the record of CC's.

`voronoi.c`
  `heap.c`, `edgelist.c`
  `geometry.c`, `memory.c`
  `sites.c`, `output.c`
	construction of Voronoi diagrams.
	utility functions for voronoi.c.

`erase.c`
  `neighbor_graph.c`
  `ccomponent_stat.c`
  `seed.c`
	extraction of text-lines from the area Voronoi diagram.
	utility functions for the neighbor graph.
	utility functions for the record of seeds.
	functions for making seeds.

`chtl.c`	functions for generation of a convex hull of an extracted text-line as well as selection of edges in a convex hull.

`bit_func.c`
`cline.c`
`usage.c`
`hash.c`
`fout.c`
`dinfo.c`
	functions for handling binary images.
	command line analyzer.
	usage.
	functions for hash tables.
	functions for the output of intermediate results.
	functions for displaying information.

For further details of Sun rasterfile, see the manual page on a computer (`man rasterfile`).

3 Installation

3.1 Requirements

For the compilation of the program, it is required that the library `libtiff` has been installed in your computer.

3.2 Installation on a Linux machine

The program is developed under the Linux OS 2.0.36 (RedHat 5.2) with gcc Ver.2.7.2.3. If you have a Linux computer, the installation is as follows:

1. Extract the files from the archive by `tar xvzf le.tgz`.
2. In the directory which contains the files, just do `make`.

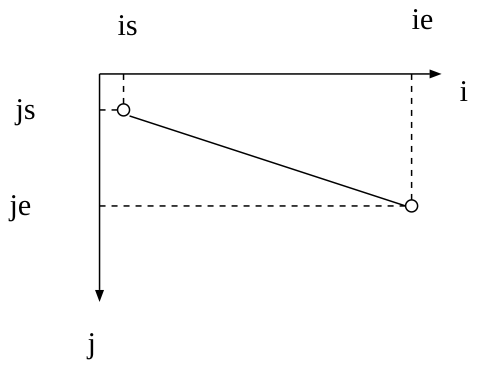

Figure 1: A line segment.

3. If the compilation completes, you can find `le` in the current directory.

3.3 Installation on machines with other OS's

We are sorry for not testing the program on machines with other OS's.

4 Usage

4.1 Requirements

The program requires at least 64MB of real memory to process an A4-size document image scanned with the resolution of 300dpi. We recommend that your computer has more than 128MB memory.

4.2 Input and output

The program `le` takes two mandatory arguments: input and output.

input The input is a binary document image whose format is either sun raster or TIFF.

output The output is a list of (i_s, i_e, j_s, j_e) (see Fig. 1) which represents a line segment in a text-line.

`le` also accepts optional arguments to change the default values of parameters, etc. The list of arguments and their brief descriptions are shown in Table 1. Note that, when you use the option `-points`, you obtain as output a list of points (i, j) instead of a list of line segments. You can find the details of these arguments in the next section.

Table 1: Optional arguments for `le`

argument	type	default	meaning
Parameters			
`-sr`	int.	`SAMPLE_RATE`	sampling rate of points on contours.
`-nm`	int.	`NOISE_MAX`	max. area of the convex hull of a noise CC.
`-af`	float	`ANGLE_FILTER`	threshold value of the variance of angles.
`-df`	float	`DIST_FILTER`	threshold value of the variance of distances.
`-sw`	int.	`SMWIND`	the size of the smoothing window for frequency distribution of d.
`-loop`	int.	`LOOPTIMES`	The number of times of iteration.
`-apt`	int.	`APTNBR`	The number of edges (N) selected in Select_N_Edge. (see Fig. 6.).
`-ta`	float	`T_AREA`	threshold value of the area ratio.
`-td`	float	`T_DIA`	threshold value of the diameter ratio.
`-ct`	float	`C_THETA`	The value of C_θ (see. Fig. 7.).
`-cd`	float	`C_DIST`	The value of C_d (see. Fig. 7.).
`-minnoe`	int	`MINNOE`	The minimum number of edges in a text-line.
`-dparam`	none	`NO`	If this option is supplied, the values of parameters used to run the program are listed.
Options for final results			
`-och`	none	`NO`	This changes the output to convex hulls of text-lines.
`-ichl`	none	`NO`	This changes the output to all edges in convex hulls of text-lines.
Options for intermediate results			
`-points`	none	`NO`	This changes the output to sample points on boundaries of CC's.
`-pvor`	none	`NO`	This changes the output to Voronoi edges of the point Voronoi diagram.
`-avor`	none	`NO`	This changes the output to Voronoi edges of the area Voronoi diagram.
`-ng`	none	`NO`	This changes the output to the neighbor graph.
`-seed`	none	`NO`	This changes the output to seeds.

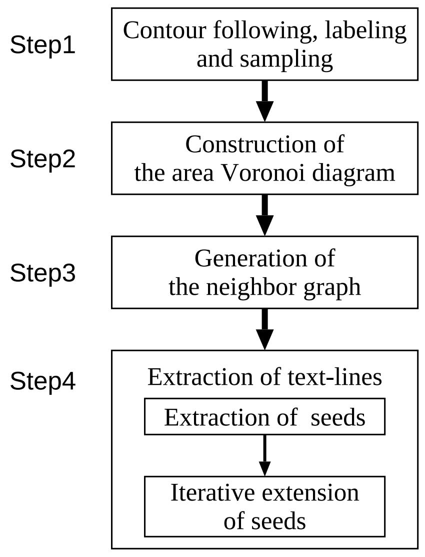
Step1
Contour following, labeling and sampling
Step2
Construction of the area Voronoi diagram
Step3
Generation of the neighbor graph
Step4
Extraction of text-lines
Extraction of seeds
Iterative extension of seeds

Figure 2: Steps of processing

5 Overview of the method

The method consists of 4 steps shown in Fig. 2. The description of the method can also be found in [1].

5.1 Contour following, labeling and sampling

The first step is to extract connected components and sample points on their contours both of which are based on contour following [1]. The default value of the sampling rate is represented as `SAMPLE_RATE` in the source code: every `SAMPLING_RATE`-th pixel on a contour of a connected component is stored as a generator for the point Voronoi diagram. In this step, connected components are discarded as noise if the area of the convex hull of their sample points is less than or equal to `NOISE_MAX`.

5.2 Construction of the area Voronoi diagram

The next step is to construct the area Voronoi diagram from the sample points and the connected components extracted in the previous step. The procedure is as follows:

1. Construct the point Voronoi diagram from the sample points.
2. Construct the area Voronoi diagram from the point Voronoi diagram by selecting Voronoi edges lying *between* connected components.

An example of the area Voronoi diagram is shown in Fig. 3.

5.3 Generation of the neighbor graph

The area Voronoi diagram represents adjacent (neighbor) relations between connected components. A connected component c_1 is a neighbor of a connected component c_2 if they share a Voronoi edge on their boundaries. The neighbor relations are represented by the neighbor graph generated in this step. An example is shown in Fig. 3.

[1]Note that this is the most time-consuming step in the method; you can speed up the processing if you replace the program with a more sophisticated one.

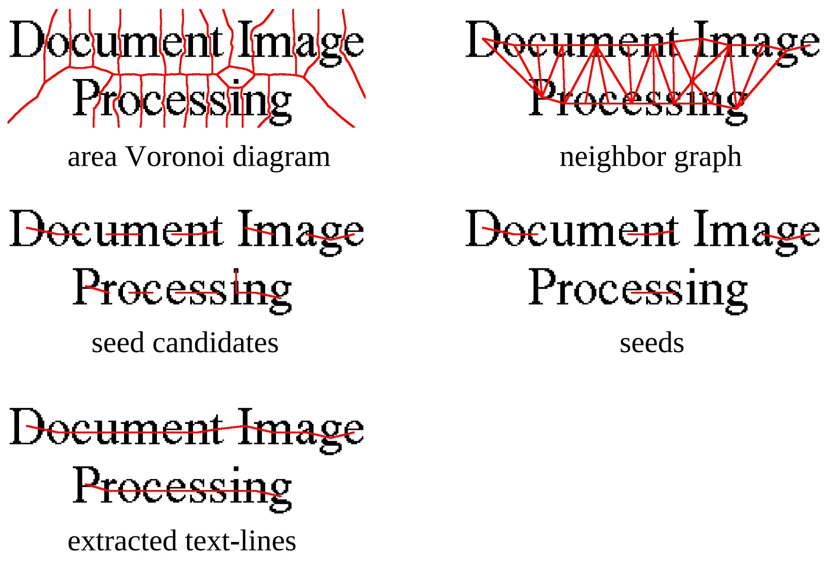
Document Image Processing
area Voronoi diagram
Document Image Processing
neighbor graph
Document Image Processing
seed candidates
Document Image Processing
seeds
Document Image Processing
extracted text-lines

Figure 3: Examples of the processing results.

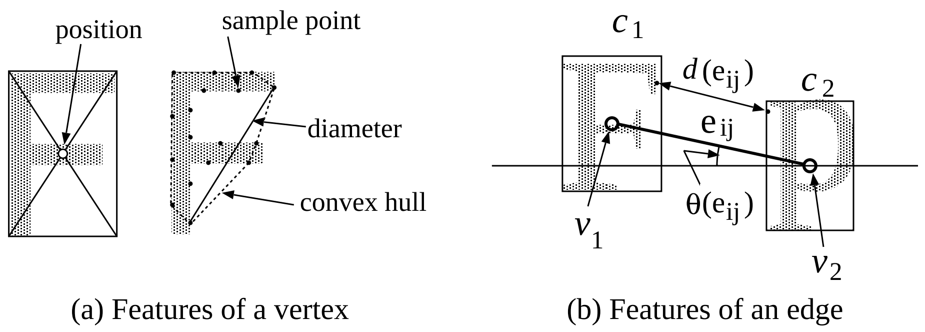

Figure 4: The features of a line segment.

The neighbor graph $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ is a graph in which a vertex $v \in V_\mathcal{G}$ corresponds to a connected component, and an edge $e \in E_\mathcal{G}$ represents a neighbor relation between two connected components. Note that text-lines in the image are represented as a *subgraph* of the neighbor graph on the assumption that every text-line consists of connected components which are neighbors with each other. We consider that this assumption is satisfied in the majority of documents. Thus the method attempts to find such a subgraph from the neighbor graph.

In order to extract a subgraph appropriate as text-lines from the neighbor graph, the neighbor graph is viewed as the following diagram (see Fig. 4.):

1. vertex $v \cdots$ a point representing a connected component. It has the following features:

   position The coordinates of the centroid of the minimum *bounding box* for a connected component. The sides of the bounding box are parallel to those of the image.

   area The area of the *convex hull* obtained from the sample points of a connected component $(a(v))$. The convex hull is introduced to reduce the influence on the area by the difference of fonts as well as the skew of an image.

   diameter The diameter $(D(v))$ of a convex hull, i.e., the Euclidean distance between the farthest pair of sample points of a connected component.

2. edge $e \cdots$ a line segment between adjacent centroids. It has the following features:

   distance the minimum distance $d(e)$ between adjacent *connected components* [2]. Let e be an edge between two adjacent connected components v_1 and v_2, and p_i and q_j be sample points on the contours of v_1 and v_2, respectively. Then, $d(e)$ is defined as:

   $$d(e) = \min_{i,j} d(p_i, q_j) \tag{1}$$

   where $d(p_i, q_j)$ indicates the Euclidean distance between points p_i and q_j.

   angle the angle ($\theta(e)$) of a line segment to the horizontal line.

5.4 Extraction of text-lines

The final step is the extraction of text-lines as a subgraph of the neighbor graph on the assumptions that:

A1 Every text-line is represented as a connected subgraph of the neighbor graph. To be precise, every text-line corresponds to an elementary path of the neighbor graph, i.e., $(v_1, e_{12}, v_2, ..., e_{m-1,m}, v_m)$ where $v_i \in V_{\mathcal{G}}$, $e_{ij} \in E_{\mathcal{G}}$, and $v_i \neq v_j$ if $i \neq j$.

A2 Every text-line consists of connected components of approximately equal size and shape,

A3 Every text-line is linear.

This step is divided into two substeps: extraction of seeds and extension of seeds. The procedures used in this step are shown in Appendix A. In Appendix A, merging of two graph $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ to produce the graph $G = \langle V_1 \cup V_2, E_1 \cup E_2 \rangle$ is represented as $G = G_1 \cup G_2$.

5.4.1 Extraction of seeds

First, we extract from the neighbor graph *seeds* which seem to be parts of text-lines. The procedure "Seed_Extraction" consists of two sub-procedures: "Edge_Grouping" and "Seed_Identification". The former is to extract candidates of seeds from the neighbor graph, and the latter is to select seeds from the candidates. Examples are shown in Fig. 3. In the following, candidates and seeds are represented as graphs $\mathcal{C}$ and $\mathcal{S}$, respectively.

In the procedure "Edge_Grouping", the threshold T_{ds} is utilized to reject edges with too large distance. Since the appropriate value of T_{ds} depends on the page layout of an input document, it is automatically set based on the frequency distribution of d. Fig. 5 shows a typical frequency distribution. The value of T_{ds} is set to the peak shown in Fig. 5, i.e., the estimated distance between text-lines. The procedure "Seed_Identification" selects candidates which satisfies the conditions described in the procedure.

5.4.2 Iterative extension of seeds

In this substep, the method extends seeds to obtain text-lines. The extension is not one-shot but iterative. Since seeds are short and thus with less statistical evidence at early stages of the iteration, they are extended carefully with a strict criterion. At later stages of the iteration, the criterion of the extension is relaxed to obtain complete text-lines. Examples of extracted text-lines are shown in Fig. 3.

The substep consists of seven procedures whose roles are summarized in Table 2.

In the top-level procedure "Iterative_Extension", seeds are extended with a criterion defined in the procedure "Acceptable" using the current number of times of iteration l. After the extension, the seeds which consist of a small number of edges are discarded.

In the procedure "Extend_Seed", both ends of a seed s are extended as much as possible. The procedure "Select_Edge" selects an edge to be merged with the seed, and the procedure "Merge_Edge"

[2]It is not the Euclidean distance between centroids.

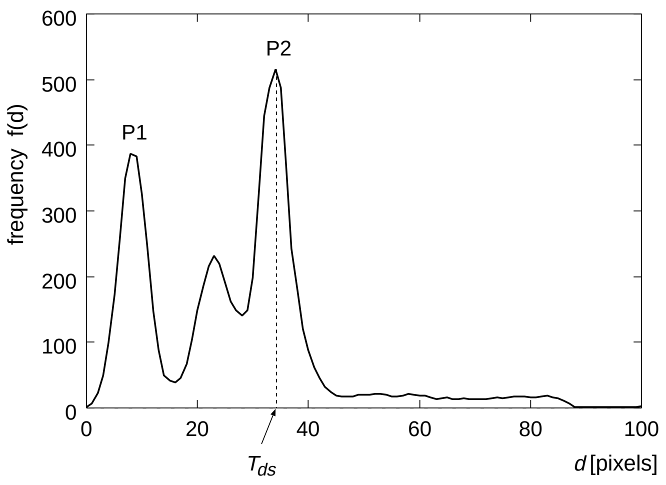

Figure 5: The value of T_{ds}.

Table 2: Overview of procedures

Proc.4	Iterative_Extension	the top-level procedure
Proc.5	Extend_Seed(s,l)	to extend a seed s at the l-th iteration
Proc.6	Select_Edge(s,v,l)	to select the best edge for the extension from a vertex v of a seed s
Proc.7	Select_N_Edge(s,v)	to select N edges as candidates for the extension from a vertex v of a seed s
Proc.8	Acceptable(e,v,s,l)	to check if an edge e is appropriate for the extension of a seed s. the current number of times of iteration l is utilized to change the criterion.
Proc.9	Merge_Edge(s,v,e)	to merge edge e at a vertex v of a seed s.
Proc.10	Update_Seeds(s_{old},s_{new})	to update the graph $\mathcal{S}$ with an updated seed s_{new} if $s_{\text{old}} \neq s_{\text{new}}$

merges it. The procedure “Update_Seeds” updates the current seeds in $\mathcal{S}$ as well as the features of seeds, when the seed s is actually updated.

The procedure “Select_Edge” selects an edge from N candidates obtained by the procedure “Select_N_Edge”. In the procedure “Select_N_Edge”, candidates are selected based on the difference of angles between an edge and a seed, as shown in Fig. 6.

The procedure “Acceptable” determines if an edge e is appropriate for the extension of a seed s. Let $\theta(s)$ be the angle of the line segment which connects both ends of a seed s to the horizontal line, and $d(s)$ be the average of the distance of edges in s. An edge e is appropriate if the following criterion is satisfied:

$$J(e,s,l) = \frac{\theta_e(e,s)}{\frac{l}{L}C_\theta} + \frac{d_e^2(e,s)}{C_d} \leq 1 \tag{2}$$

where $\theta_e(e,s)$ is the difference of angles:

$$\theta_e(e,s) = |\theta(e) - \theta(s)|, \tag{3}$$

$d_e^2(e,s)$ is the square difference of the minimum distances:

$$d_e^2(e,s) = (d(e) - d(s))^2, \tag{4}$$

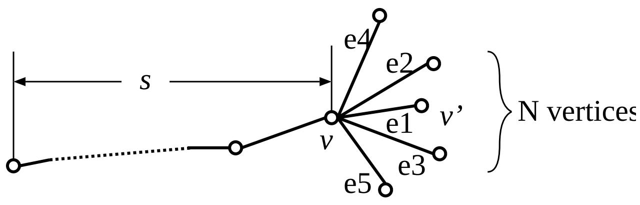

Figure 6: Select_N_Edge.

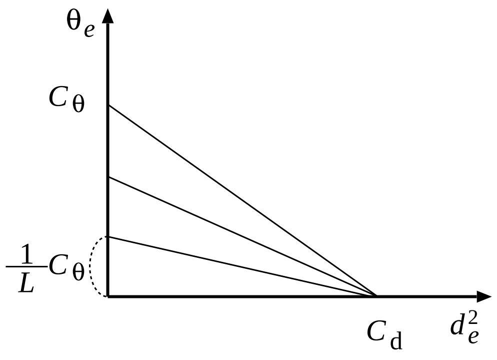

Figure 7: A criterion for the extension of a seed. A seed corresponds to a point in the d_e^2–θ_e plane.

L is the total number of times of iteration, C_θ = `C_THETA` and C_d = `C_DIST`. As shown in Fig. 7, the criterion varies according to the current number of times of iteration l. In this procedure, an input edge e for a seed s is tested whether Eq. (2) is satisfied in the case that e connects only with the seed s. Test from a different seed s' is additionally applied in the case that the edge e is between s and s', in order to avoid the situation shown in Fig. 8.

The procedure "Merge_Edge" simply merges an edge e with a seed s at a vertex v, if a vertex v' which is the other end of e is not contained in a different seed. If v' is in a different seed s', the procedure merges s, s' and e.

6 Limitations

Since the method is based on the area Voronoi diagram of connected components, there exist some limitations on the extraction of text-lines.

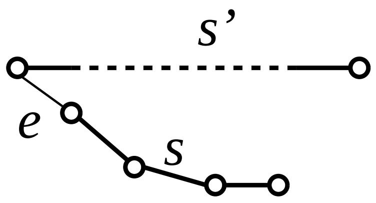

Figure 8: Need of a test with a different seed s'. The edge e is acceptable from s but not from s'.

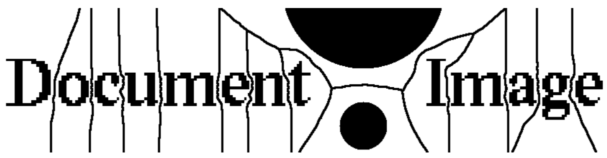

Figure 9: A text-line which cannot be represented as a subgraph of the neighbor graph.

- *The limitations caused by touching underlines, touching text-lines*: If a text-line touches with its underline, it cannot be successfully extracted; the program often rejects the text-line due to its large area. The extraction of a text-line also fails in the case that it touches with another text-line. A similar kind of situation can be caused by touching characters, if they form a large connected component.
- *The limitations caused by the definition of "neighbor" relations*: The neighbors of a connected component is defined based on the area Voronoi diagram. Since the method searches the neighbors of connected components to extract text-lines, there is no way to find the text-line whose parts have no neighbor relation.

  In the case shown in Fig. 9, for example, the text-line "Document Image" cannot be extracted, since no Voronoi edge is shared by "t" and "I".

7 Questions, Comments

If you have any questions or comments on the program, the algorithm and this note, please feel free to ask me. The e-mail address is:

`kise@cs.osakafu-u.ac.jp`

Bug reports are also welcomed.

References

[1] K.Kise, M.Iwata, A.Dengel and K.Matsumoto, A Computational Geometric Approach to Text-line Extraction from Binary Document Images, *Proc. of 3rd Workshop on Document Analysis Systems (DAS98)*, Nagano, Japan, pp.346-355, 1998.11

A Procedures of text-line extraction

A.1 Symbols

Table 3: Symbols

$\mathcal{G}\ \ (=\langle V_{\mathcal{G}}, E_{\mathcal{G}}\rangle)$	the neighbor graph
$\mathcal{C}\ \ (=\langle V_{\mathcal{C}}, E_{\mathcal{C}}\rangle)$	the graph in which each connected subgraph corresponds to a seed candidate
$\mathcal{S}\ \ (=\langle V_{\mathcal{S}}, E_{\mathcal{S}}\rangle)$	the graph in which each connected subgraph corresponds to a seed
$\mathcal{T}\ \ (=\langle V_{\mathcal{T}}, E_{\mathcal{T}}\rangle)$	the graph in which each connected subgraph corresponds to a text-line
e	an edge of the neighbor graph
v	a vertex of the neighbor graph
c	a seed candidate (a connected subgraph of $\mathcal{C}$)
s	a seed(a connected subgraph of $\mathcal{S}$)
$d(e)$	the minimum distance between CC's connected with an edge e (see Eq.(1))
$v_d(c)$	the variance of distances between CC's in a seed candidate c
$v_\theta(c)$	the variance of angles between CC's in a seed candidate c
$a(s)$	the average of $a(v)$ in a seed s
$d(s)$	the average of d of edges in a seed s
$\theta(s)$	the angle of the line segment between two ends of s
$D(s)$	the average of $D(v)$ in a seed s
$\theta_e(s, e)$	the difference of angles (see Eq.(3))
$J(e, s, l)$	the evaluation function of an edge e for a seed s at l-th iteration (see Eq.(2))
T_{ds}	a threshold of the distance (see Fig. 5)
N	the number of times of iteration

A.2 Procedures

Procedure 1 Seed_Extraction
```
1  Edge_Grouping
2  Seed_Identification
```

Procedure 2 Edge_Grouping
```
1  let $(e_1, ..., e_n)$ be the sorted list of edges in $\mathcal{G}$, where $\forall i\ d(e_i) \leq d(e_{i+1})$.
2  $\mathcal{C} \leftarrow \langle \emptyset, \emptyset \rangle$
3  $i \leftarrow 1$
4  while $d(e_i) \leq T_{ds}$ do
   ▷ $T_{ds}$ is a threshold not to select edges with too large distance.
5      let $v_{i1}$ and $v_{i2}$ be the vertices connected by an edge $e_i$ in $G$.
6      if $v_{i1} \notin V_{\mathcal{C}}$ or $v_{i2} \notin V_{\mathcal{C}}$ then
           ▷ This guarantees that a seed candidate contains no loop.
7          $\mathcal{C} \leftarrow \mathcal{C} \cup \langle \{v_{i1}, v_{i2}\}, \{e_i\} \rangle$
8      end if
9      $i \leftarrow i + 1$
10 end
```

Procedure 3 Seed_Identification
```
1  $\mathcal{S} \leftarrow \langle \emptyset, \emptyset \rangle$
2  foreach seed candidate $c$ in the graph $\mathcal{C}$ do
   ▷ $c$ is a connected subgraph in $\mathcal{C}$.
3      if $c$ consists of more than a single edge and
4          $c$ does not include a branch and
5          $v_{\theta}(c) \leq$ ANGLE_FILTER and
           ▷ Connected components in a seed should be arranged approximately linear.
6          $v_d(c) \leq$ DIST_FILTER
           ▷ Connected components in a seed should be arranged at approximately even intervals.
7      then
8          $\mathcal{S} \leftarrow c \cup \mathcal{S}$
9      end if
10 end
```

Procedure 4 Iterative_Extension
```
1  $\mathcal{T} \leftarrow \langle \emptyset, \emptyset \rangle$
2  for $l = 1$ to $L(=$ LOOPTIMES$)$ do
3      foreach seed $s$ in the graph $\mathcal{S}$ do
       ▷ $s$ is a connected subgraph in $\mathcal{S}$
4          Extend_Seed($s$,$l$)
           ▷ Extend a seed $s$ as much as possible with a criterion depending on $l$.
5      end
6  end
7  foreach seed $s$ in $\mathcal{S}$ do
8      $n \leftarrow$ the number of edges in $s$
9      if $n \geq$ MINNOE then
       ▷ Only the seeds with more than or equal to MINNOR edges are identified as text-lines.
           $\mathcal{T} \leftarrow s \cup \mathcal{T}$
10     end if
11 end
```

Procedure 5 Extend_Seed(s,l)

```
1  if the seed $s$ exists in $\mathcal{S}$ then
   ▷ Unless the seed $s$ has been removed by Update_Seeds, do the following steps.
2      do
       ▷ Continue executing the following steps while the seed $s$ is updated.
3          let $v_1$ be a vertex at an end of $s$
4          let $v_2$ be a vertex at the other end of $s$
5          $s_{\text{old}} \leftarrow s$
6          $e_1 \leftarrow$ Select_Edge($s$,$v_1$,$l$)
7          $e_2 \leftarrow$ Select_Edge($s$,$v_2$,$l$)
8          $s \leftarrow$ Merge_Edge($s$,$v_1$,$e_1$)
9          $s \leftarrow$ Merge_Edge($s$,$v_2$,$e_2$)
10     while Update_Seeds($s_{\text{old}}$,$s$)
11 end if
```

Procedure 6 Select_Edge(s,v,l)

```
1  $(e_1, ..., e_N) \leftarrow$ Select_N_Edge($s$,$v$)
2  for $i = 1$ to $N$ do
3      if Acceptable($e_i$,$v$,$s$,$l$) then
4          return $e_i$
5      end if
6  end
7  return $\phi$
```

Procedure 7 Select_N_Edge(s,v)

```
   ▷ See Fig. 6.
1  $E \leftarrow \emptyset$
2  foreach edge $e (\in E_{\mathcal{G}})$ whose end is $v$ do
3      let $v' (\in V_{\mathcal{G}})$ be the other end of $e$ $(v' \neq v)$
4      if $v'$ is not contained in the seed $s$ and
5          $v'$ is at an end if it is a vertex of a different seed and
6          $R(a(s), a(v')) \geq$ T_AREA and
7          $R(D(s), D(e)) \geq$ T_DIA
           ▷ where $R(x, y) = \min(x, y)/\max(x, y)$
8      then
9          $E \leftarrow E \cup \{e\}$
10     end if
11 end
12 return the sorted list of edges $(e_1, ..., e_N)$ where
13     $e_i \in E$, $N \leq$ APTNBR, $\theta_e(e_i, s) \leq \theta_e(e_{i+1}, s)$
       ▷ Sort the edges in the ascending order of the difference of angles.
```

Procedure 8 Acceptable(e,v,s,l)

```
1  if $J(e, s, l) \leq 1$ then
2      let $v'$ be a vertex on the other end of $e$ ($v' \neq v$)
3      if $v'$ belongs to a different seed $s'$ then
4          if $e \in$ Select_N_Edge($s'$,$v'$) and
5          $J(e, s', l) \leq 1$ then
6              return true
7          else
8              return false
9          end if
10     else
11         return true
12     end if
13 else
14     return false
15 end if
```

Procedure 9 Merge_Edge(s,v,e)

```
1  if $e \neq \phi$ then
   ▷ If there is an edge to be merged:
2      let $v'$ be a vertex on the other end of $e$ ($v' \neq v$)
3      if $v'$ is in a different seed $s'$ then
4          return the updated seed $s \cup s' \cup \langle\{v, v'\}, \{e\}\rangle$
           ▷ The seeds $s$, $s'$ and the edge $e$ are merged.
5      else
6          return the updated seed $s \cup \langle\{v, v'\}, \{e\}\rangle$
           ▷ The seed $s$ and the edge $e$ are merged.
7      end if
8  else
   ▷ If the seed $s$ is not updated:
9      return $s$
10 end if
```

Procedure 10 Update_Seeds(s_{old},s_{new})

```
1  if $s_{\text{new}} \neq s_{\text{old}}$ then
   ▷ if the seed $s_{\text{new}}$ is updated:
2      foreach seed $s$ in $\mathcal{S}$ do
3          if $s$ shares some edges with $s_{\text{new}}$ then
4              Delete $s$ from $\mathcal{S}$
5          end if
6      end
7      $\mathcal{S} \leftarrow s_{\text{new}} \cup \mathcal{S}$
8      Recalculate the features of $s_{\text{new}}$ ($a(s_{\text{new}})$,$d(s_{\text{new}})$, $\theta(s_{\text{new}})$, $D(s_{\text{new}})$)
9      return true
10 else
11     return false
12 end if
```