Vario-scale topological data structures suitable for progressive transfer: the GAP-face tree and GAP-edge forest

Peter van Oosterom

March 3, 2009

Section GIS Technology

Delft University of Technology

Contents

1. Introduction

- Multi-scale databases: often multiple representation
  drawbacks: redundancy, fixed levels of detail
- Scaleless data structures: single representation with additional structure to access at any level of detail

- Often also spatial organization (clustering/indexing)
- Progressive transfer: keep sending more details (compare to raster formats: data pyramids, wavelets)

TU Delft

Contents

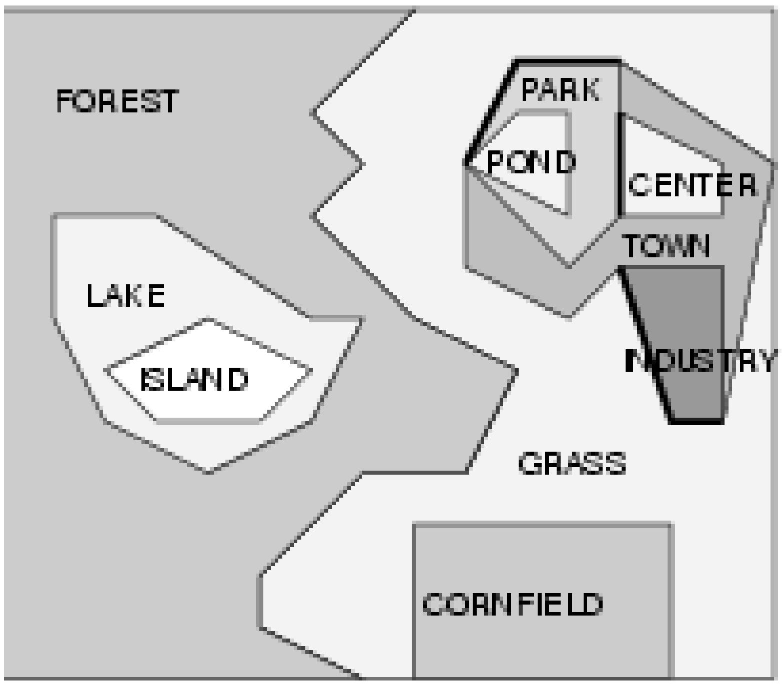
TU Delft

2. GAP-tree historic overview motivation

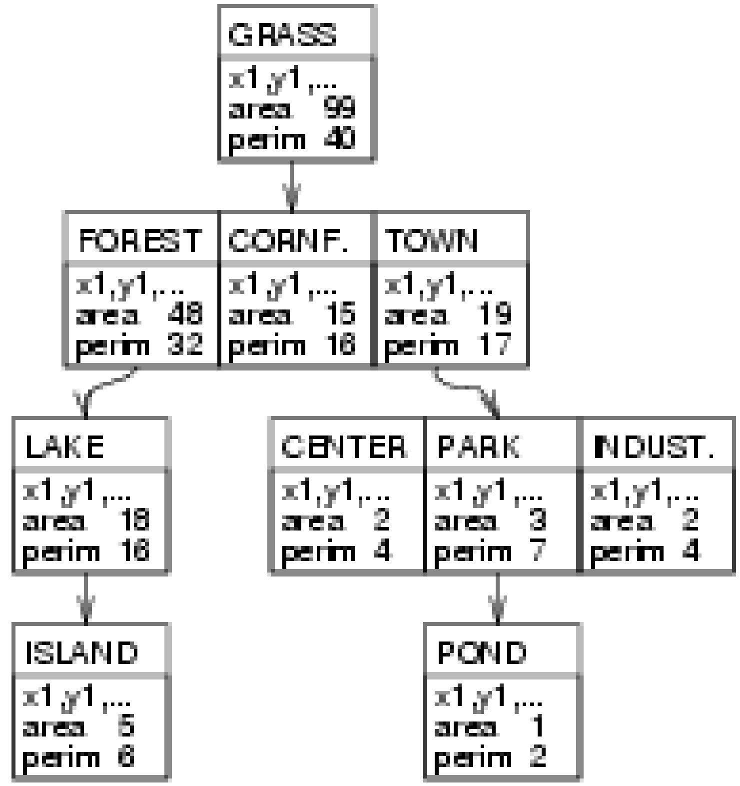
- Independent generalization of (boundaries of) two neighbor objects will result in small slivers (gaps, overlaps)
- The Generalized Area Partitioning (GAP)-tree does solve this (vO'93)
- GAP-tree can be used together with BLG-tree and Reactive-tree (each taking care of a different aspect of generalization: selection, aggregation, simplification,..)
- Several improvements published over time

2. GAP-tree historic overview the original concept

a.The scene



b.The GAP-tree



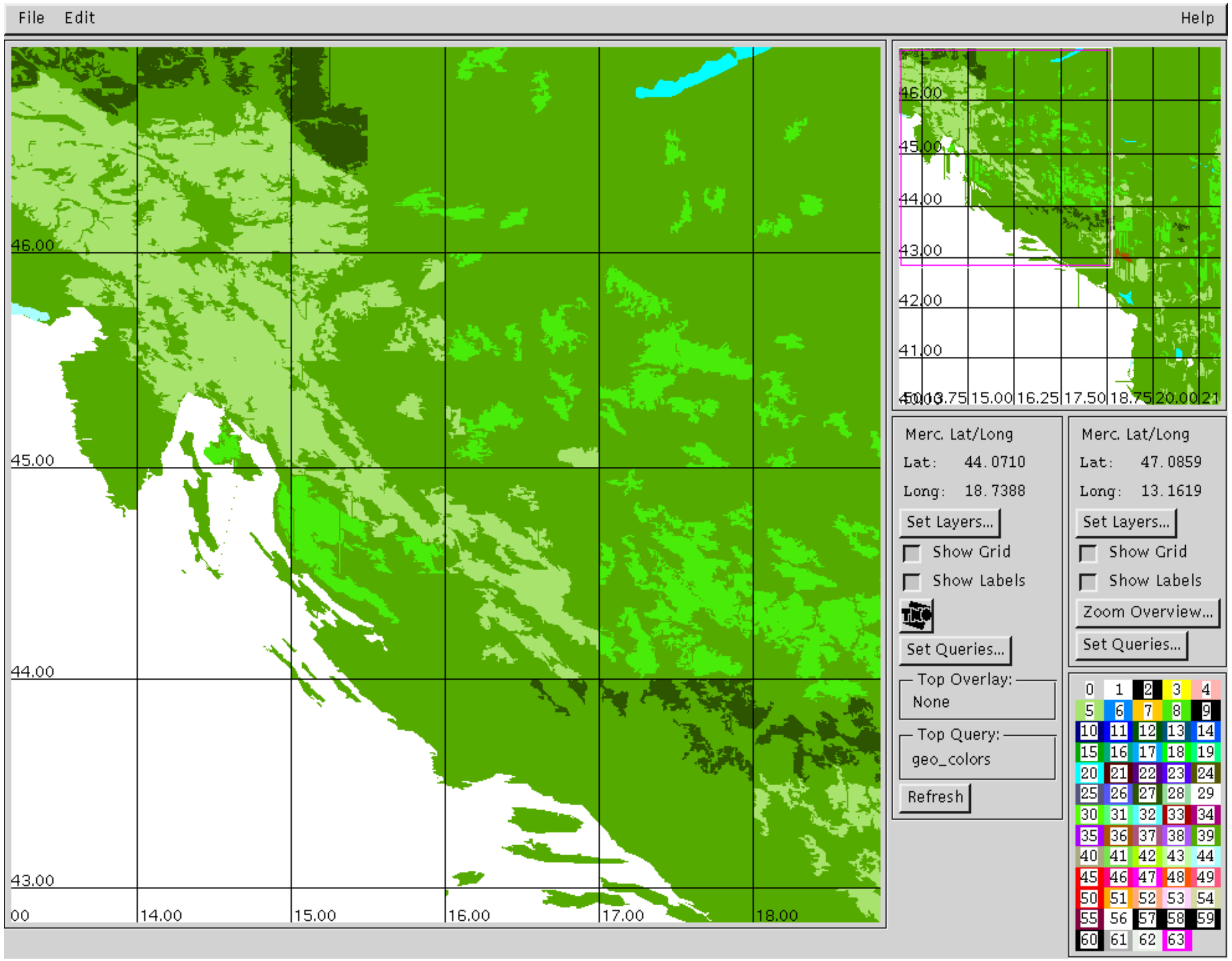
TU Delft

2. GAP-tree historic overview Constructing

- Find least important object, minimum value for Imp(a) = Area(a) * WeightClass(a)
- Find most compatible neighbor, maximum value for Collapse(a,b) = Length(a,b) * CompatibleClasses(a,b)
- Merge a in b (make link in GAP-tree), recompute Imp(b)
- Repeat steps above until one area left (root of tree)

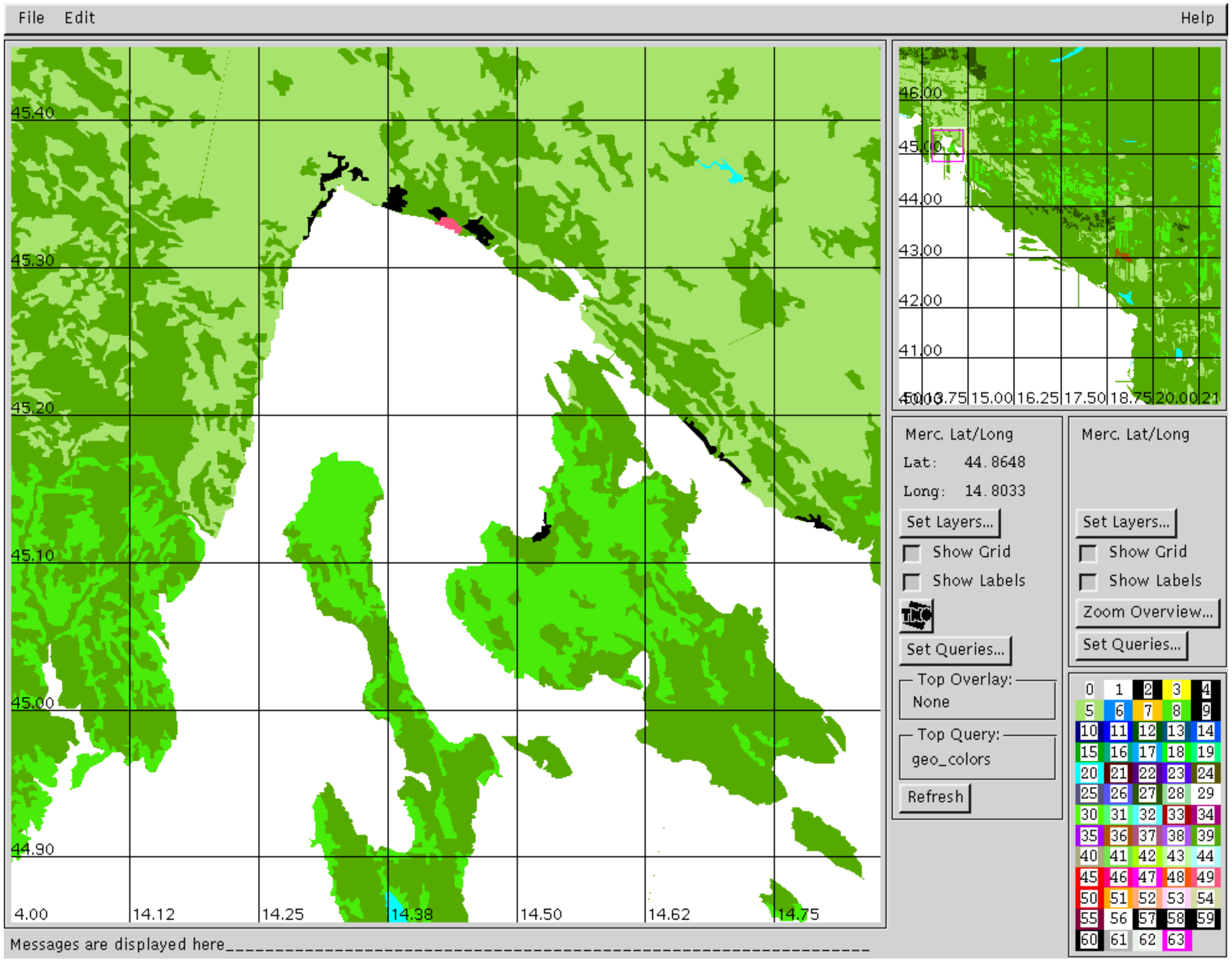
- Use of tree: start drawing top area, next visit relevant nodes (imp and bbox), draw on top (Painters algorithm)

Example 1: DLMS DFAD, scale change

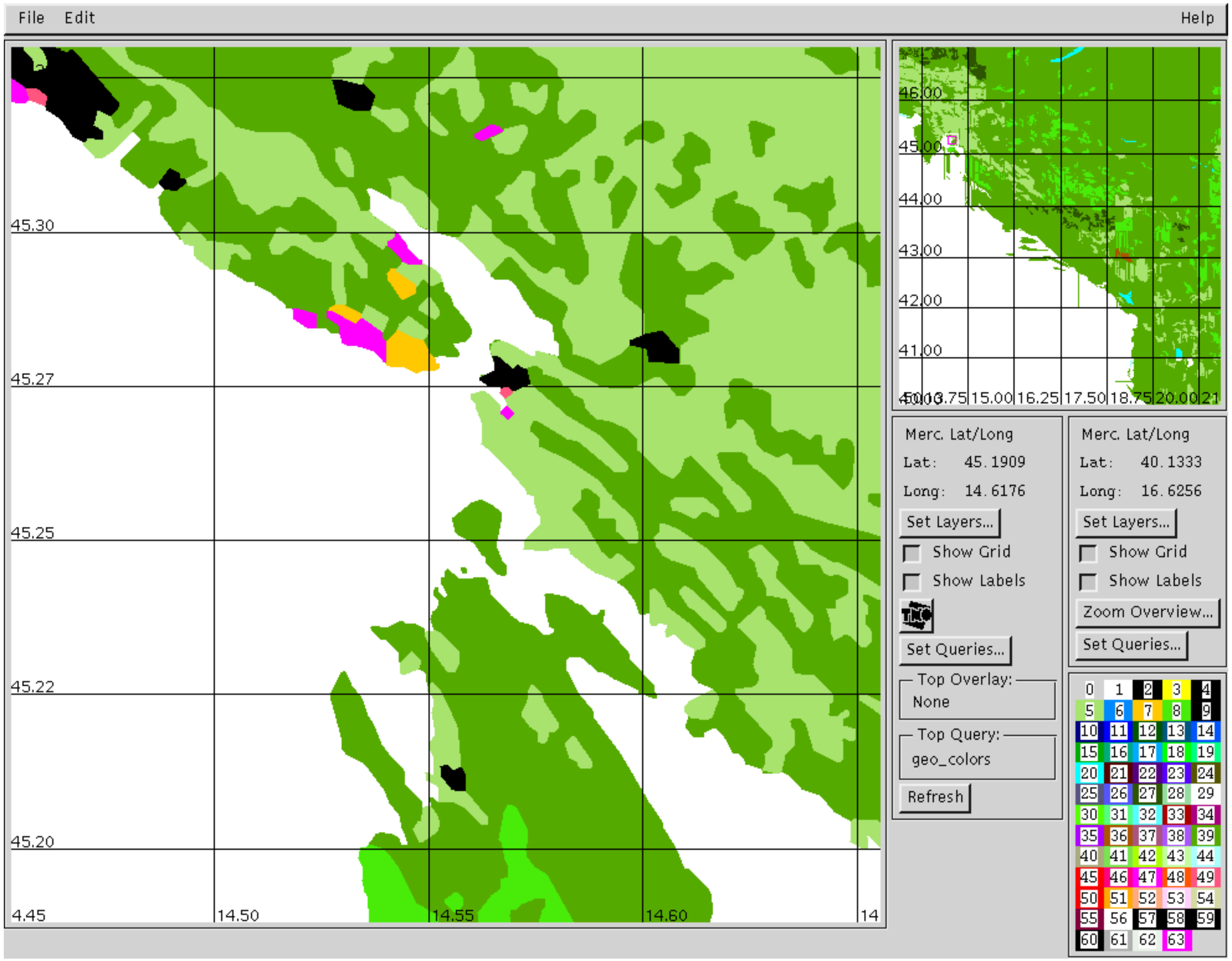


TU Delft

Example 1: DLMS DFAD, scale change

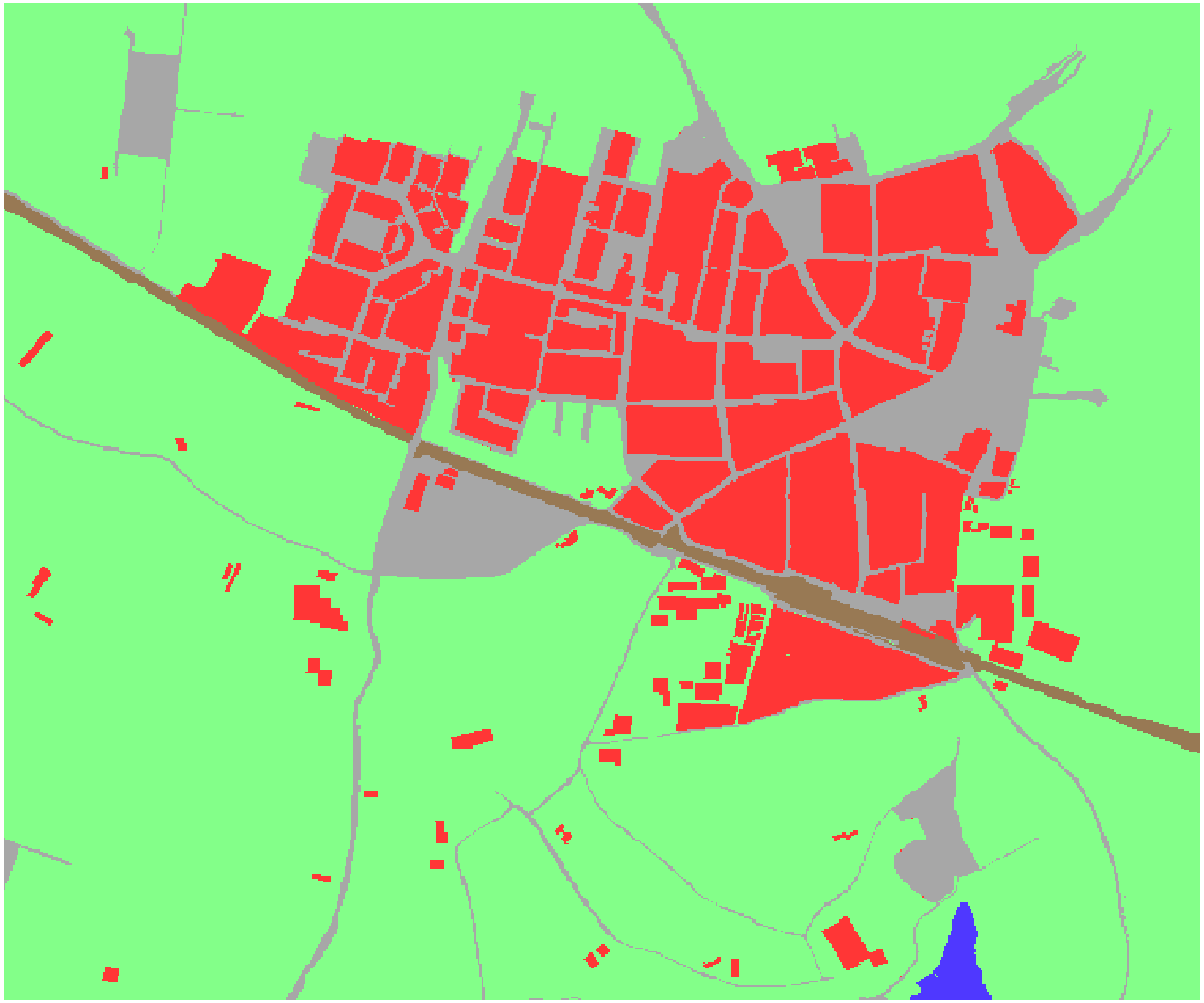


Example 1: DLMS DFAD, scale change

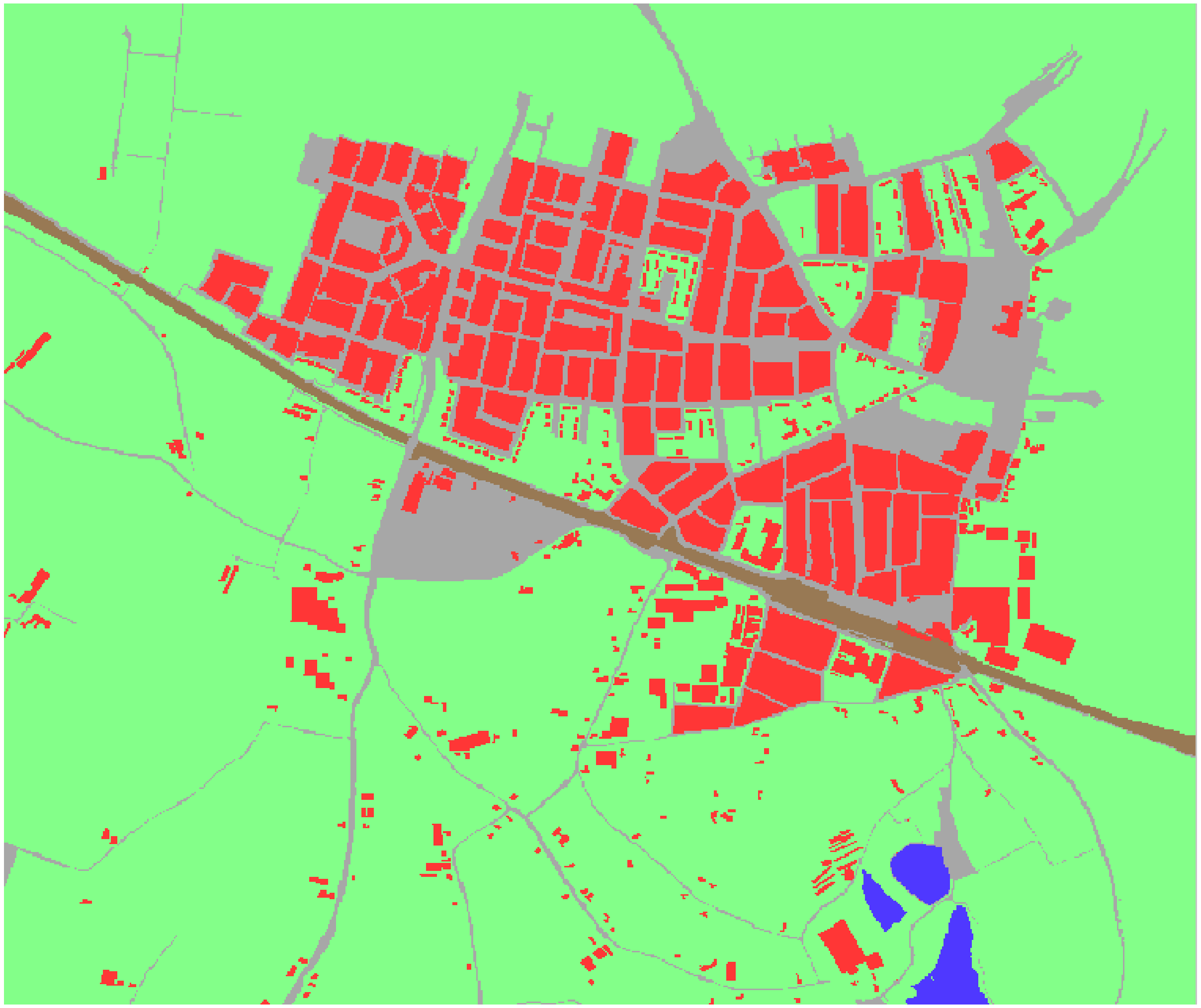


TU Delft

Example 2: GBKN, scale fixed

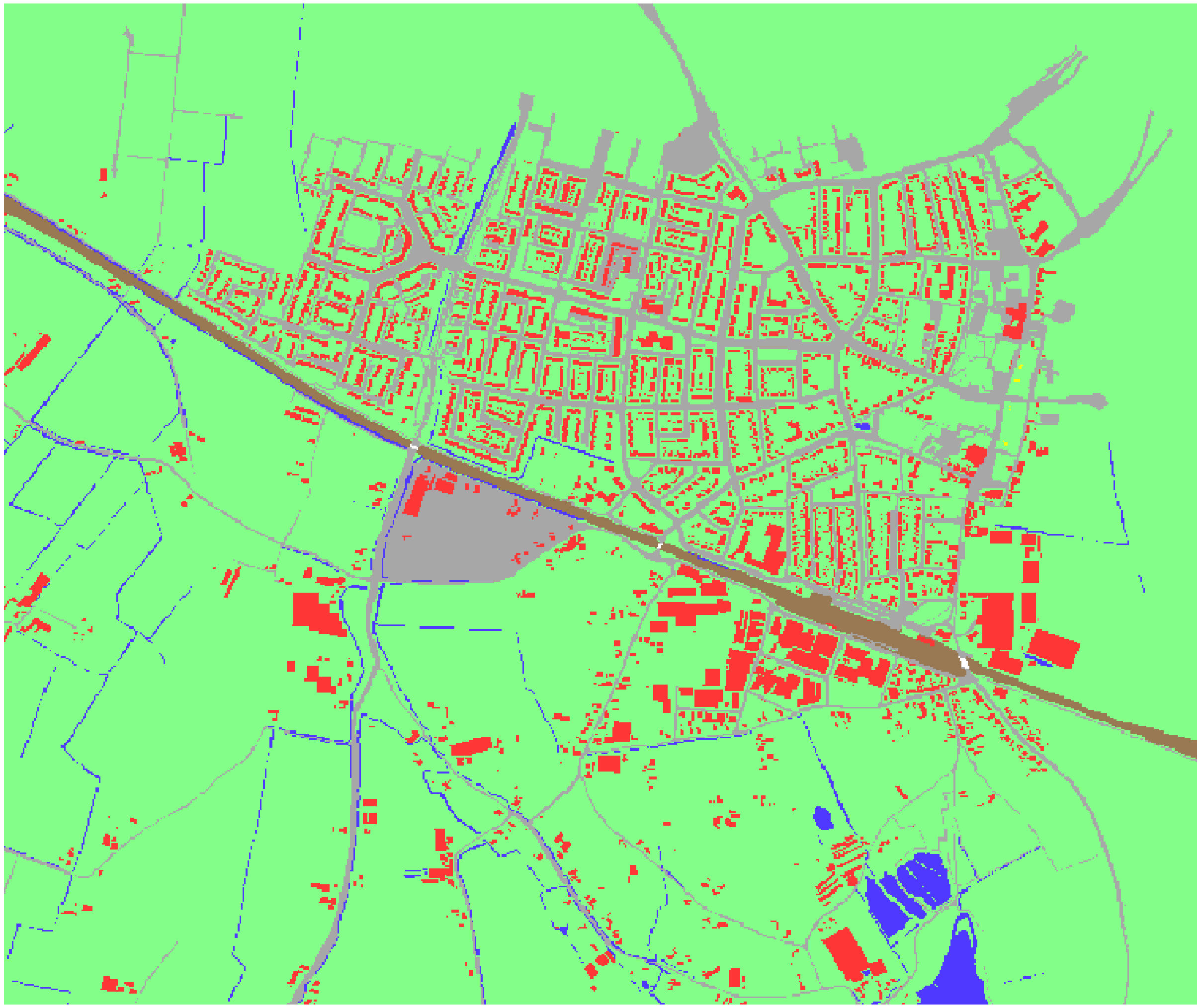


Example 2: GBKN, scale fixed



TUDelft

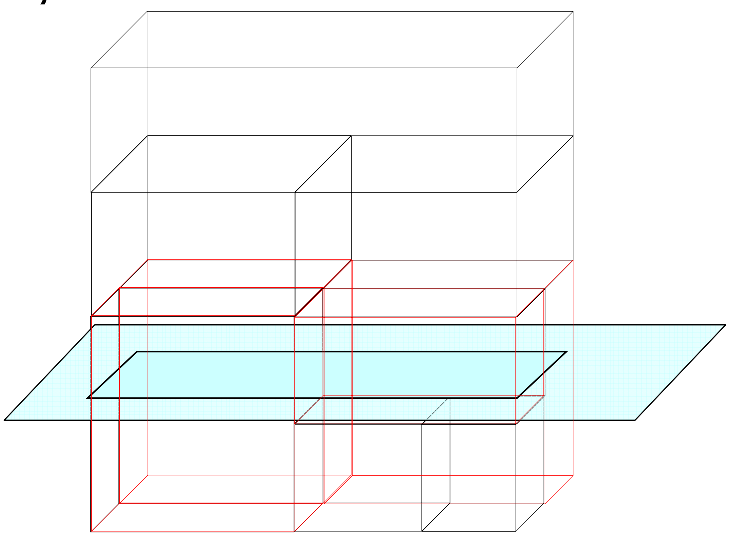
Example 2: GBKN, scale fixed



TU Delft

2. GAP-tree historic overview topological GAP-tree

- In normal GAP-tree areas are stored as independent polygons, drawback (computed) redundancy
- Vermeij et al.'03 proposed topological GAP-tree: edges and faces (with importance range, consider as height), reduced redundancy between neighbors
- Still some redundancy left: coordinates in higher level edge also present in lower (more detailed) level edges



TU Delft

Contents

1. Introduction
2. GAP-tree historic overview
3. Topological GAP-face tree/GAP-edge forest
4. Storage structure
5. Client-server progressive refinement
6. Conclusions

TU Delft

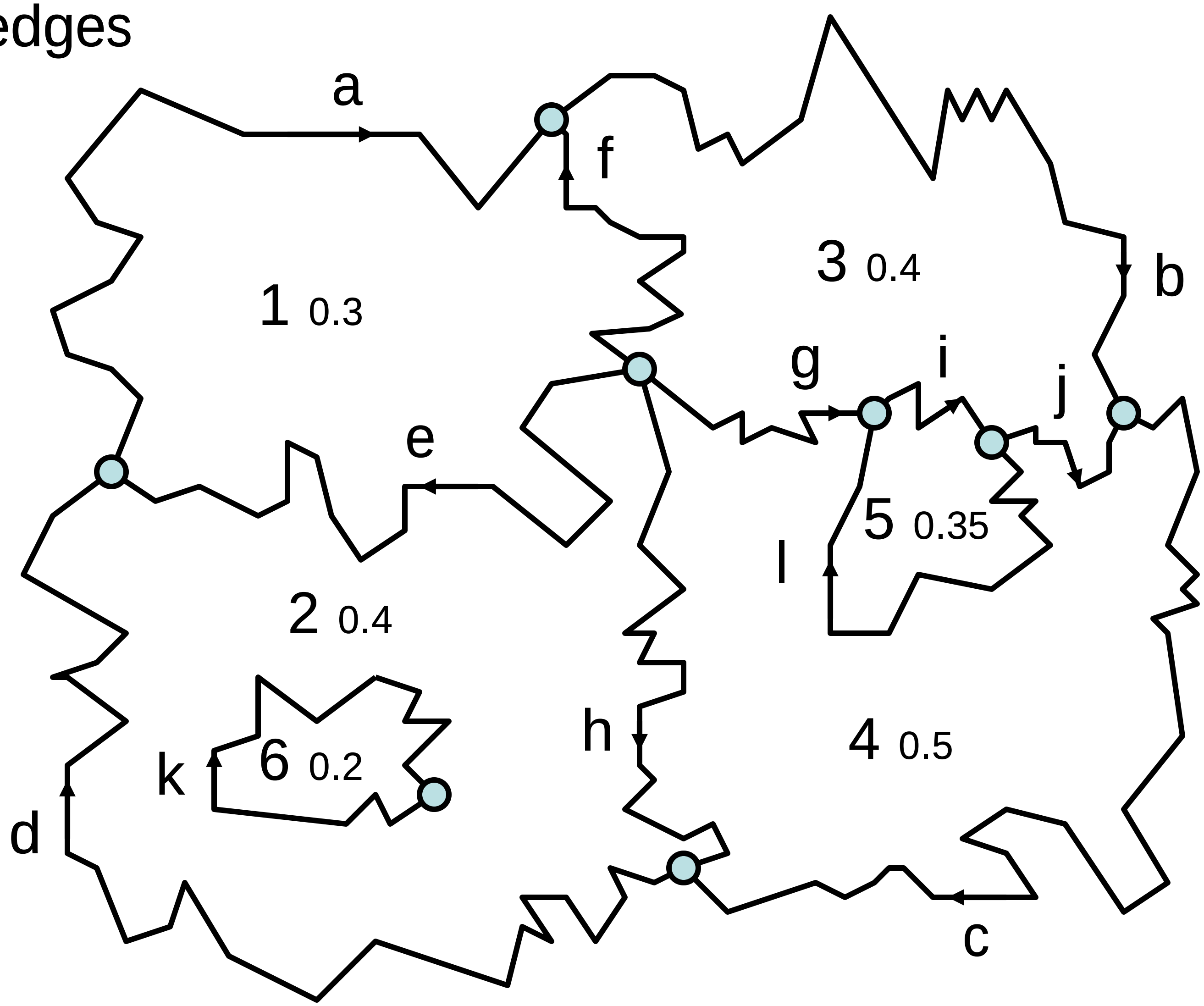
3. Topological GAP-face tree (GAP-edge forest)

- Also coordinate redundancy between edges at different aggregation levels is removed

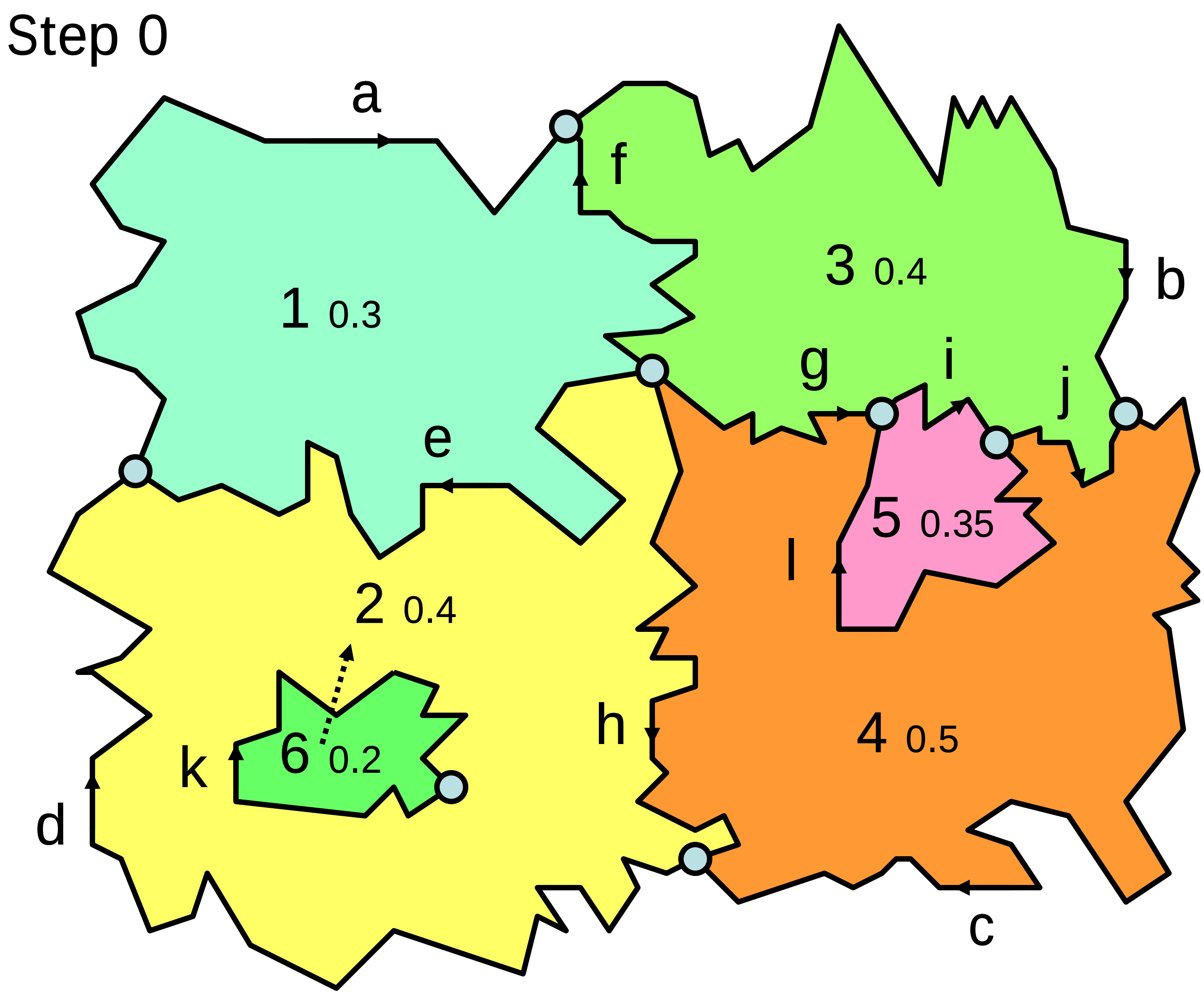
- Throughout remainder of presentation one example is used with edges and faces (and nodes) shown

- Creation of the tGAP-tree is shown in pairs of steps
  1. removal of least important face (merge face)
  2. removal of edges, merge of edges (BLG-tree)

edges

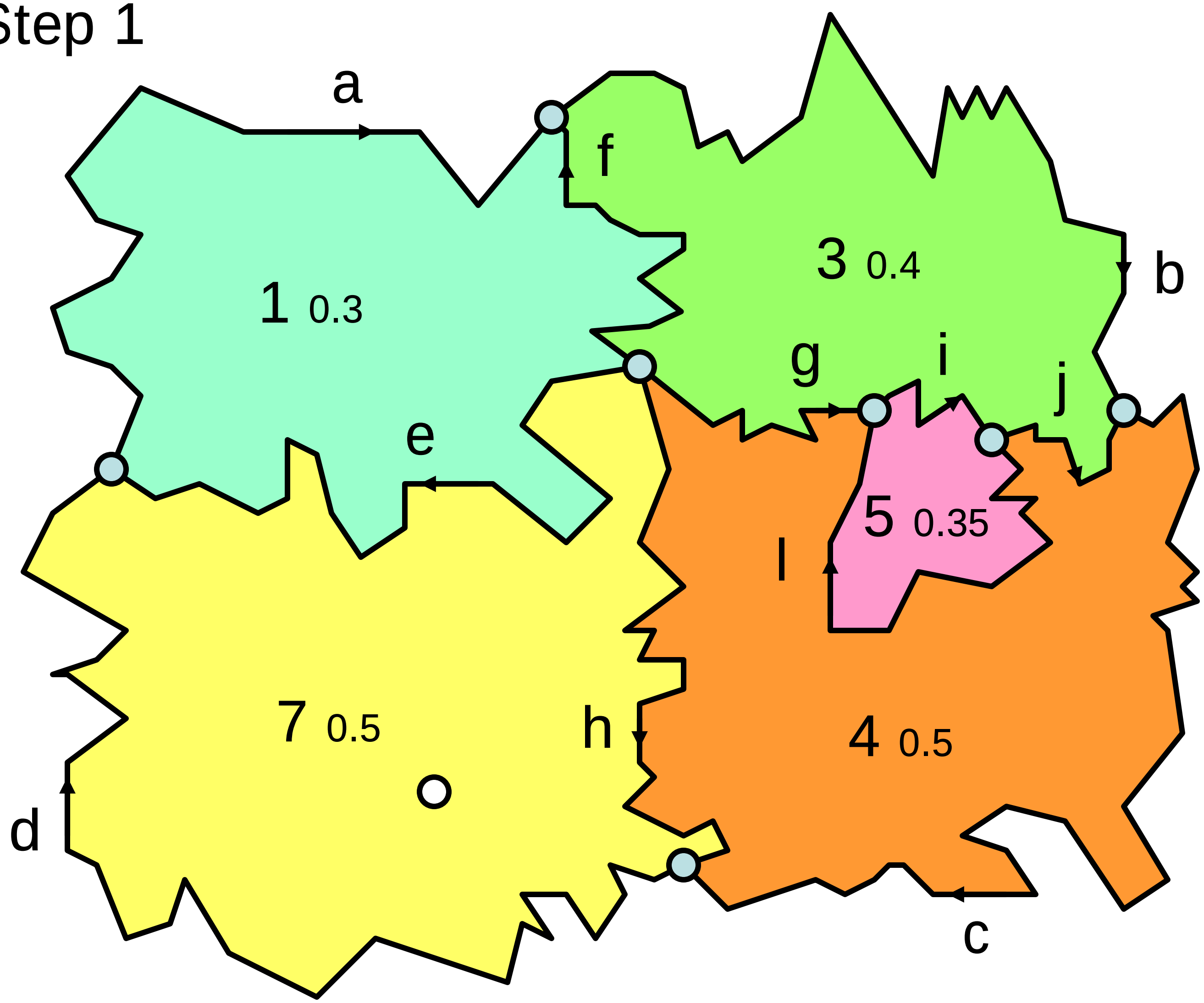


TU Delft

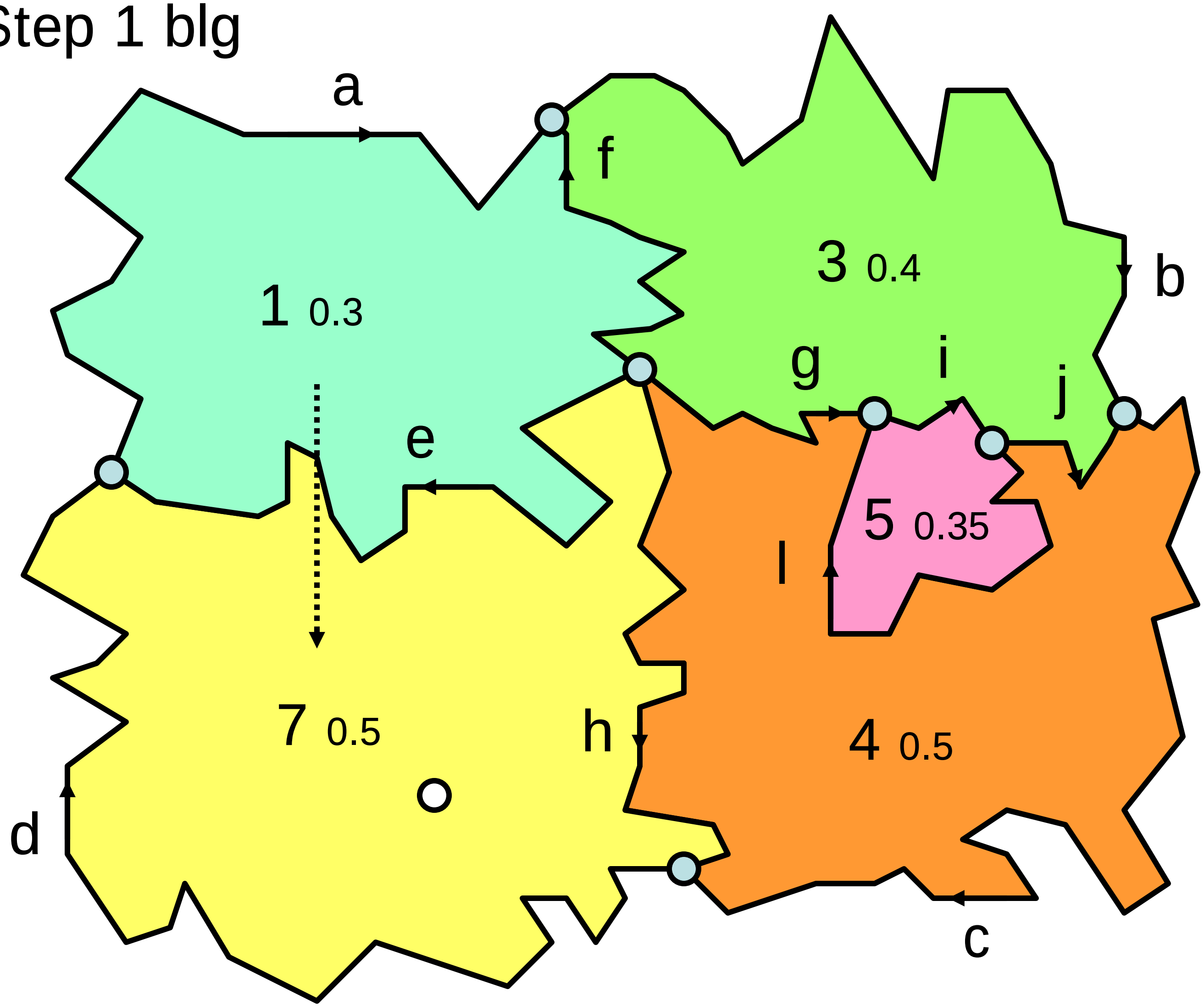


TU Delft

Step 1

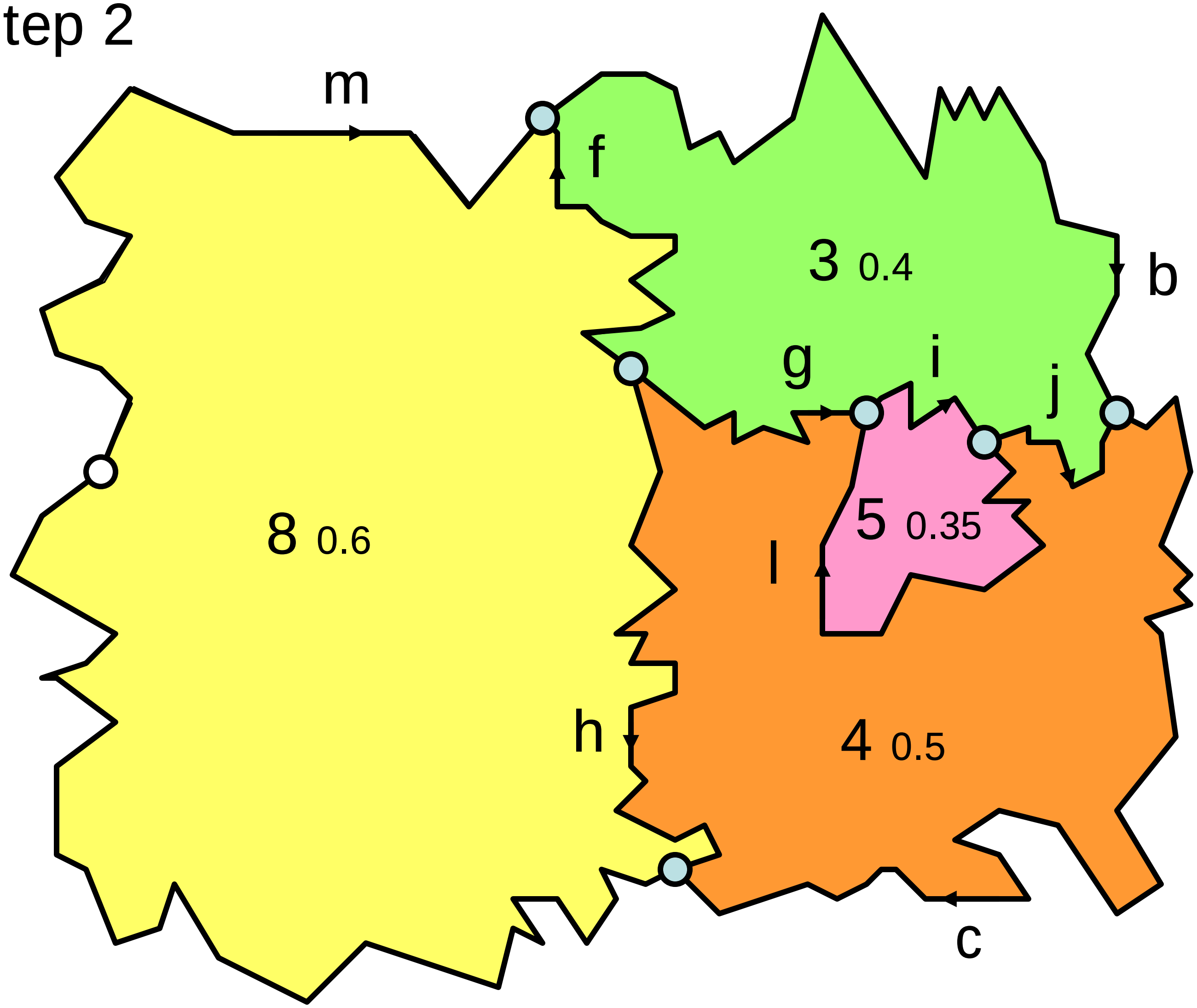


Step 1 blg

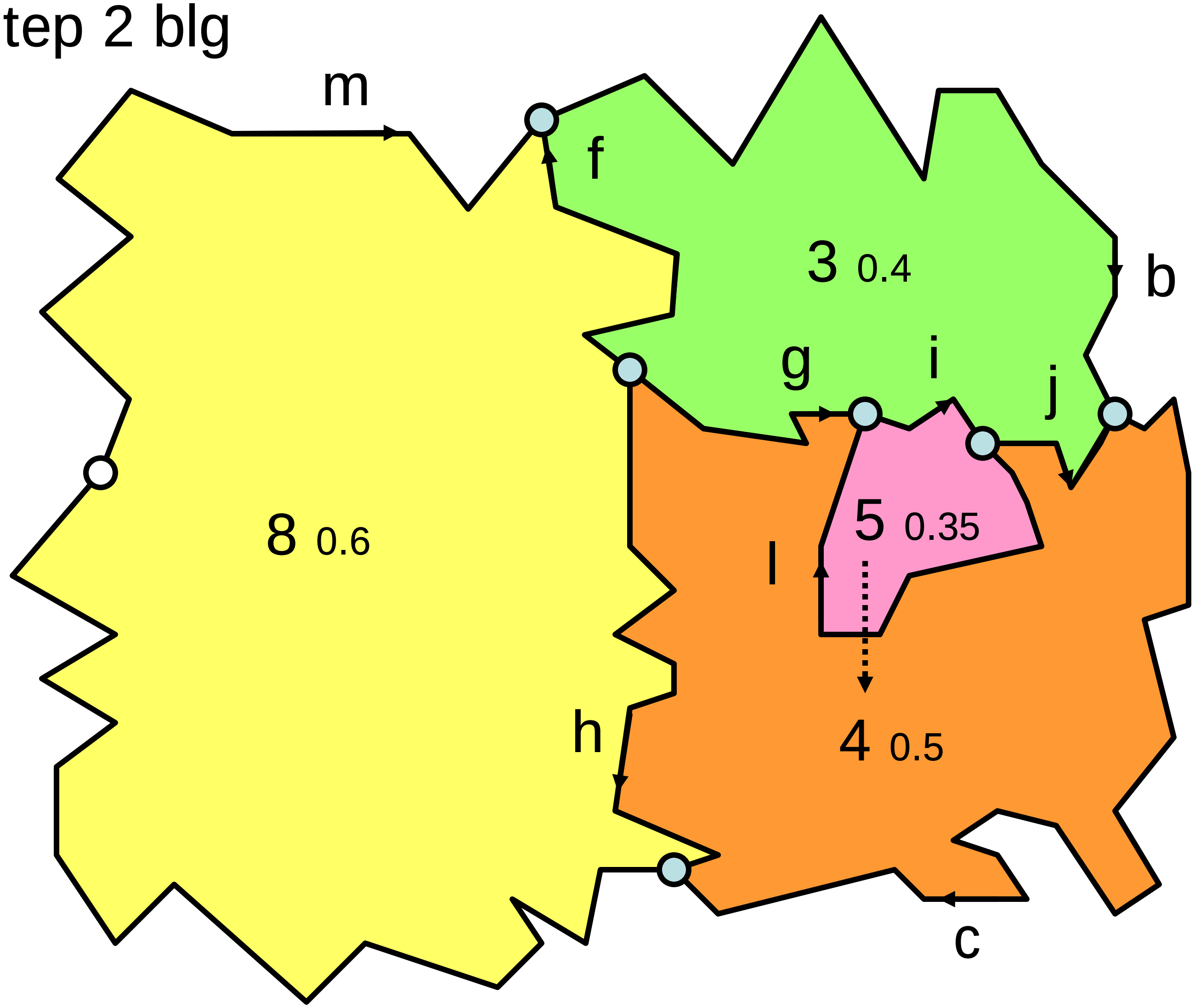


TUDelft

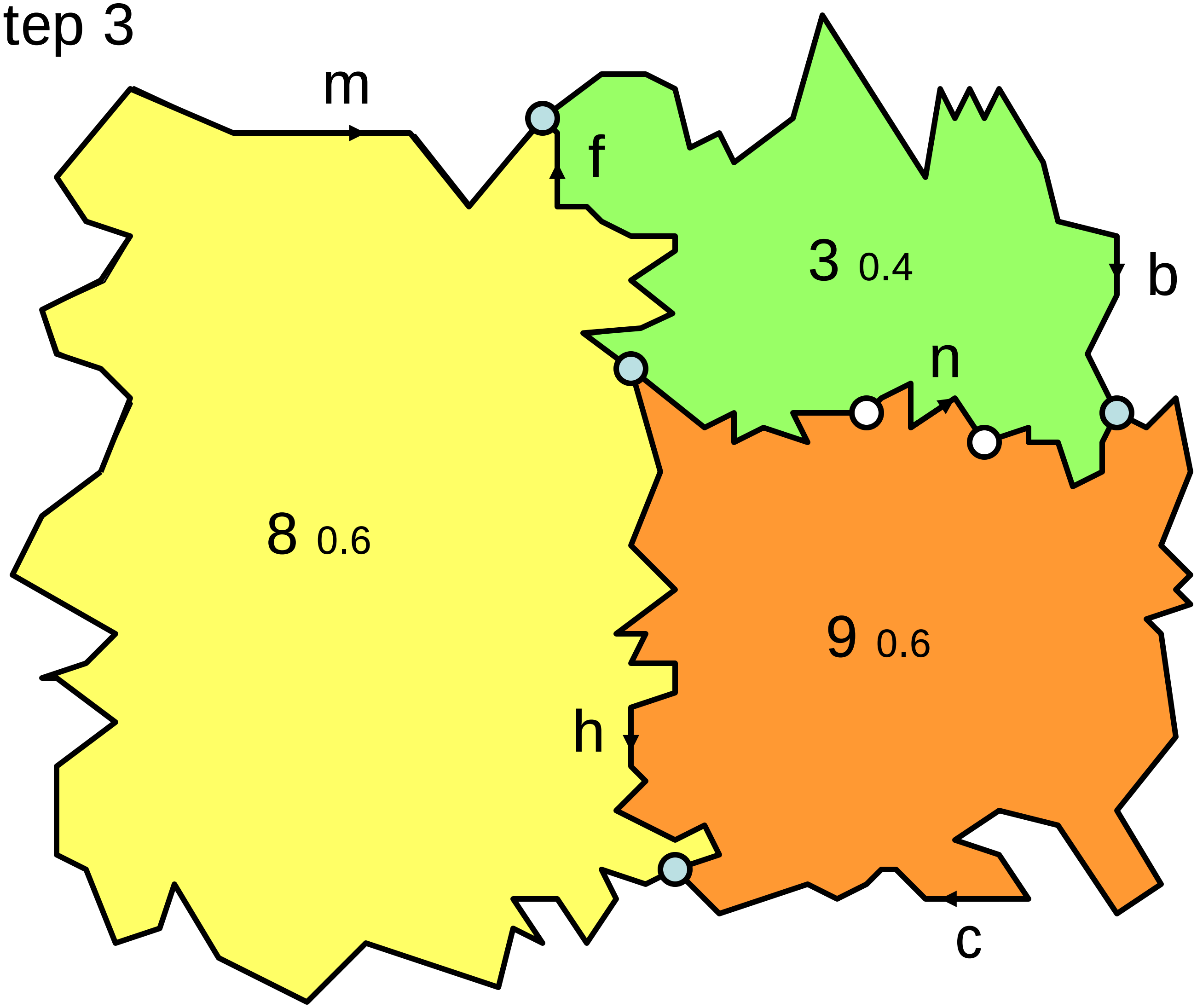
Step 2



Step 2 blg

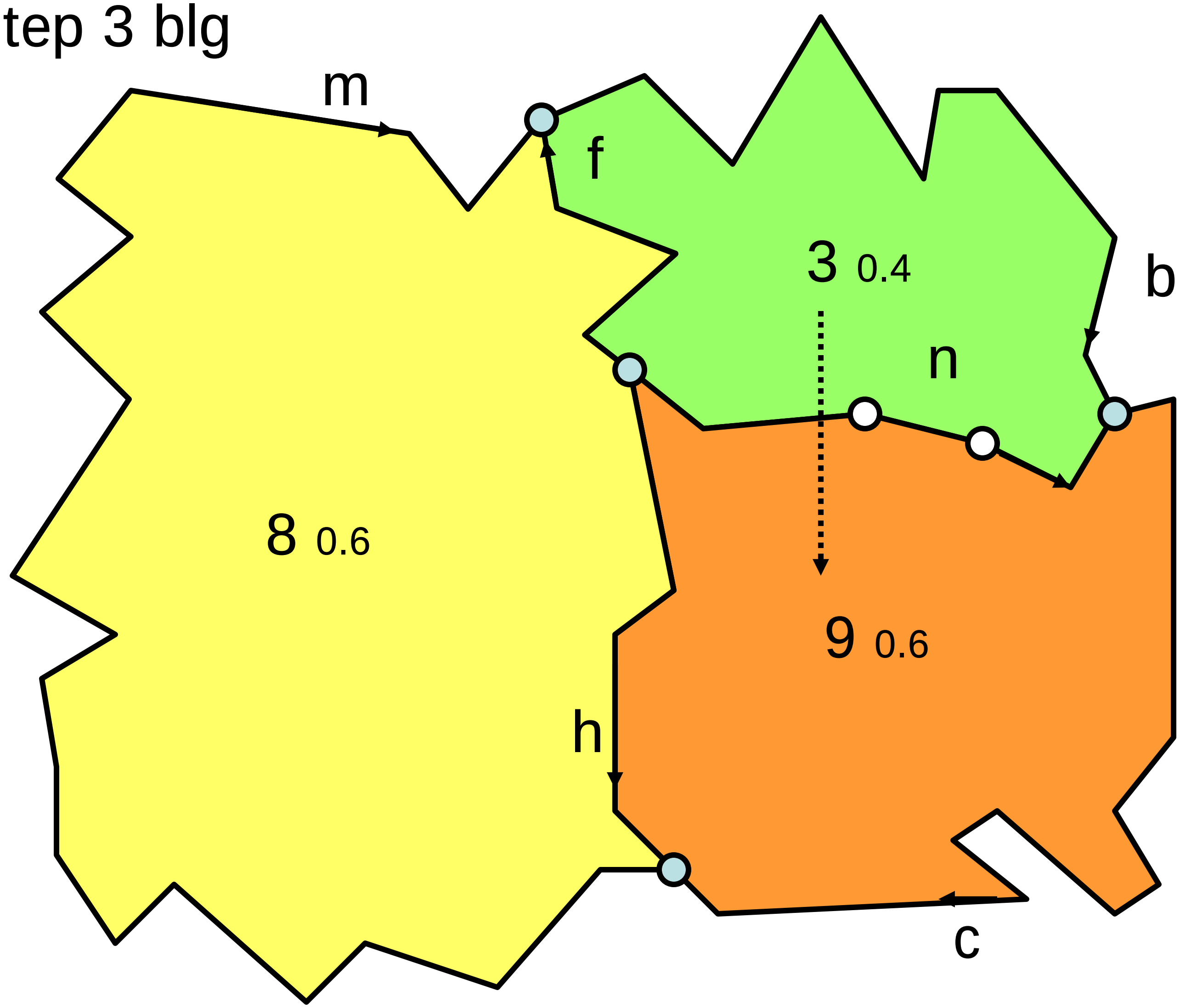


Step 3



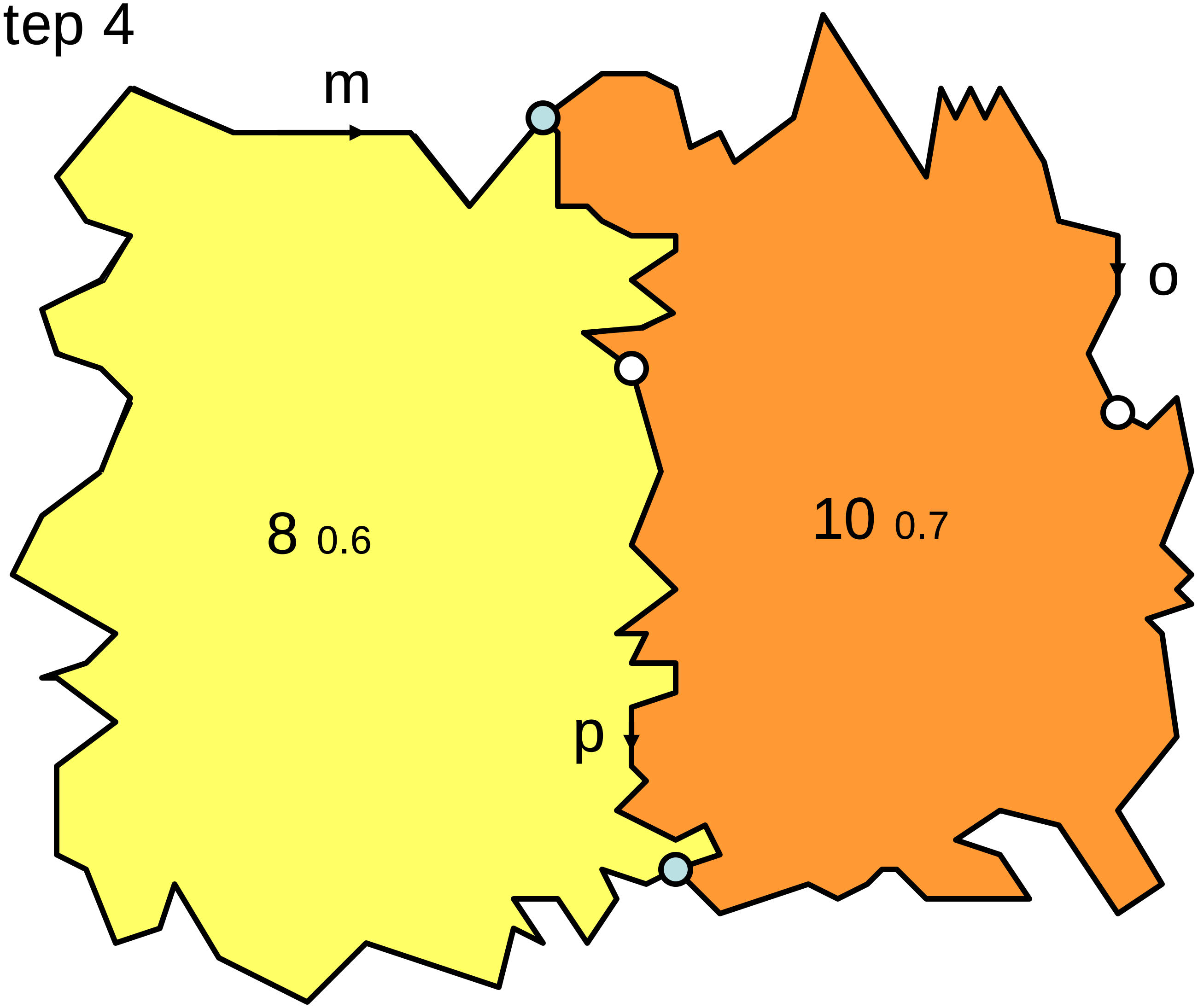
TUDelft

Step 3 blg



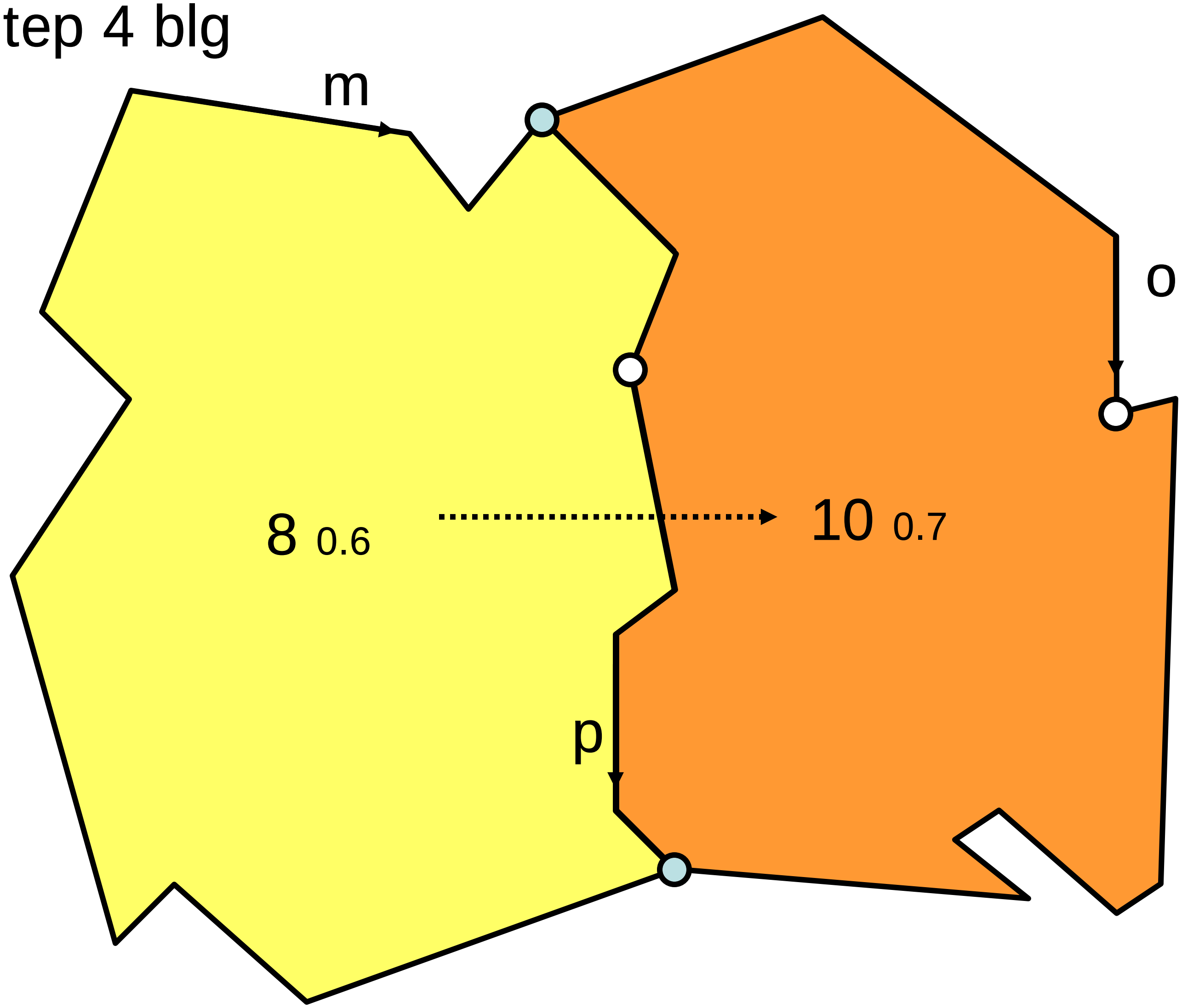
TU Delft

Step 4

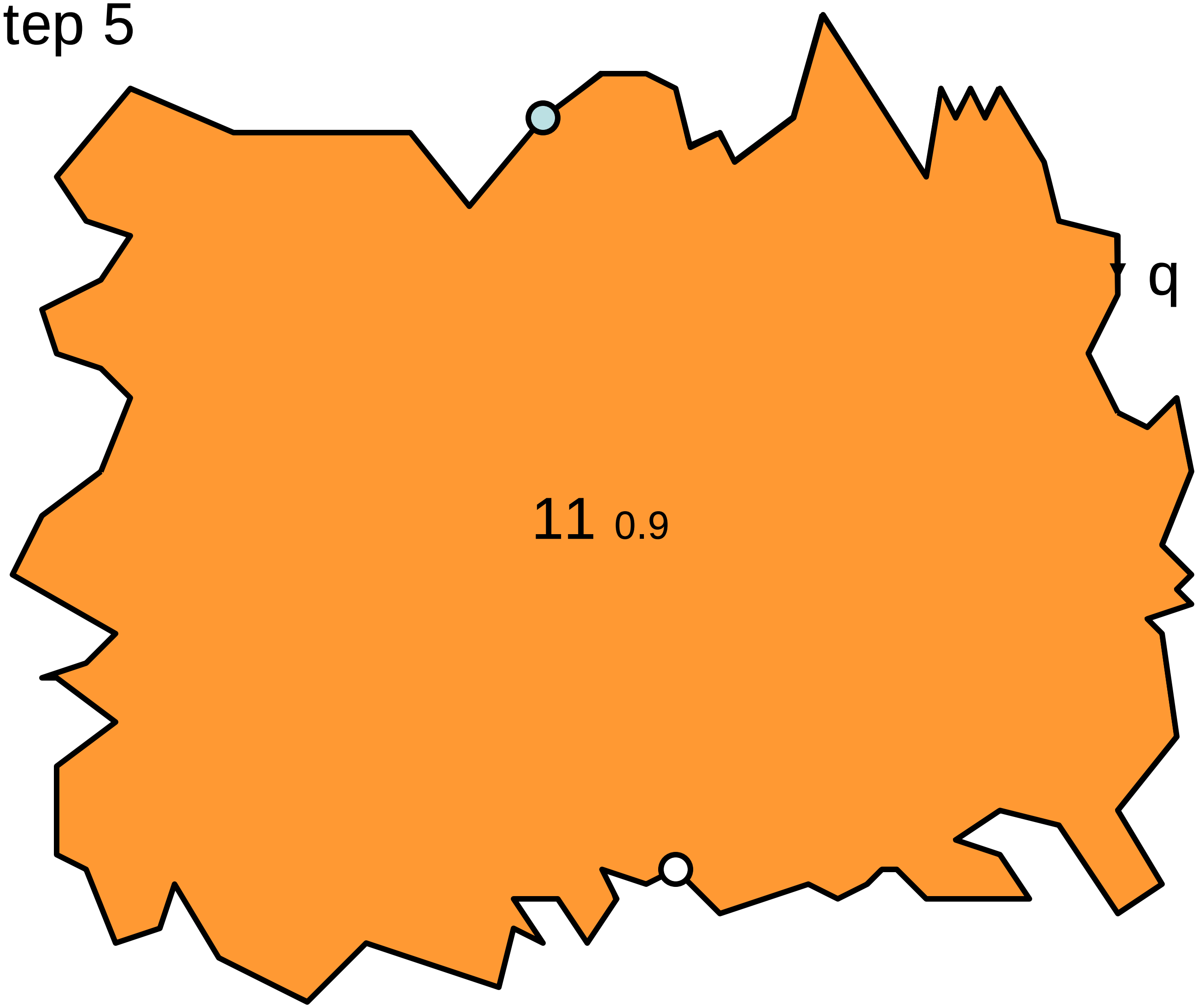


TU Delft

Step 4 blg

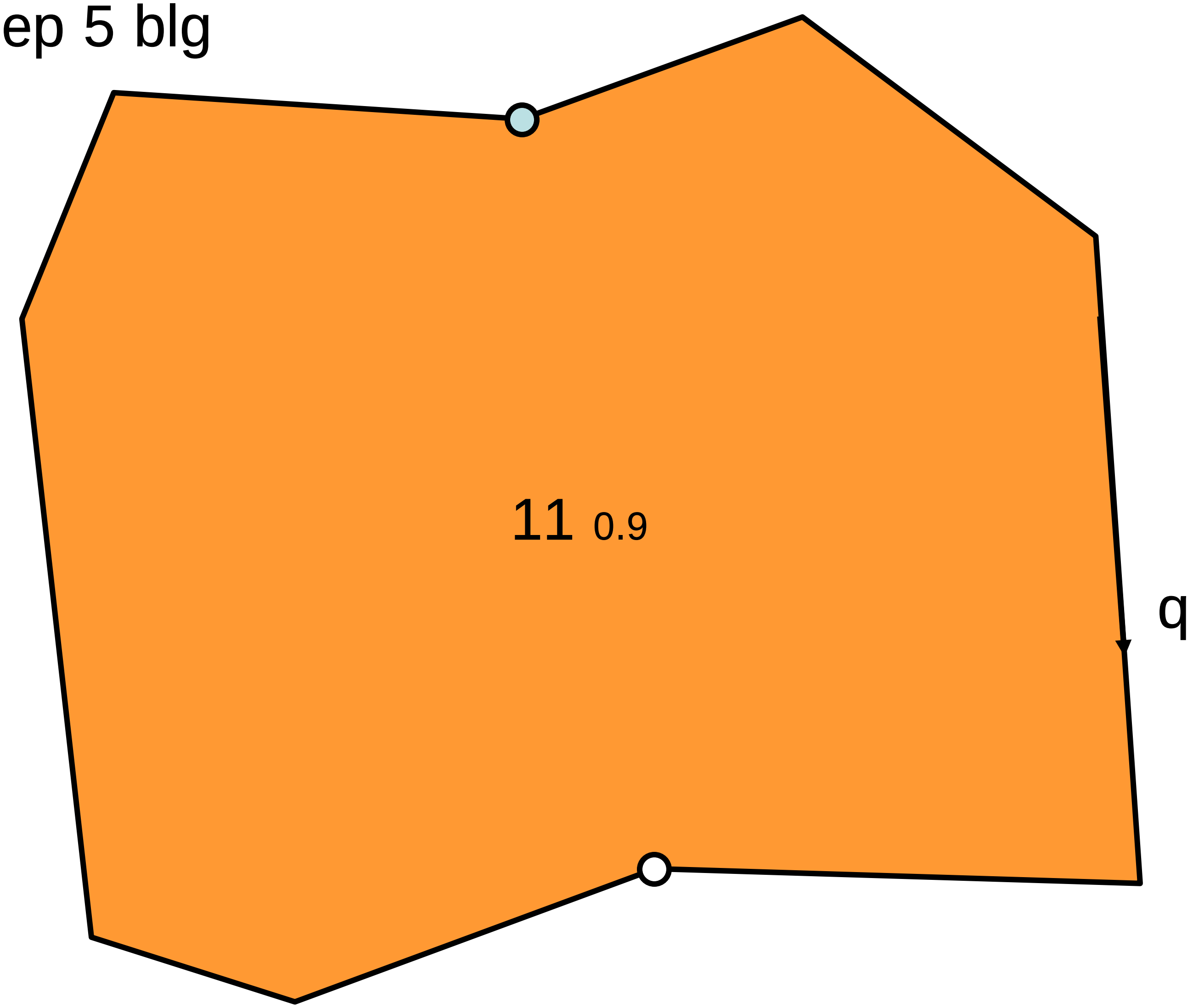


Step 5



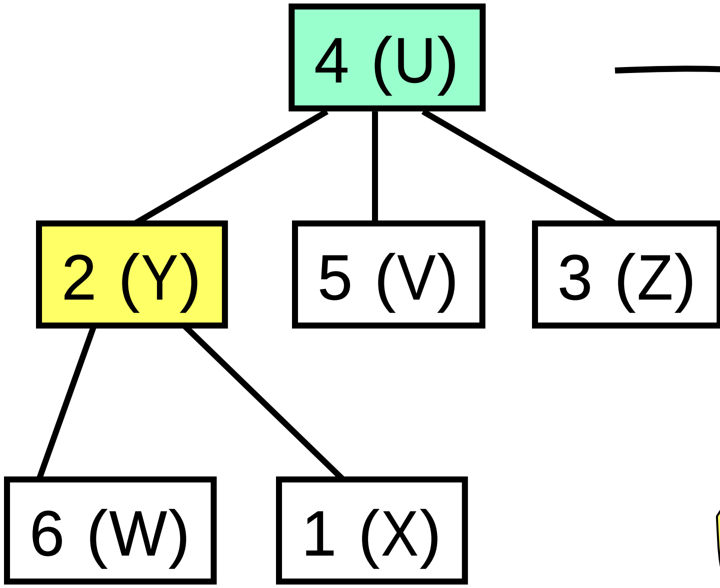
TU Delft

Step 5 blg

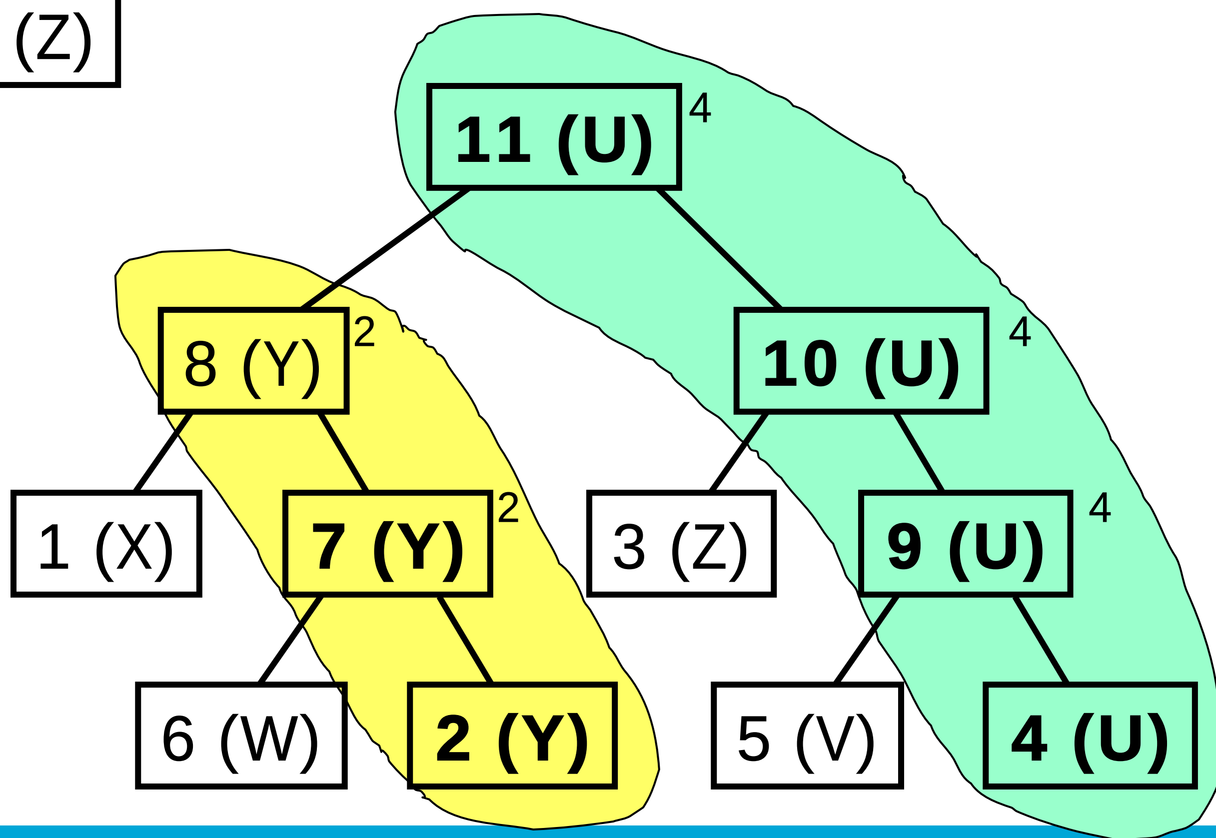


TU Delft

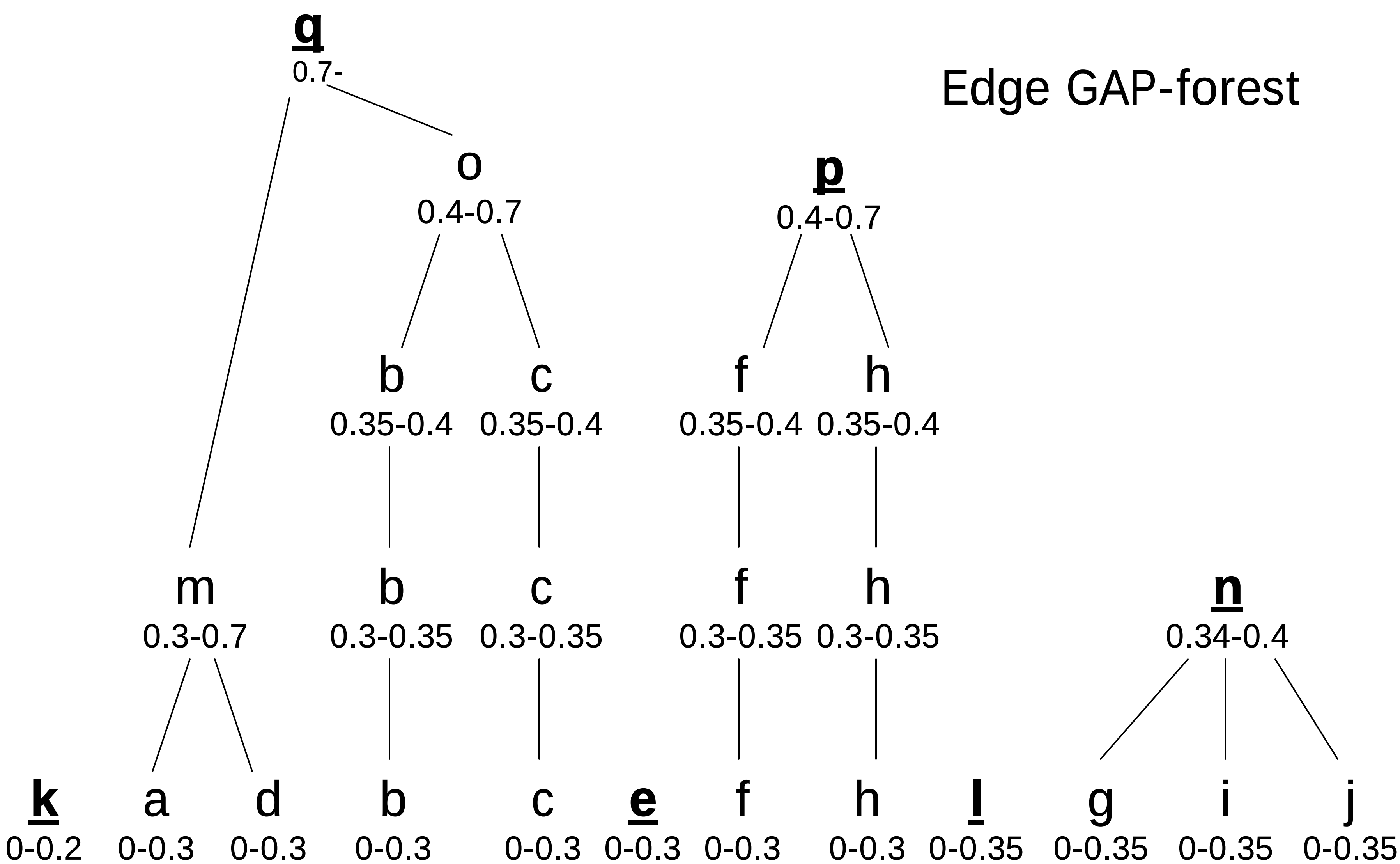
Original GAP-tree

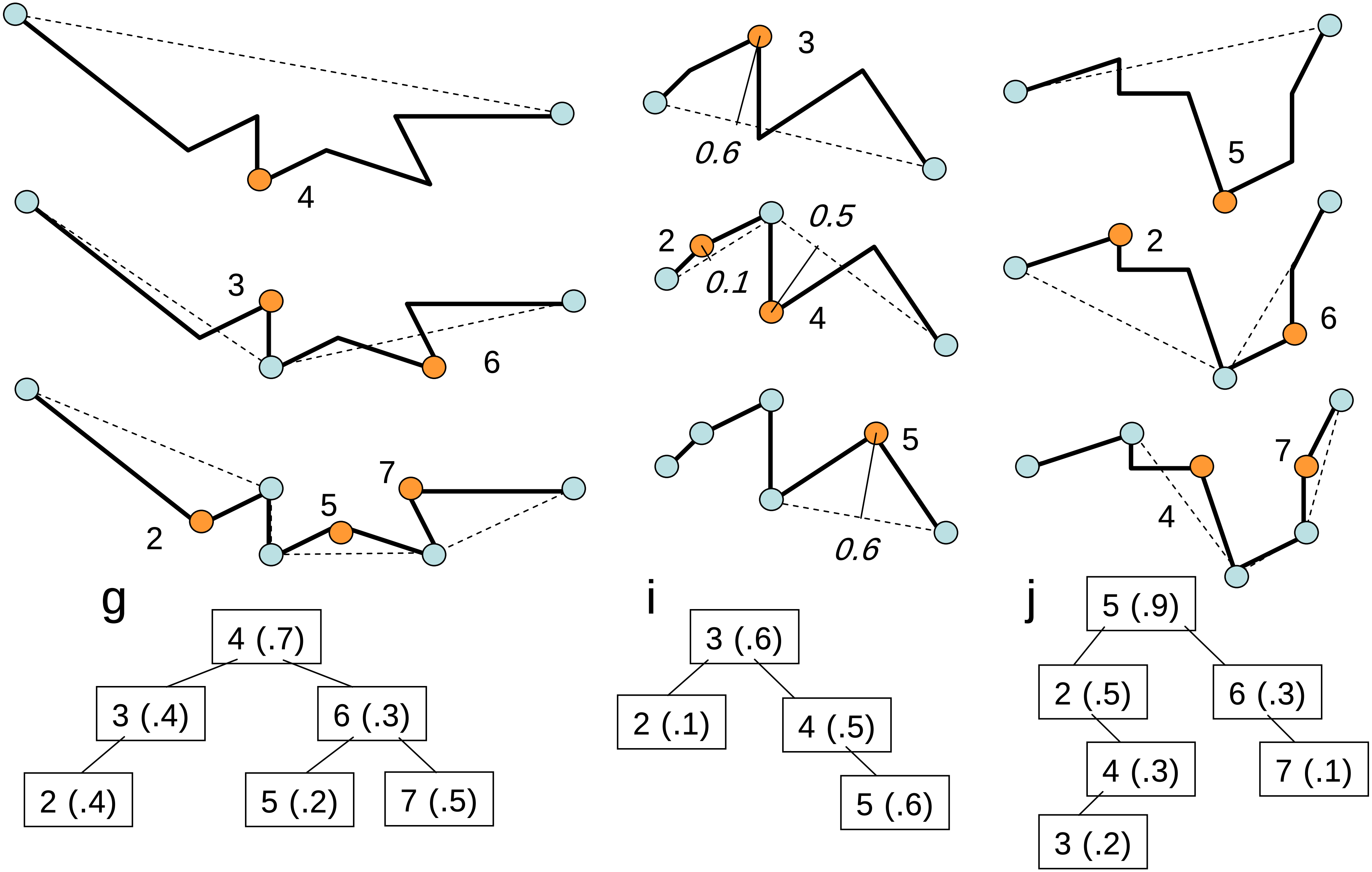


Rewritten face GAP-tree

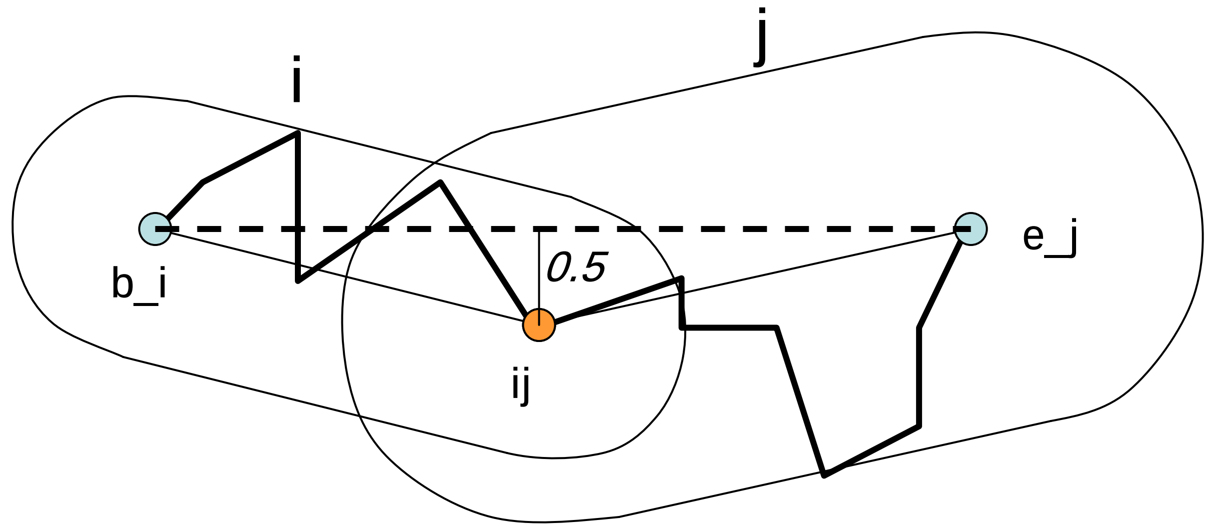


Edge GAP-forest



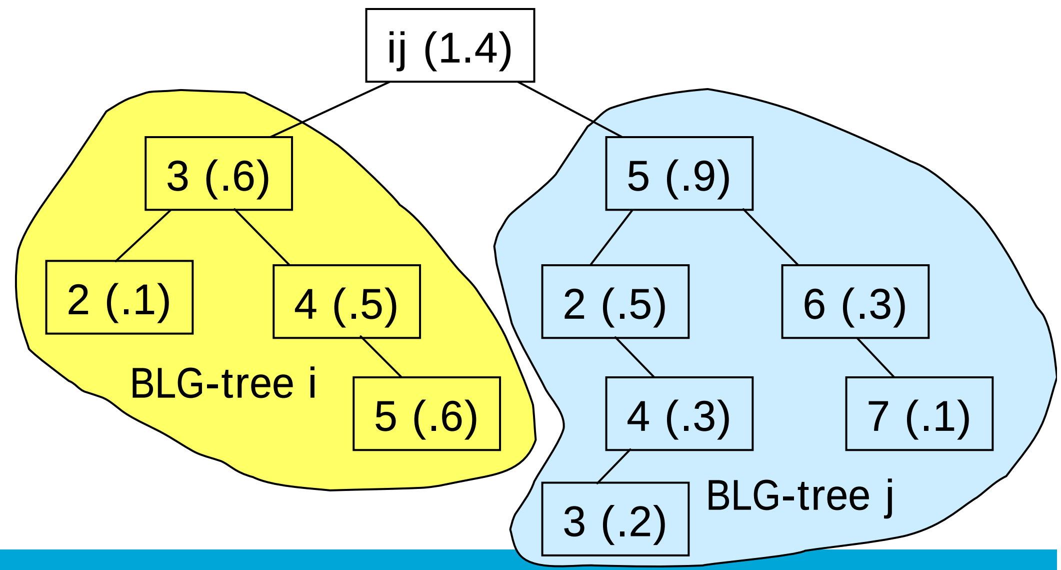

4
3
0.6
5
3
6
2
0.5
0.1
4
2
6
7
5
2
5
0.6
7
4
g
4 (.7)
3 (.4)
6 (.3)
2 (.4)
5 (.2)
7 (.5)
i
3 (.6)
2 (.1)
4 (.5)
5 (.6)
j
5 (.9)
2 (.5)
6 (.3)
4 (.3)
7 (.1)
3 (.2)

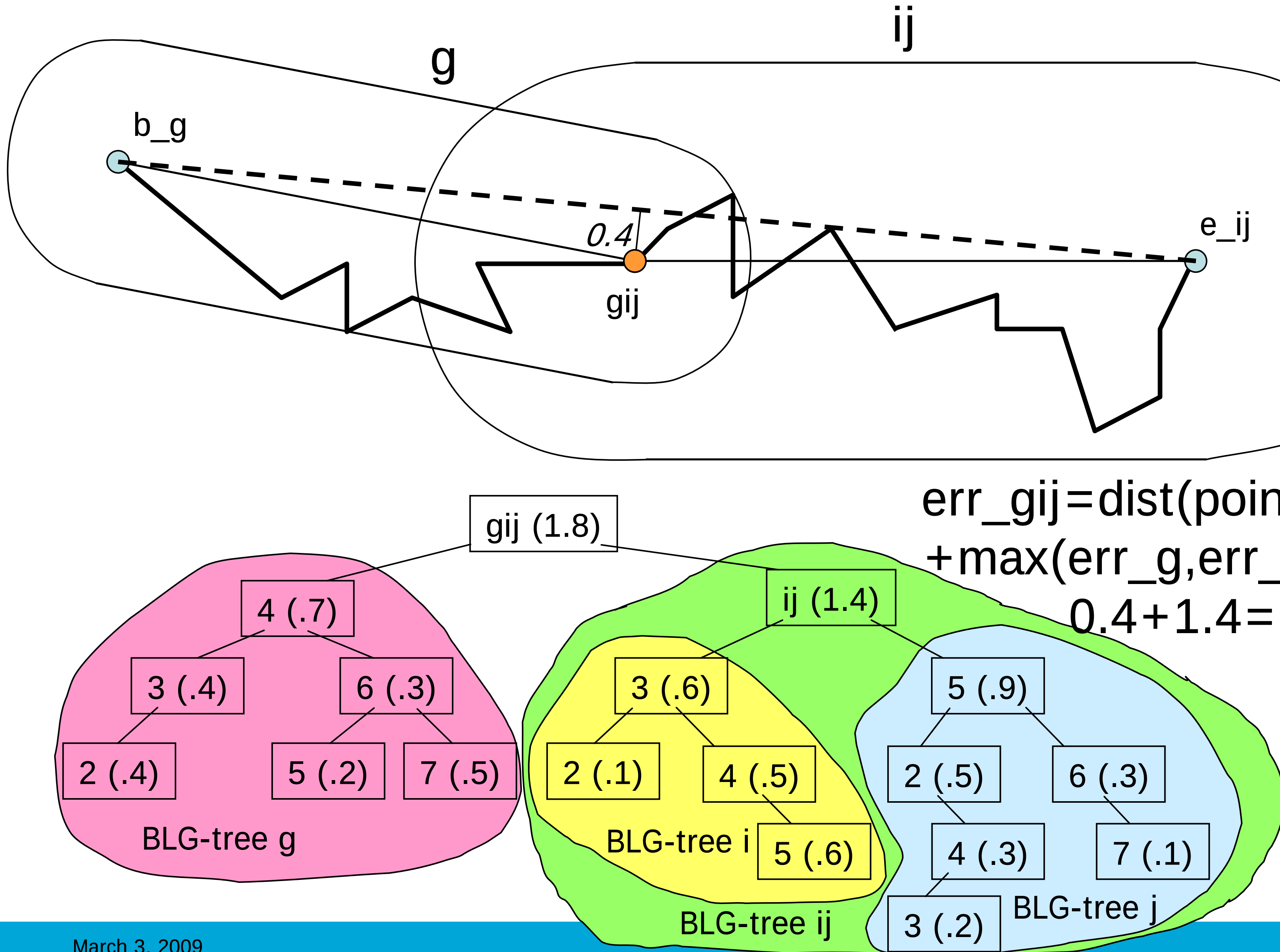
Join BLG-tree's
of edges i and j



err_ij=dist(point(ij), line(b_i,e_j))
+ max(err_i, err_j)=

0.5+0.9=1.4





err_gij=dist(point,line)
+max(err_g,err_ij) =
0.4+1.4=1.8

TU Delft

Contents

4. Storage structure

- Object-relational model
- Spatial data types available (incl. BLG-tree polyline)
- Tables for tgap_face, tgap_edge, and tgap_blg
- Heavy use of views (and spatial functions)
- Also functional indices used

4. Storage structure: tgap_face (1)

step	face id	imp low	imp high	imp orig	first edges	class	pid	*bbox* .
0	1	0.00	0.30	0.30	+a	X	8	(xl,yl,xh,yh)
0	2	0.00	0.20	0.40	+b,-k	Y	7	(xl,yl,xh,yh)
0	3	0.00	0.40	0.40	+c	Z	10	(xl,yl,xh,yh)
0	4	0.00	0.35	0.50	+d	U	9	(xl,yl,xh,yh)
0	5	0.00	0.35	0.35	+i	V	9	(xl,yl,xh,yh)
0	6	0.00	0.20	0.20	+k	W	7	(xl,yl,xh,yh)
1	7	0.20	0.30	0.50	+d	Y	8	(xl,yl,xh,yh)
2	8	0.30	0.40	0.60	+m	Y	10	(xl,yl,xh,yh)
3	9	0.35	0.60	0.60	+c	U	11	(xl,yl,xh,yh)
4	10	0.40	0.60	0.70	+o	U	11	(xl,yl,xh,yh)
5	11	0.60	-	0.90	+q	U	-	(xl,yl,xh,yh)

TU Delft

4. Storage structure: tgap_face (2)

- **face_id** is primary key
- step and bbox (is view) are not stored
- imp_low - imp_high is used importance range
  imp_orig is original importance when face created
- first_edges is variable length array (for islands)
- edge references are singed (clockwise outer boundary)
- polygons can be computed via chain of edge references (winged edge structure)
- GAP-face tree: connect child to parent via pid

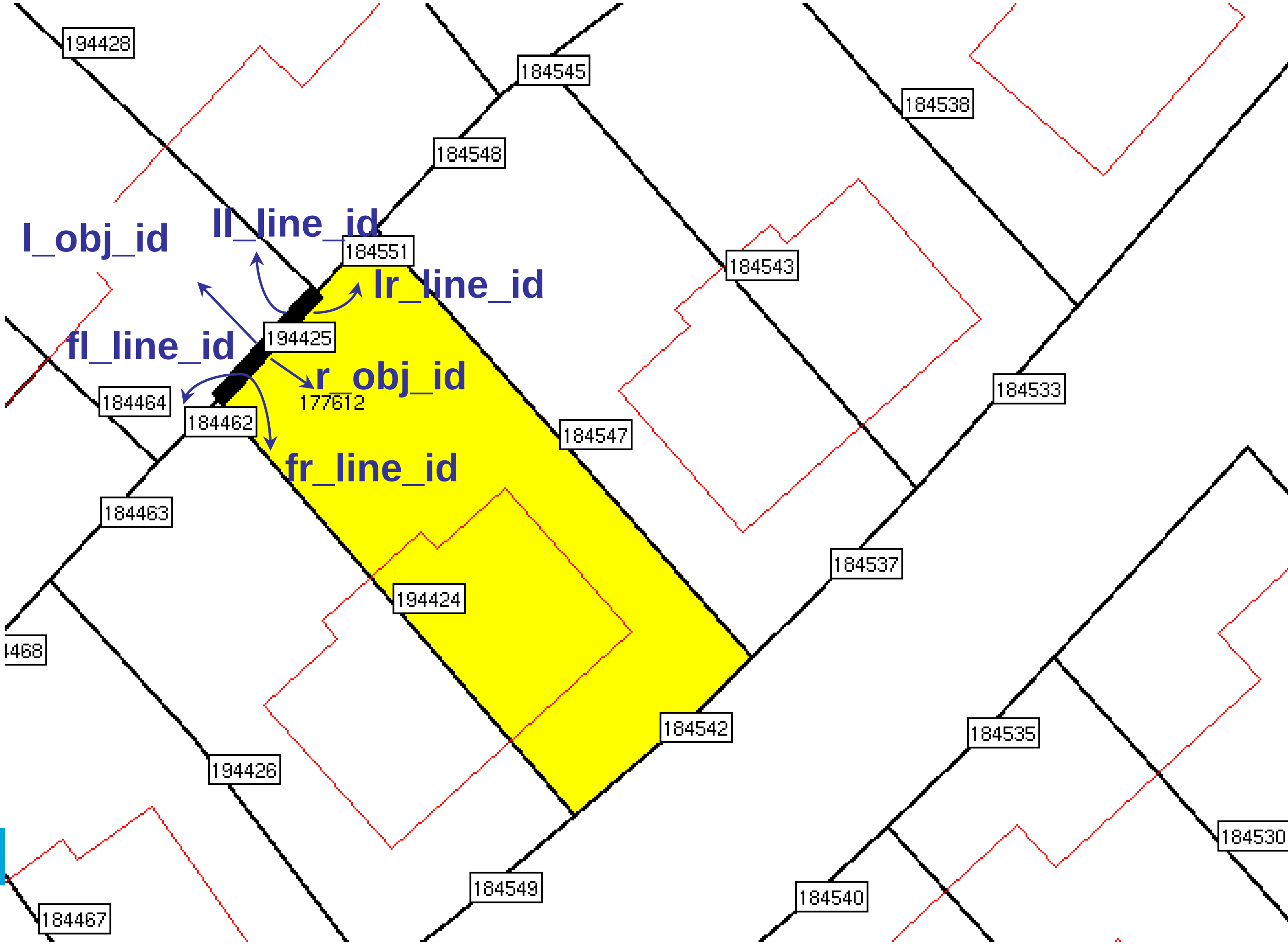
TU Delft

4. Storage structure: tgap_face (3)

```
create view tgap_face_v1 as
select f.face_id, f.imp_low, f.imp_high, f.imp_orig,
  f.first_edges, f.class, f.pid,
  return_polygon(f.face_id) shape
from tgap_face f;

create view tgap_face_v2 as
select f.face_id, f.imp_low, f.imp_high, f.imp_orig,
  f.first_edges,f.class,f.pid,f.shape, get_bbox(f.shape) bbox,
  get_area(f.shape) area, get_perimeter(f.shape) perimeter
from tgap_face_v1 f;
```

4. Storage structure: topology model winged edge



4. Storage structure: tgap_edge (1)

step	edge id	imp low	imp high	face left	face right	edge fl	edge fr	edge ll	edge lr	pid	abox	BLG-tree (blg_id)
0	a	0.00	0.30	0	1	-d	-e	+b	-f	m	Union(l,r)	tree+xy..
0	b	0.00	0.30	0	3	-a	-f	+c	-j	b	Union(l,r)	tree+xy..
0	c	0.00	0.30	0	4	-b	-j	+d	-h	c	Union(l,r)	tree+xy..
0	d	0.00	0.20	0	2	-c	-h	+a	-e	d	Union(l,r)	tree+xy..
...												
0	l	0.00	0.35	4	5	+j	-i	-g	+i	-	Union(l,r)	tree+xy..
1	d	0.20	0.30	0	7	-c	-h	+a	-e	m	old	old
1	e	0.20	0.30	7	1	+h	+f	-d	+a	-	old	old
1	h	0.20	0.30	4	7	+g	+e	-c	+d	h	old	old
2	m	0.30	0.40	0	8	-c	-h	+b	-f	m	Union(l,r)	BLG a+d
2	b	0.30	0.35	0	3	-m	-f	+c	-j	b	old	old
...												
5	q	0.60	-	0	11	-q	-q	+q	+q	-	Union(l,r)	BLG m+o

4. Storage structure: tgap_edge (2)

- **edge_id, imp_low** are primary key
- Step not stored
- reference changes to previous version in red
- Winged edge: face (left/right) edge (fl,fr,ll,lr)
- GAP-edge forest: connect child to parent via pid (interesting parents, unequal to prev, in red), note the multiple roots
- abox (union left/right bbox) is function/view (needed for efficient selection)
- blg-tree stored in separate table (avoid redundancy)

4. Storage structure: tgap_edge (3)

- Less edge references are possible (‘fr’ or ‘ll’ not used), this would save rows in table;
- It is even possible to drop all edge references, avoiding more rows (e.g. b, 0.30), but more searching
- View to compute abox:

```
create view tgap_edge_v1 as
  select e.edge_id, e.imp_low,.. , union(l.bbox, r.bbox) abox
  from tgap_edge e, tgap_face_v2 l, tgap_face_v2 r
  where e.face_left=l.face_id and e.face_right=r.face_id;
```


TU Delft

4. Storage structure: tgap_blg (1)

blg_id	BLG_tree_source	top_tolerance	child1	child2
1	tree+xy..	-1	-	-
2	tree+xy..	–1	-	-
..				
10	-	1.4	1	2

- **blg_id** is primary key
- 2 types of rows:
  1. Leafs: contain blg-tree/polyline source (in Postgres by Schenkelaars/vO'95 and in Oracle by Vermeij'03
  2. Non-leafs: contain 2 references to childeren

TUDelft

4. Storage structure: tgap_blg (2)

```
create view tgap_blg_v1 as (
  select b.blg_id,
    b.BLG_tree_source BLG_tree
  from tgap_blg b
  where b.top_tolerance = -1)
union all (
  select b.blg_id,
    merge_BLG(b.top_tolerance,b.child1,b.child2) BLG_tree
  from tgap_blg b
  where b.top_tolerance <> -1);
```

View to hide differences between leafs and non-leafs

4. Storage structure: tgap_blg (3)

```
create view tgap_edge_v2 as
  select e.edge_id, e.imp_low, e.imp_high,
    e.face_left, e.face_right,
    e.edge_fl, e.edge_fr, e.edge_ll,
    e.edge_lr, e.pid, e.abox,
    b.BLG_tree
  from tgap_edge_v1 e, tgap_blg_v1 b
  where e.blg_id=b.blg_id;
```

View definition to combine the edge and its BLG-tree

TUDelft

4. Storage structure: Reactive-tree

```
create index tgap_face_idx on
    tgap_face(compute_3D_block(get_bbox(return_polygon(face_id)),
                               get_imp_range(imp_low, imp_high)))
  indextype is 3D_rtree;
```

- Instead of real Reactive-tree an pseudo Reactive-tree is used: 3D R-tree with 3rd dimension importance range
- Note that this is a functional index on the 3D blocks (xl,yl,imp_low,xh,yh,imp_high)
- Several views (and tables) are used to compute this
- Besides indexing also spatial/imp clustering needed

TUDelft

Contents

5. Client-server progressive refinement: concept

- Server starts sending most important nodes in GAP face-tree/edge-forest (in selected search rectangle)
- Client builds partial copy of GAP/BLG-structure
  - → can be used to display coarse impression
  - → every (x) seconds this structure is redisplayed
- Server keeps on sending more data and GAP/BLG-structure at client is growing (with more details)
- Possible stop criteria:
  1. 1000 objects (meaningful info density on screen)
  2. Required imp level is reached (with tolerance value)
  3. User interrupts the client

TU Delft

5. Client-server progressive refinement: in practice

- MSc-thesis student working on server side (Oracle)
- Proposal for extending Web Feature Service with notion of importance (GetFeature with importannce and delta-importance)
- Real testing will start in context new 3 year project ‘Usable, well-scaled mobile maps’ (with TNO Human factors, ANWB, ITC, ESRI, LaserScan)

TU Delft

Contents

TU Delft

6. Conclusions, main results

- First time ever non-redundant geometry scaleless data structure has been presented (based on topology)
- tGAP is well suited for web-environment:
  1. No geometric processing at client side
  2. Supports progressive refinement
- The class importance values and classes compatibility matrix are crucial for quality of the structure
- Views can be used for ‘stupid’ clients (non-tGAP-aware)

TU Delft

6. Conclusions, tuning

- Implementation and practical test (millions of rows) are needed for tuning the structure
- Benchmark have to be performed with alternatives (multiple-representation approaches and redundant scaleless approaches)
- Two important test client environments:
    1. Desktop GIS
    2. Distributed Web-GIS
- What is the price of non-redundancy, that is, the many references? (storage and speed)

TU Delft

6. Conclusions, further enhancements

- Data editing (at most detailed level), local propagation to higher levels, dynamic structures
- Support for non-area objects (Reactive-tree for index):
  1. Points: own table with importance range
  2. Lines: same but now with reference to BLG-repr.
  3. Maybe also combine 2 less important lines in 1
- Change from area to line (or point) representation at certain moment. Similar to normal GAP-face tree when face is removed, but now at same time it is introduced in point or line table (with link).