Vario-scale data server in a web service context

Peter van Oosterom, Marian de Vries and Martijn Meijers

ICA Workshop on Generalisation and Multiple Representation,
25 June 2006, Vancouver, Washington USA

July 6, 2006

Section GIS Technology

TU Delft
Delft University of Technology

Contents

1. Introduction
2. Background tGAP structure
3. First implementation
4. Client-server set-up and progressive refinement
5. Conclusions

TU Delft

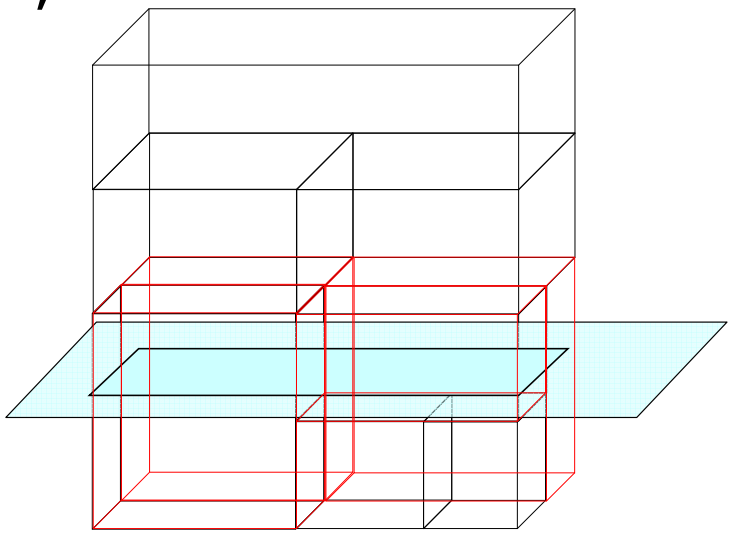
1. Introduction

- 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)

1. Generalized Area Partitioning-tree (GAP-tree) history

- In normal GAP-tree (van Oosterom 1993) 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

1. tGAP structure (GAP-face tree + GAP-edge forest)

- Also coordinate redundancy between edges at different aggregation levels is removed
- Throughout remainder of presentation examples of the tGAP-structure (creation and use) will be 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)

1. Proposed solution: tGAP structure

- Variable scale: infinite amount of levels
- Base level with most detailed geometry/topology
- Create links/structure on top

- Only theory → validate
- Last year at Autocarto 2005, presentation was concluded with question: What is the price of non-redundancy, that is, the many references?

- Test and implement structure

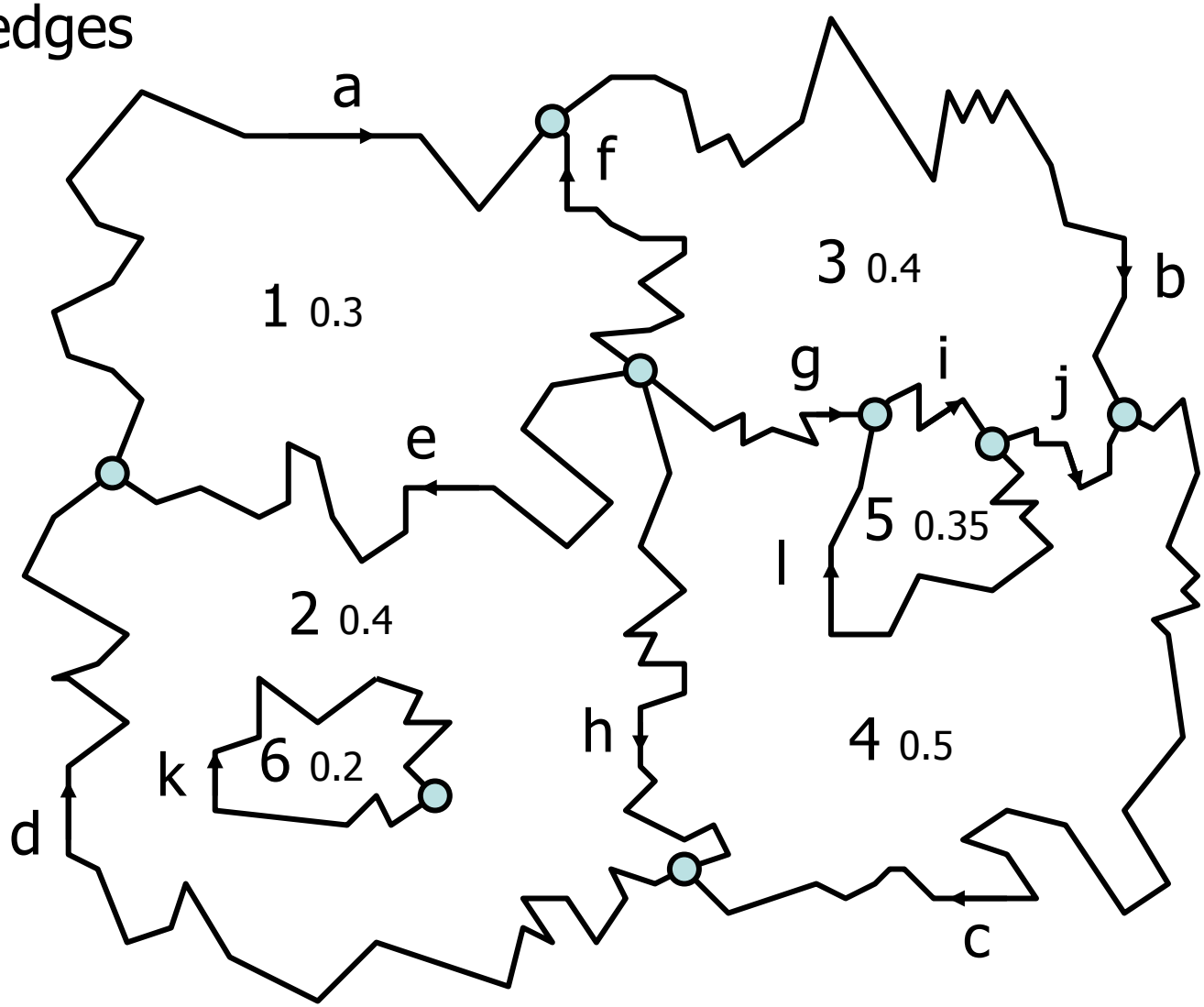
TU Delft

Contents

1. Introduction
2. Background tGAP structure
3. First implementation
4. Client-server set-up and progressive refinement
5. Conclusions

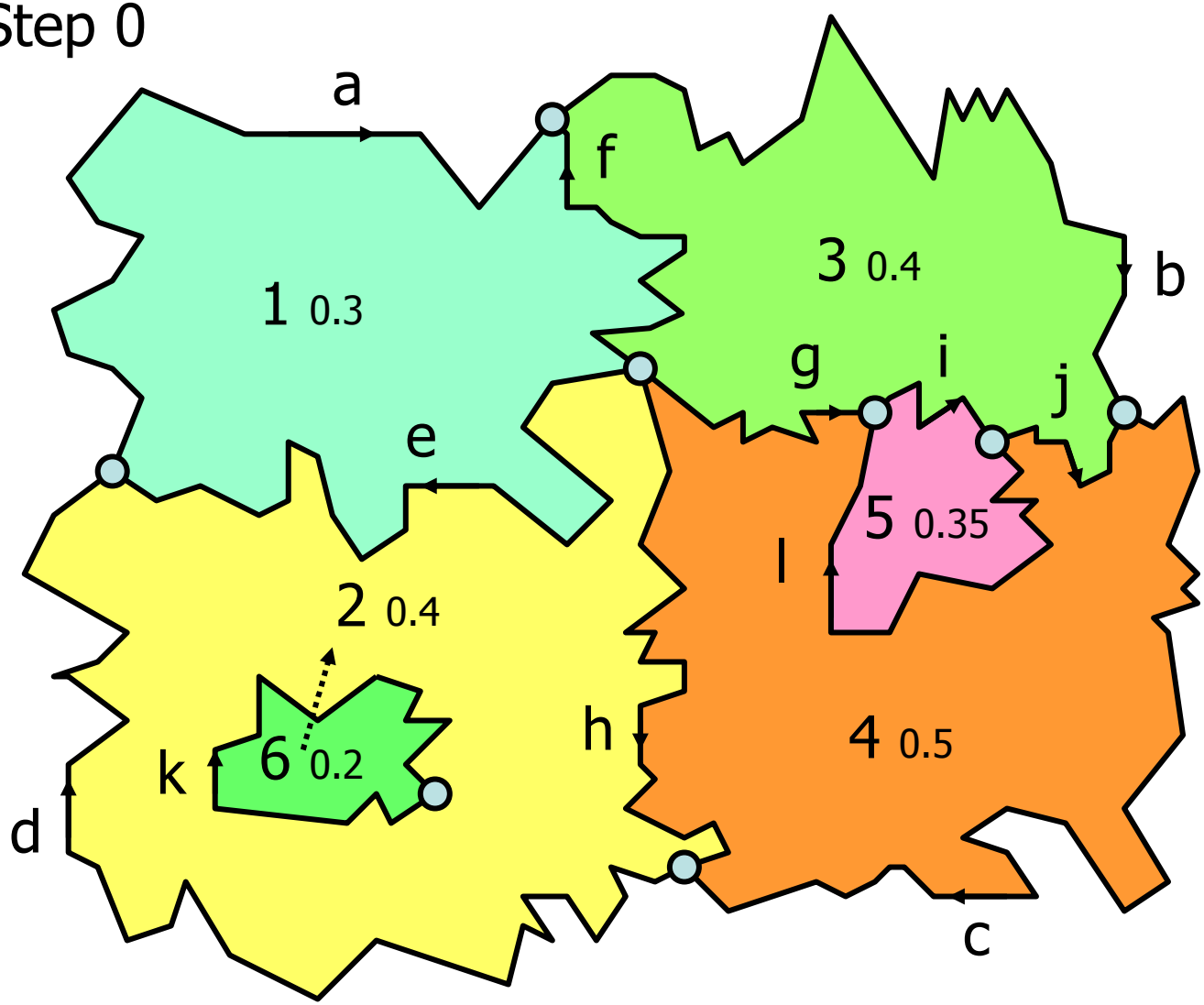
TU Delft

edges



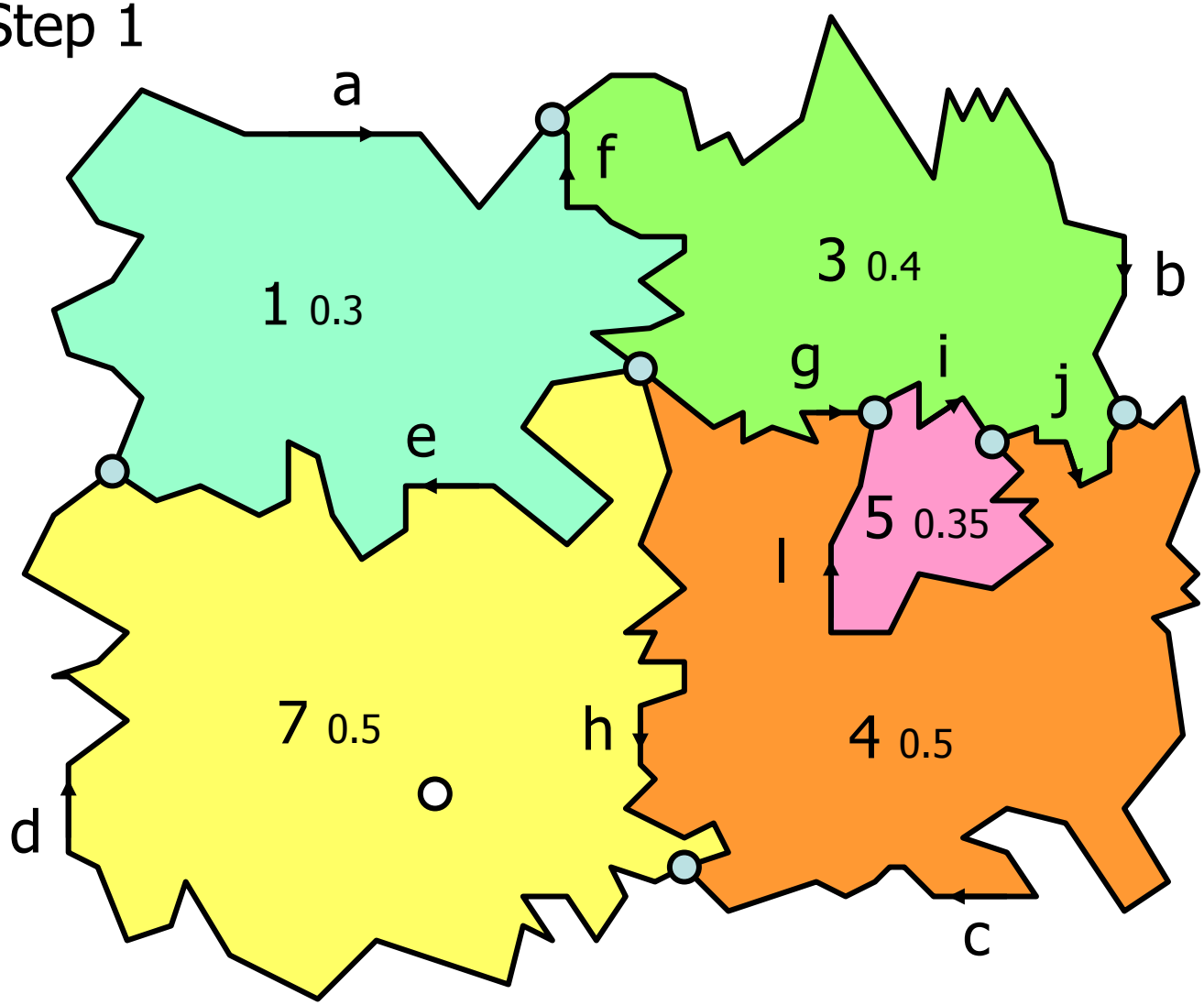
TU Delft

Step 0



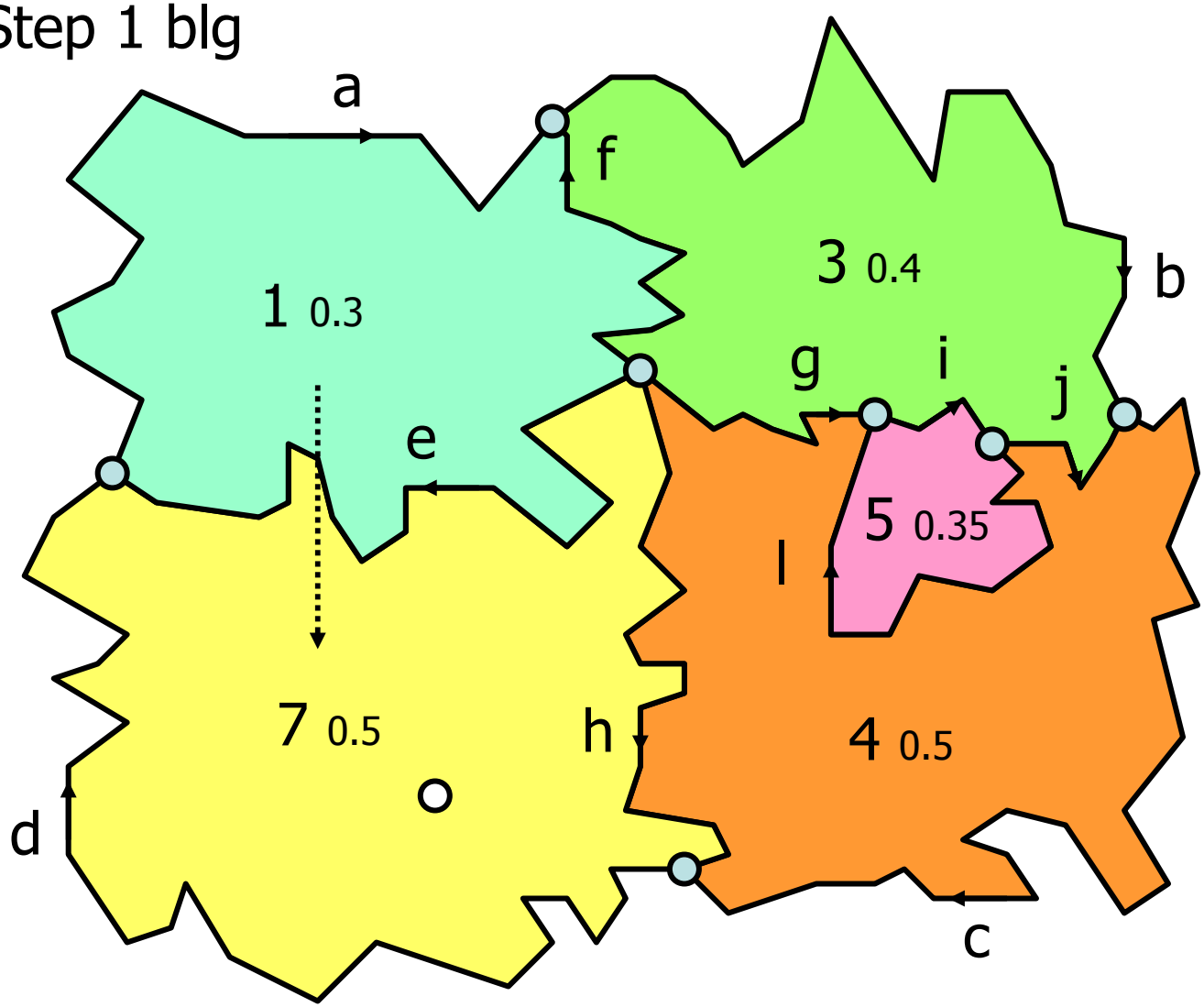
TU Delft

Step 1



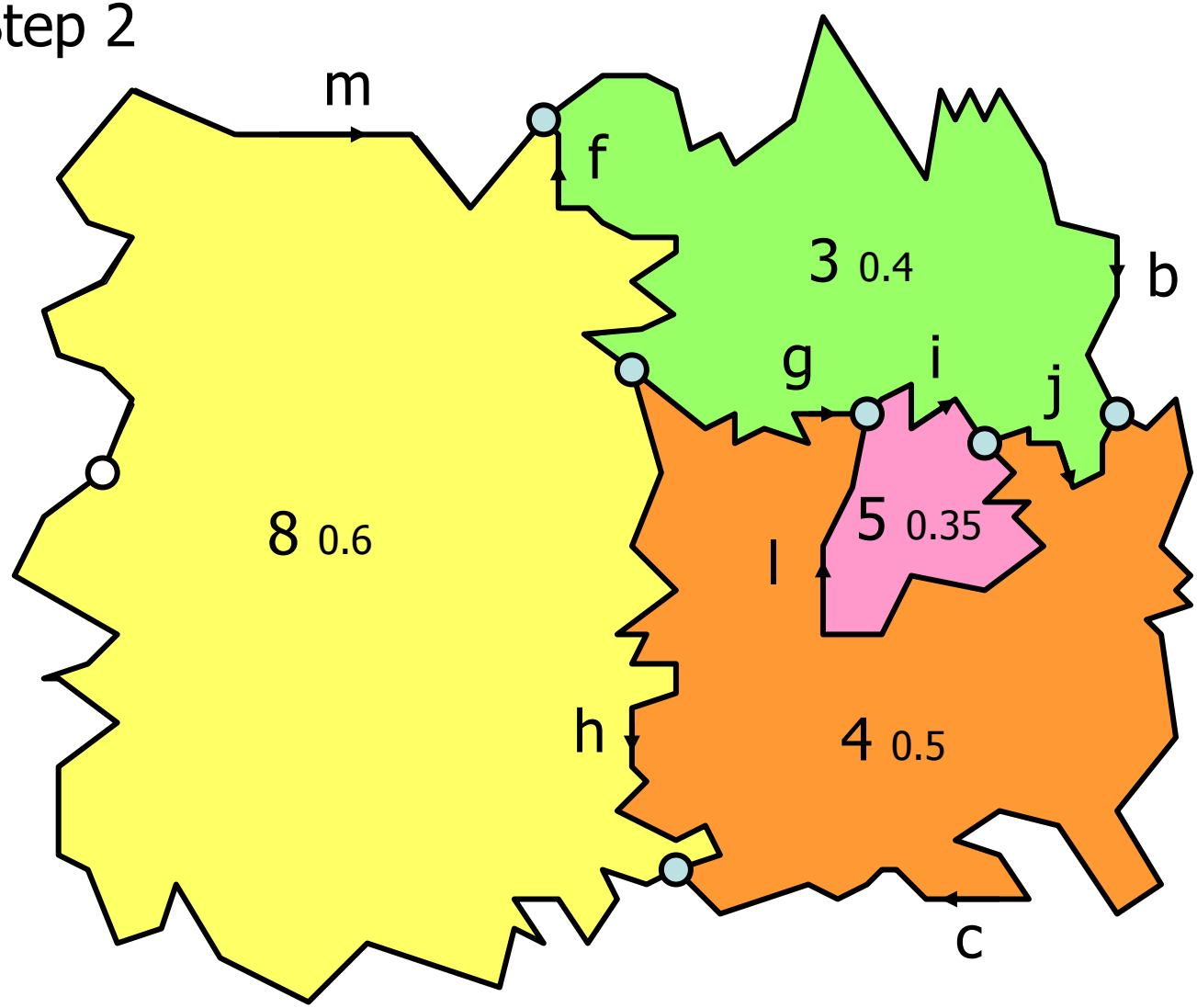
TU Delft

Step 1 blg



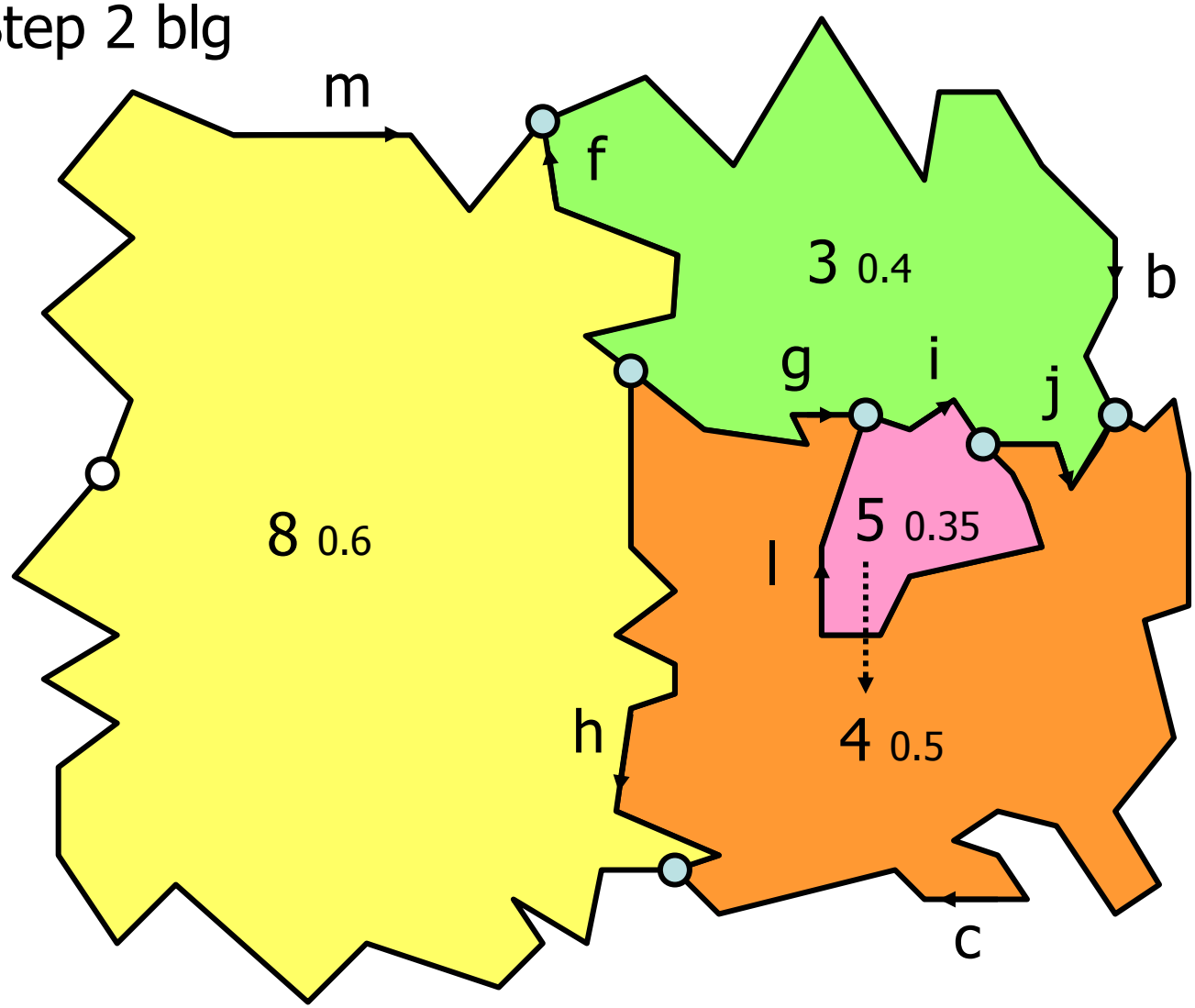
TU Delft

Step 2



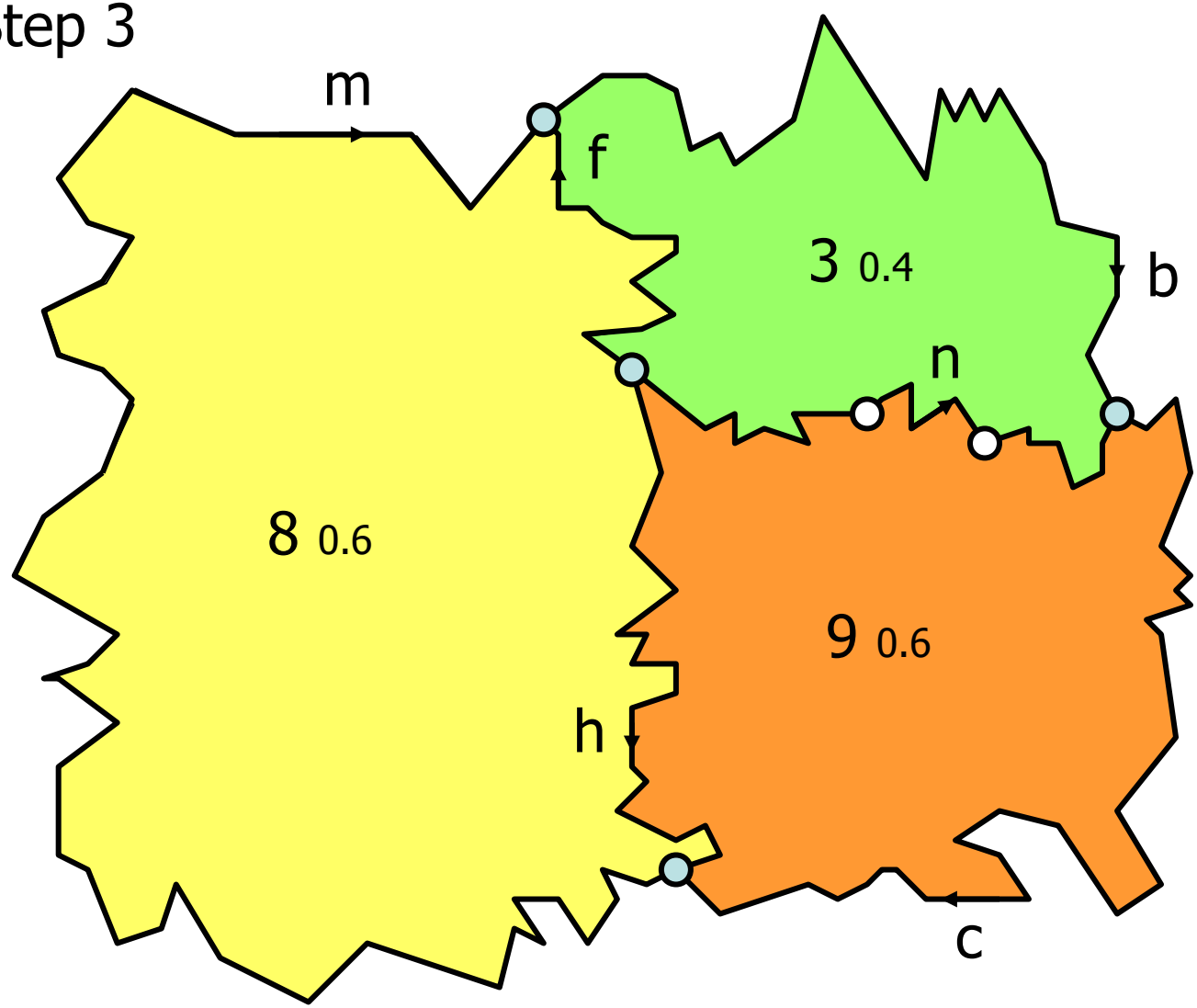
TU Delft

Step 2 blg



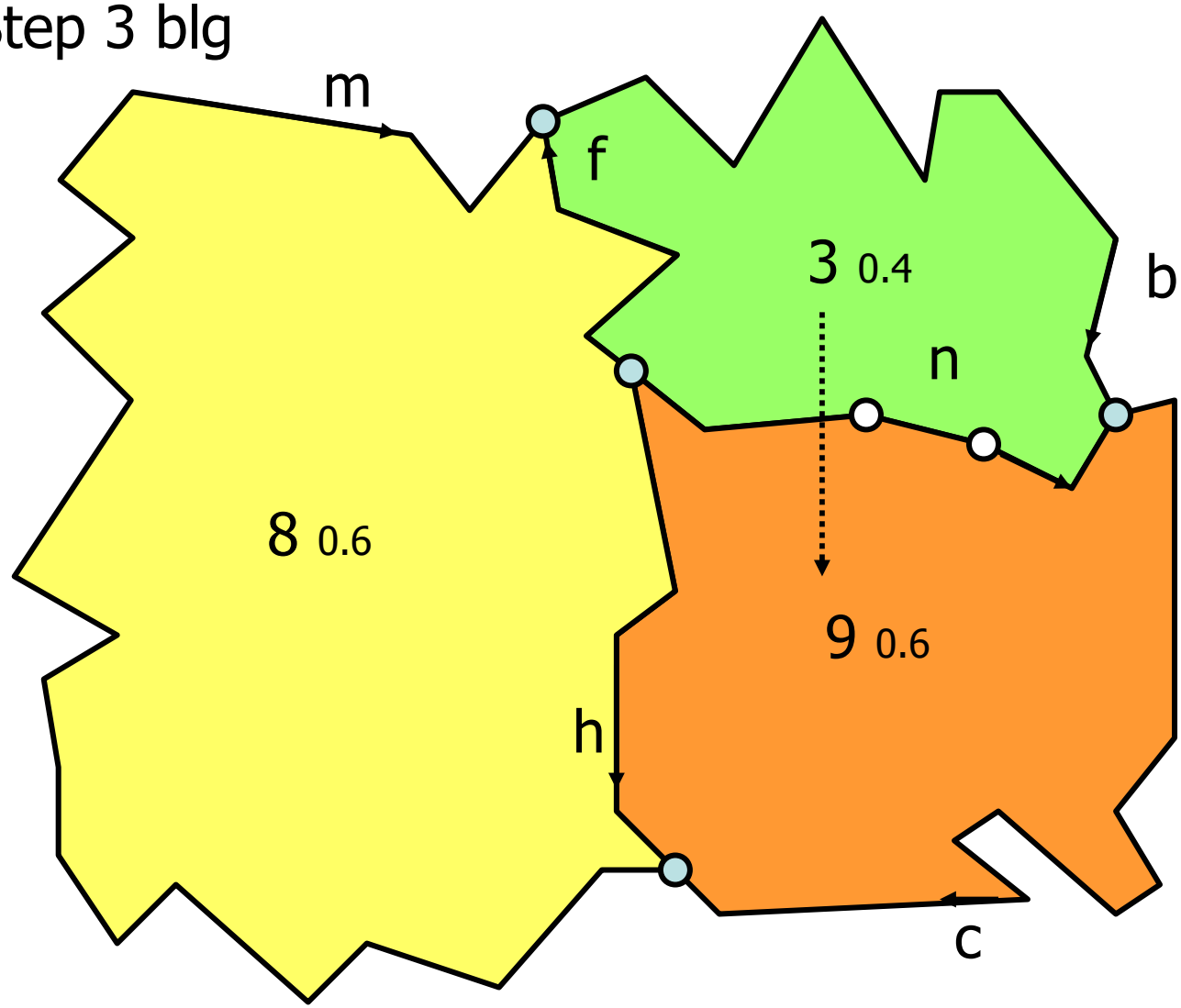
TU Delft

Step 3



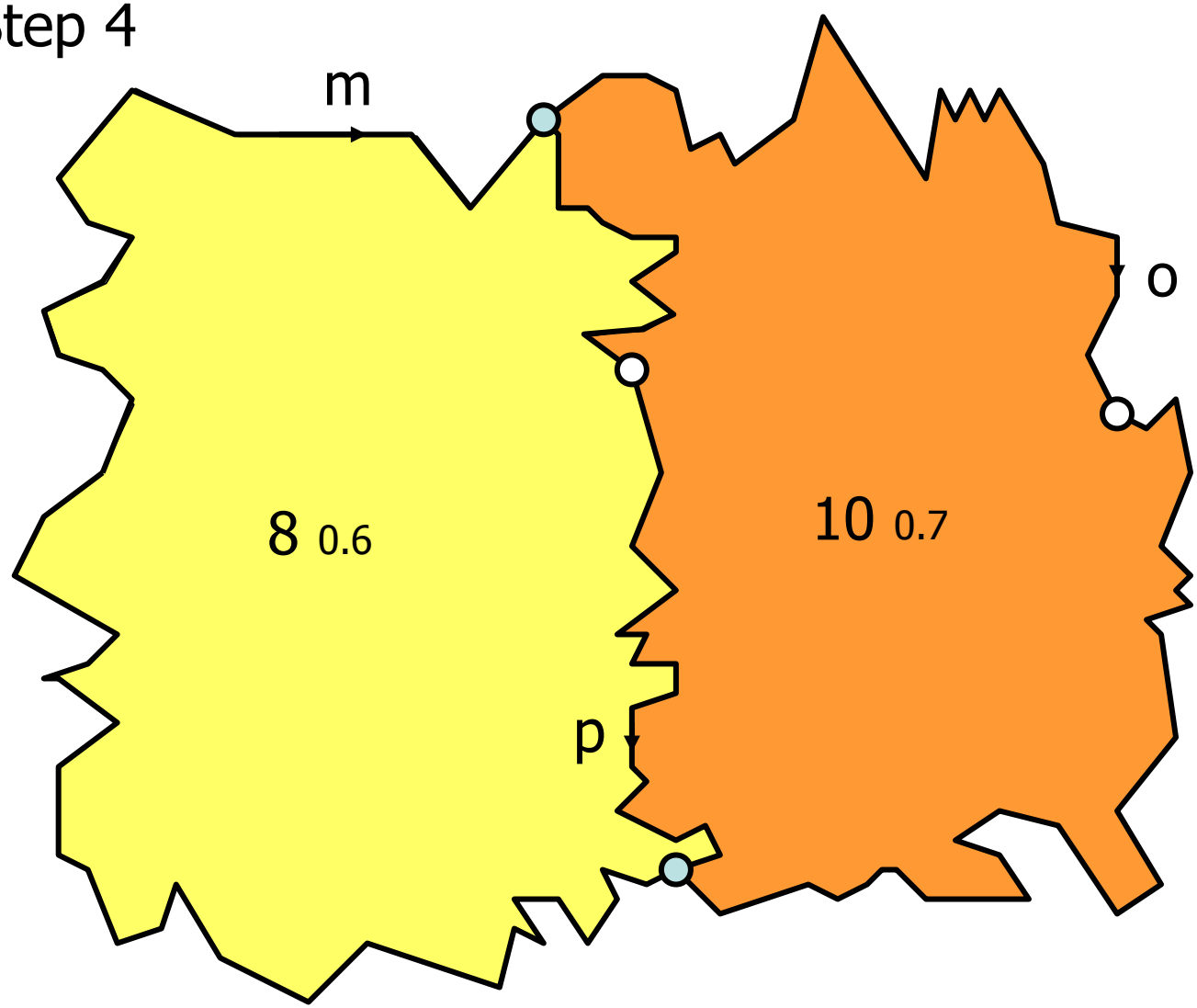
TU Delft

Step 3 blg



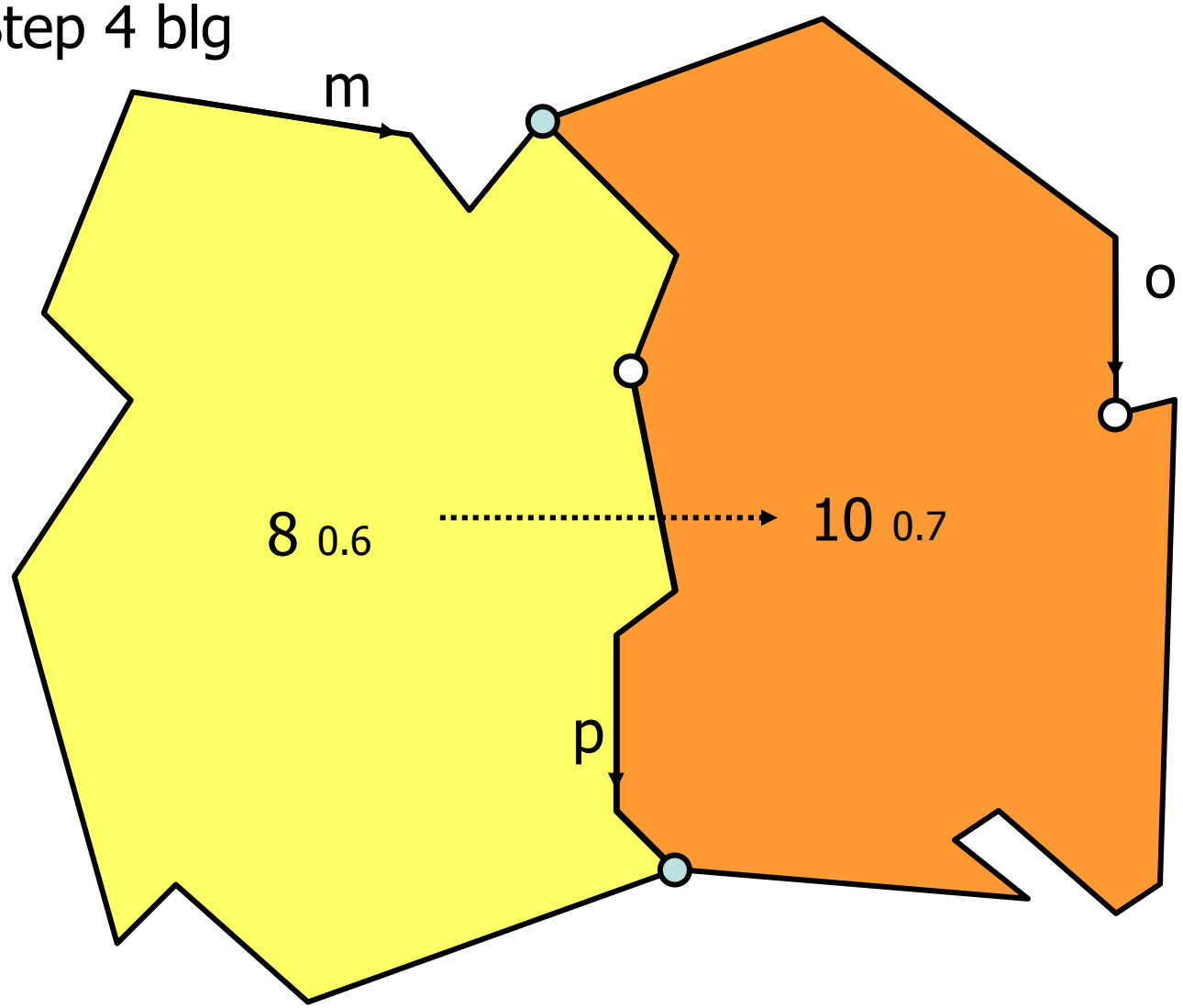
TU Delft

Step 4



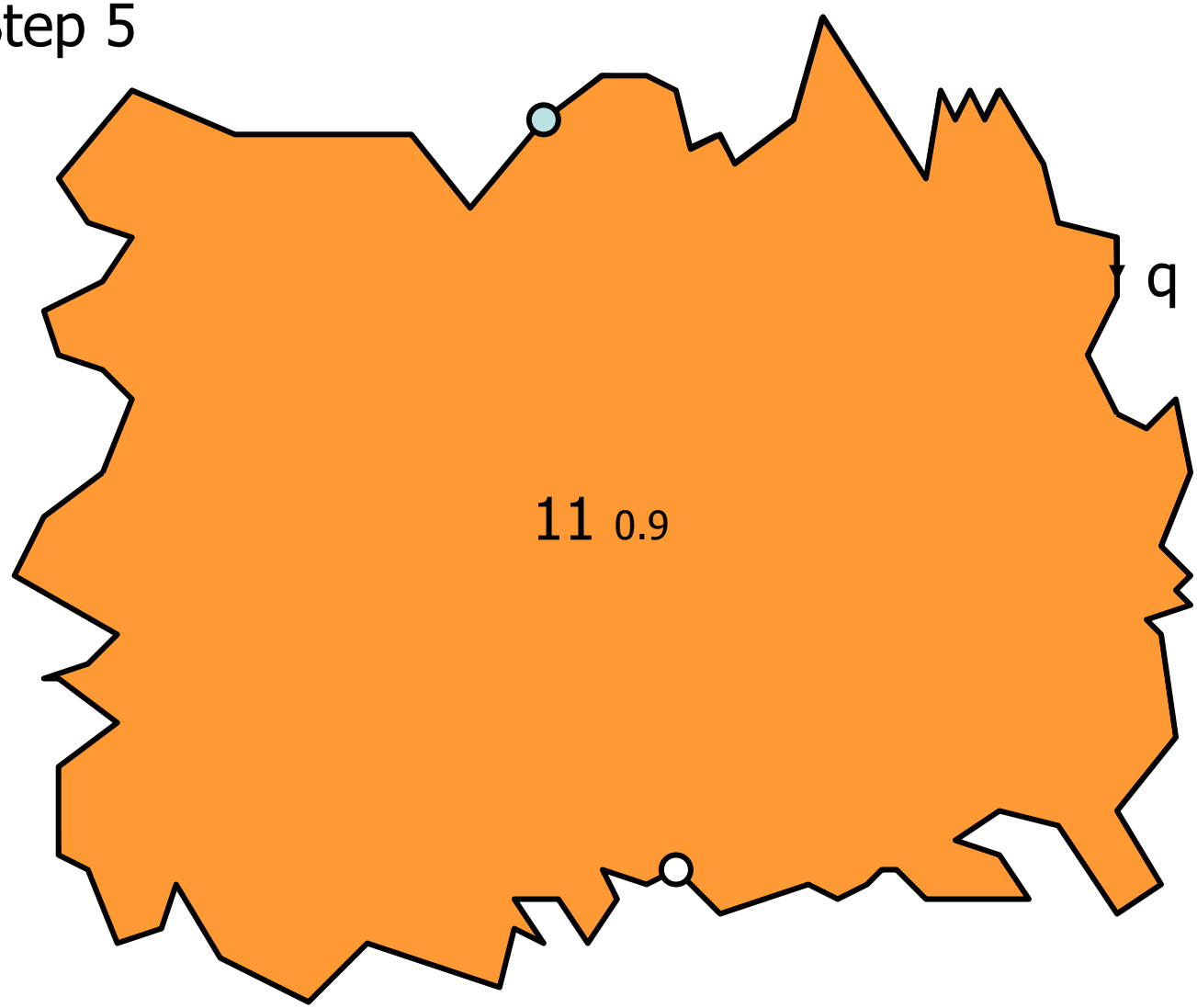
TU Delft

Step 4 blg



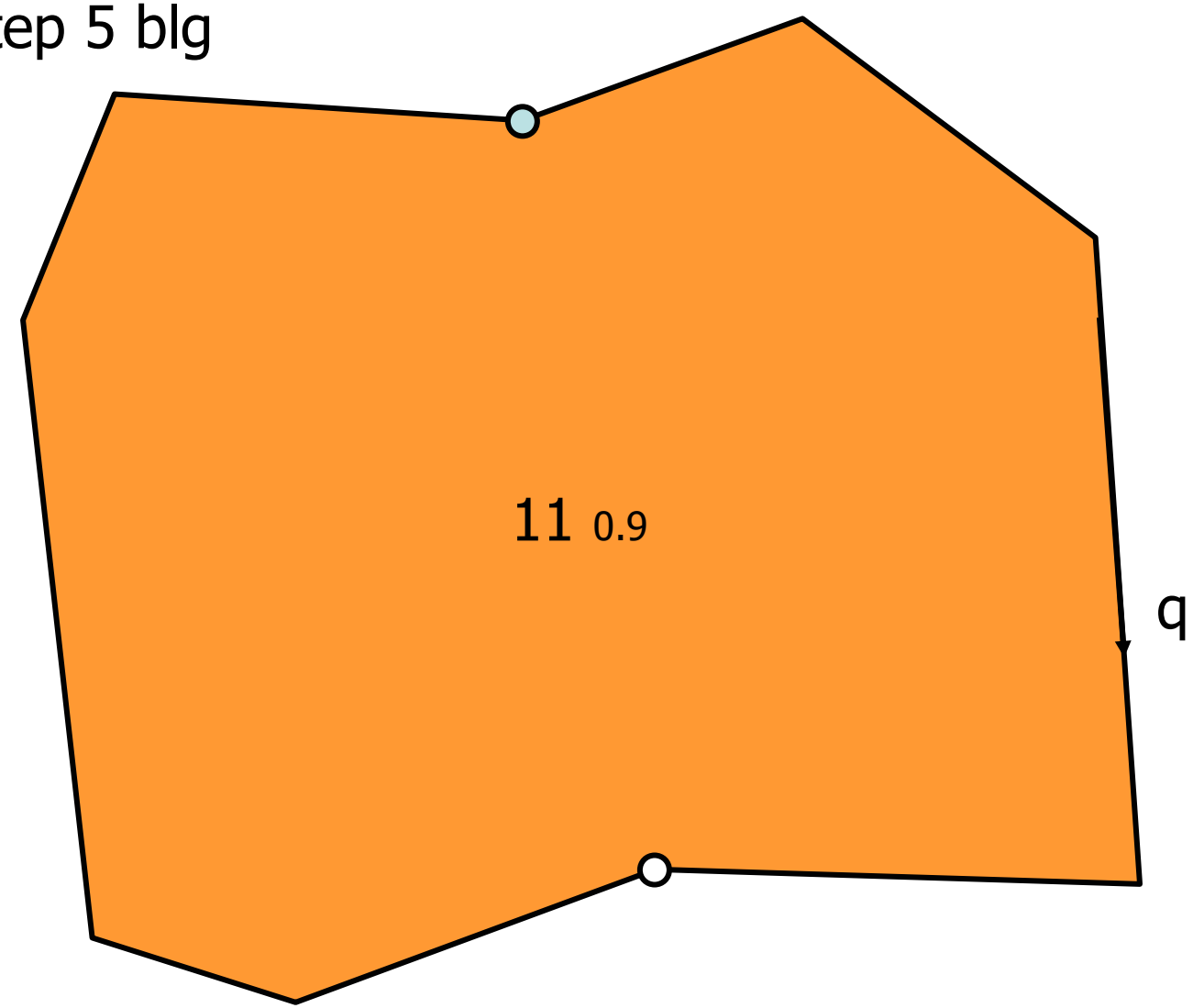
TU Delft

Step 5

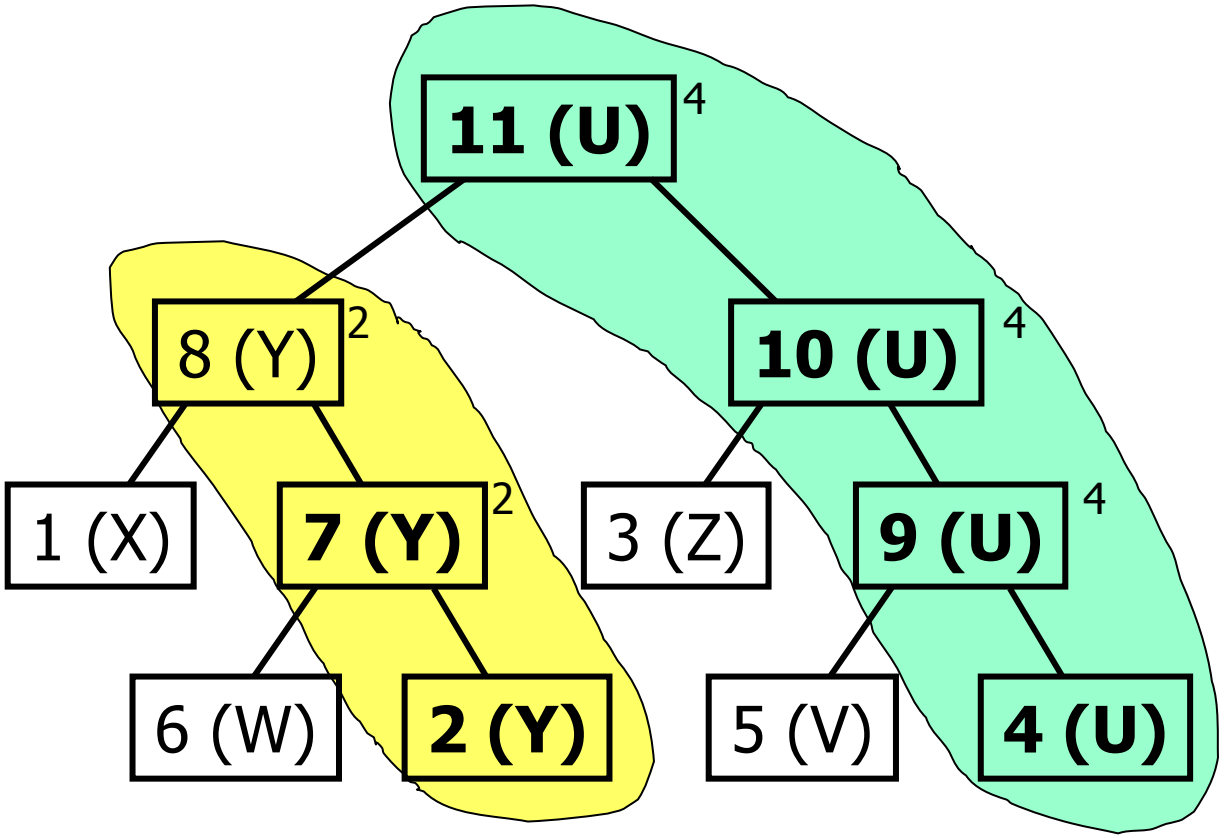


TU Delft

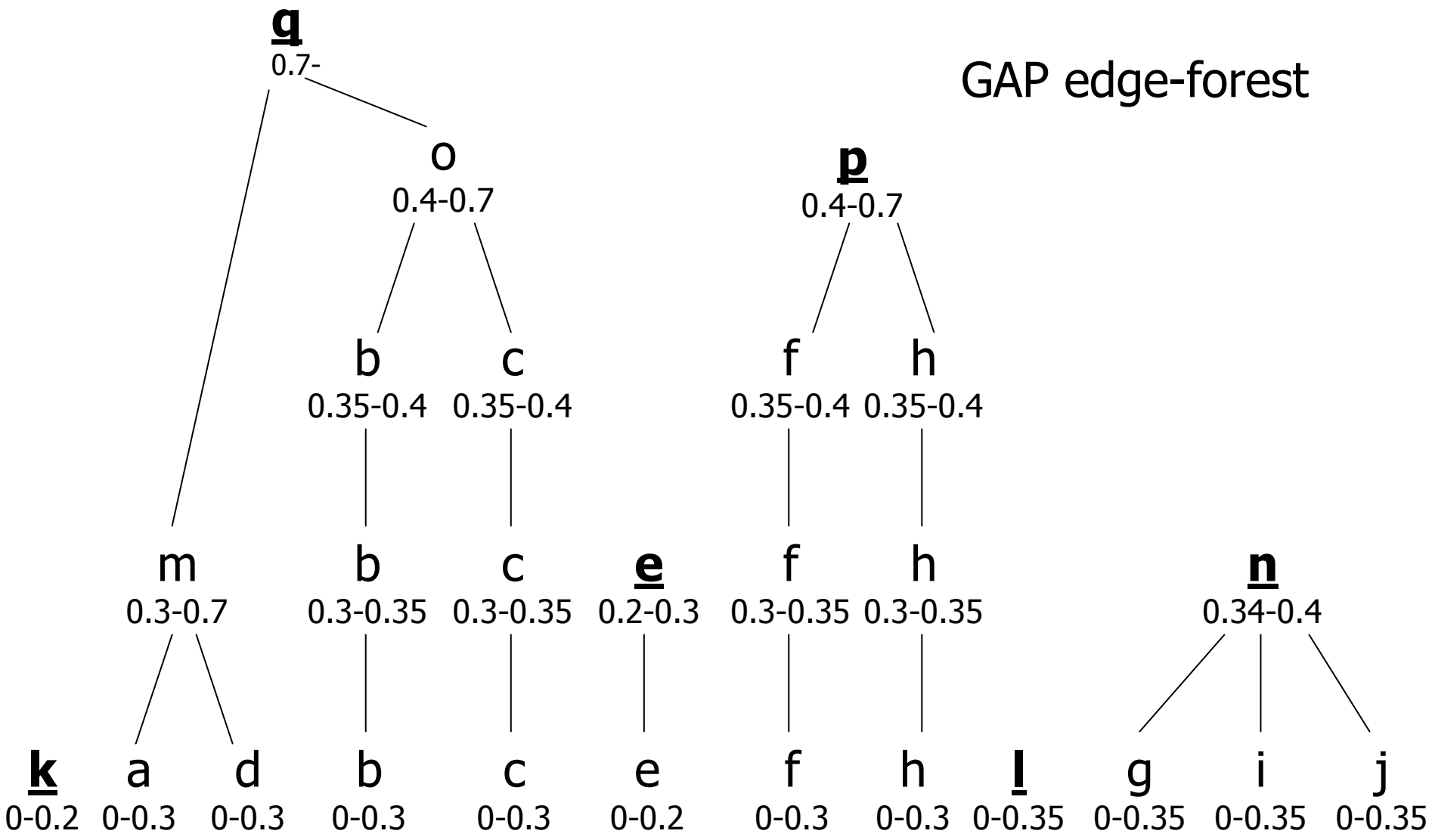
Step 5 blg



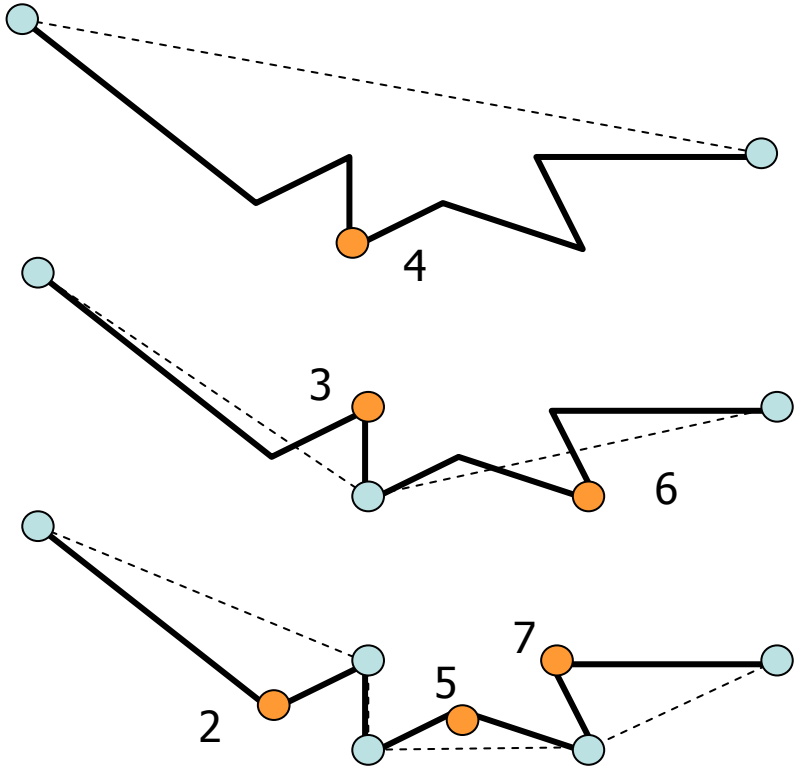
GAP face-tree

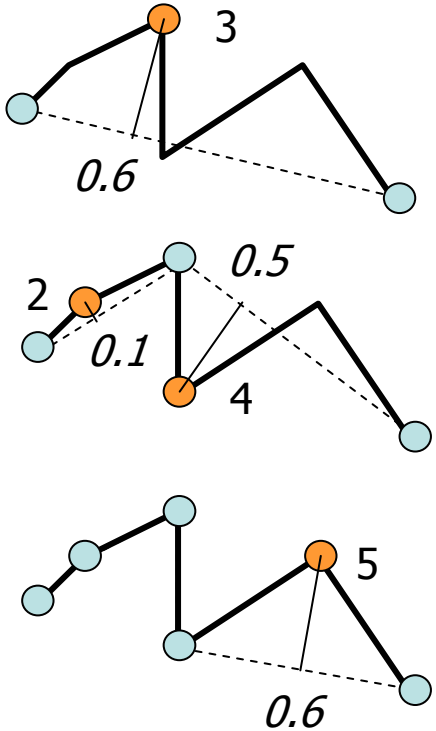


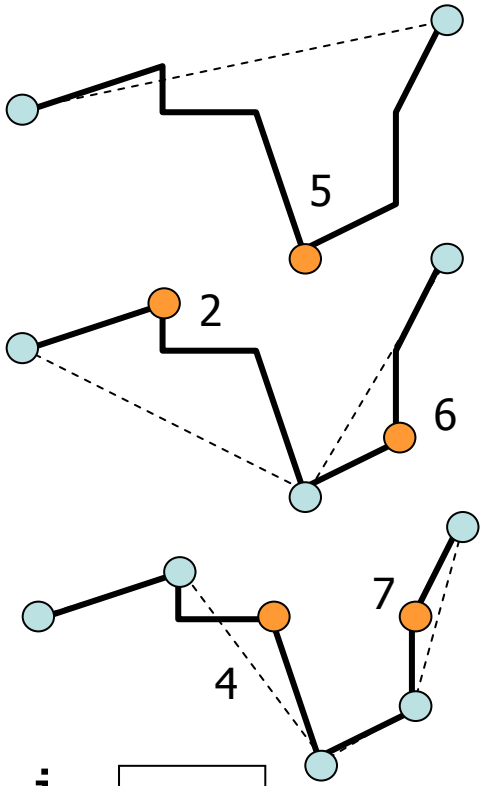
GAP edge-forest



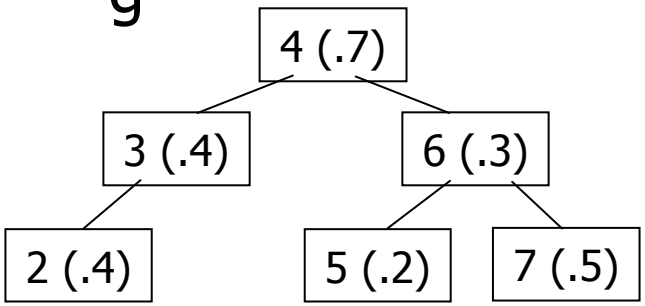
TU Delft


4
3
6
2
5
7

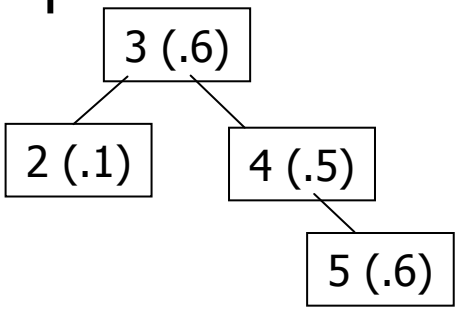

3
0.6
2
0.5
0.1
4
5
0.6


5
2
6
7
4

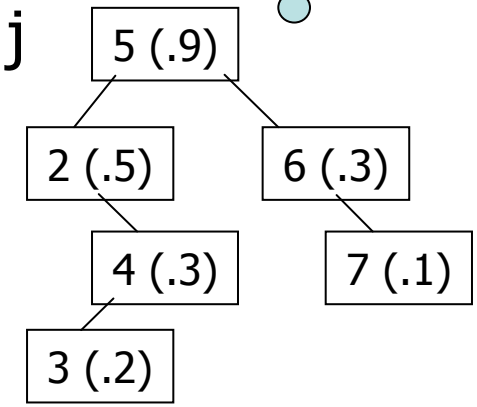
g


4 (.7)
3 (.4)
6 (.3)
2 (.4)
5 (.2)
7 (.5)

i

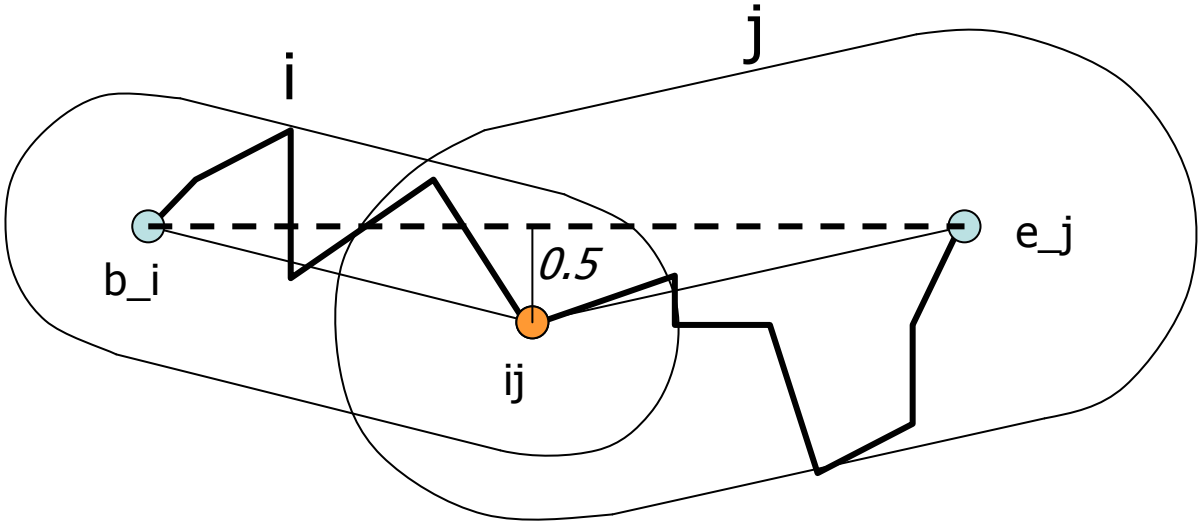

3 (.6)
2 (.1)
4 (.5)
5 (.6)

j


5 (.9)
2 (.5)
6 (.3)
4 (.3)
7 (.1)
3 (.2)

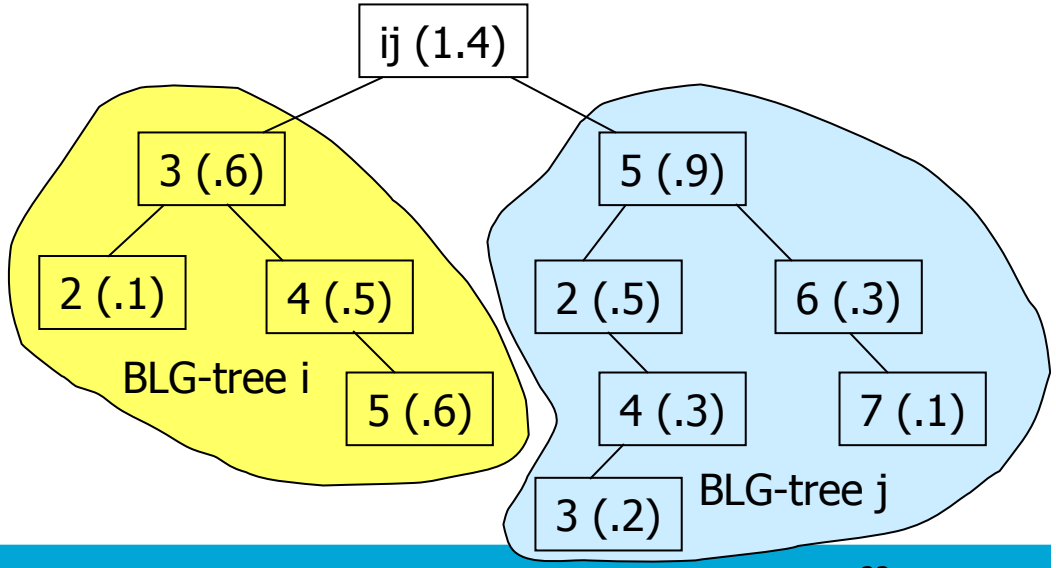
TU Delft

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



TU Delft

Contents

1. Introduction
2. Background tGAP structure
3. First implementation
4. Client-server set-up and progressive refinement
5. Conclusions

TU Delft

3. First implementation

- 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 functions) to avoid redundant storage, but to provide 'easy access'
- Functional index (3D R-tree: 2D box+imp range)

- Different from earlier 'theory'
  1. tgap_node table added (last redundancy+BLG-tree)
  2. No winged edge, only left/right refs (less storage)
  3. Signed references when merging BLG-trees

TU Delft

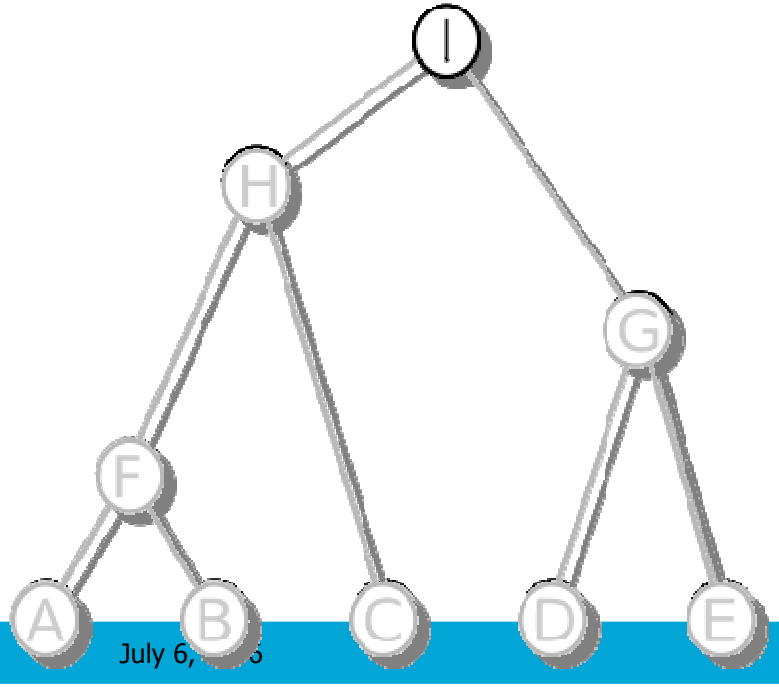
3: tGAP structure: combination of structures

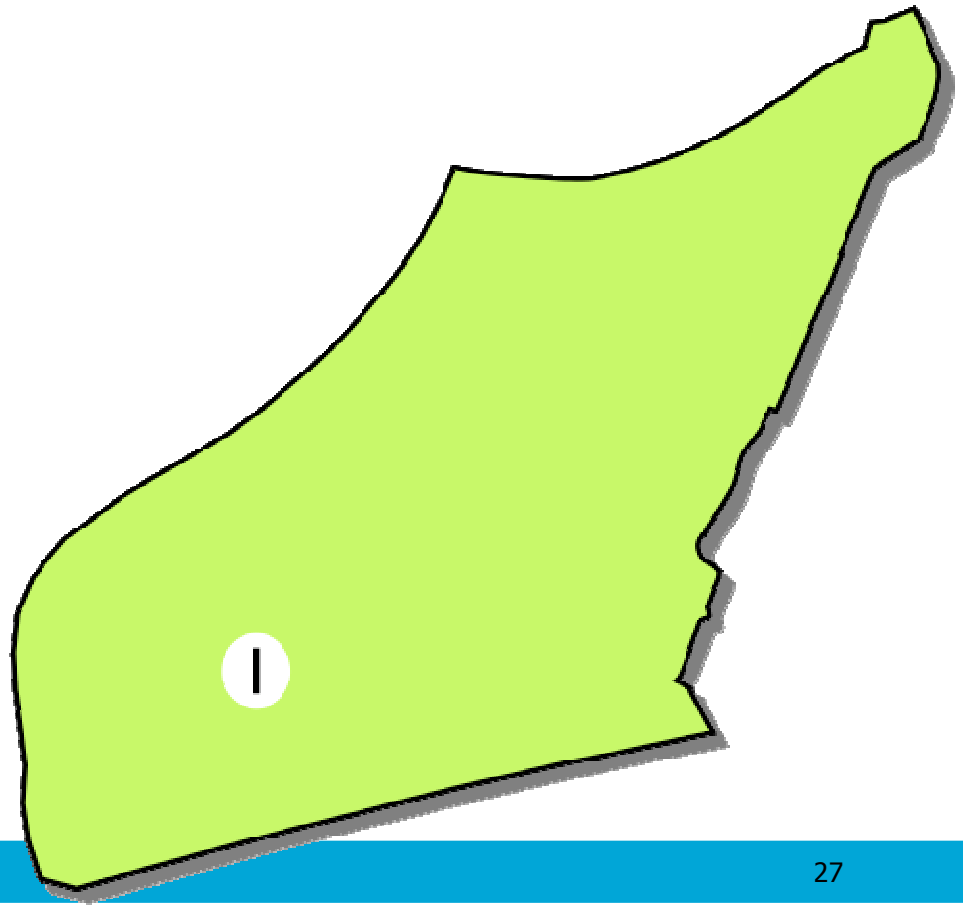
- Uses topology
- Stores results of Generalization
- Suitable for Area Partitioning

GAP face tree	allow face selection
GAP edge forest	allow line selection
BLG tree	allow line simplification
3D R-tree	allow fast selection

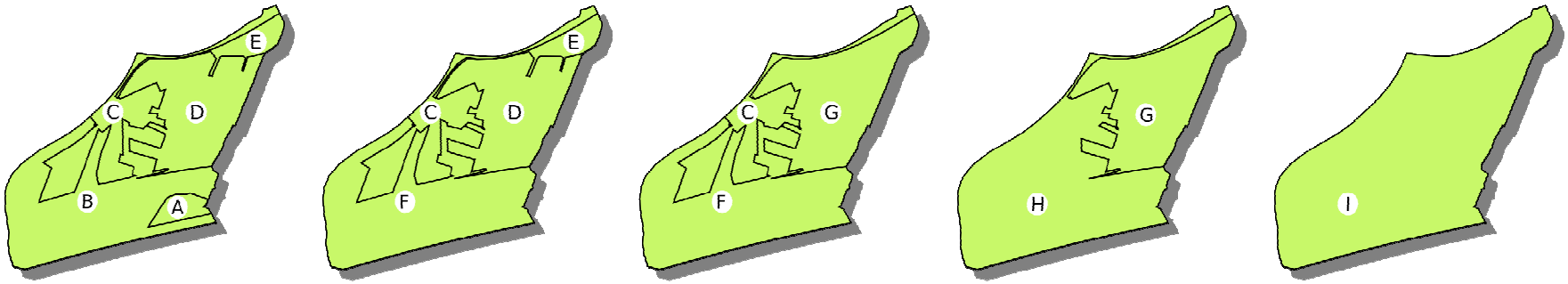
TU Delft

3. Constructing GAP face tree



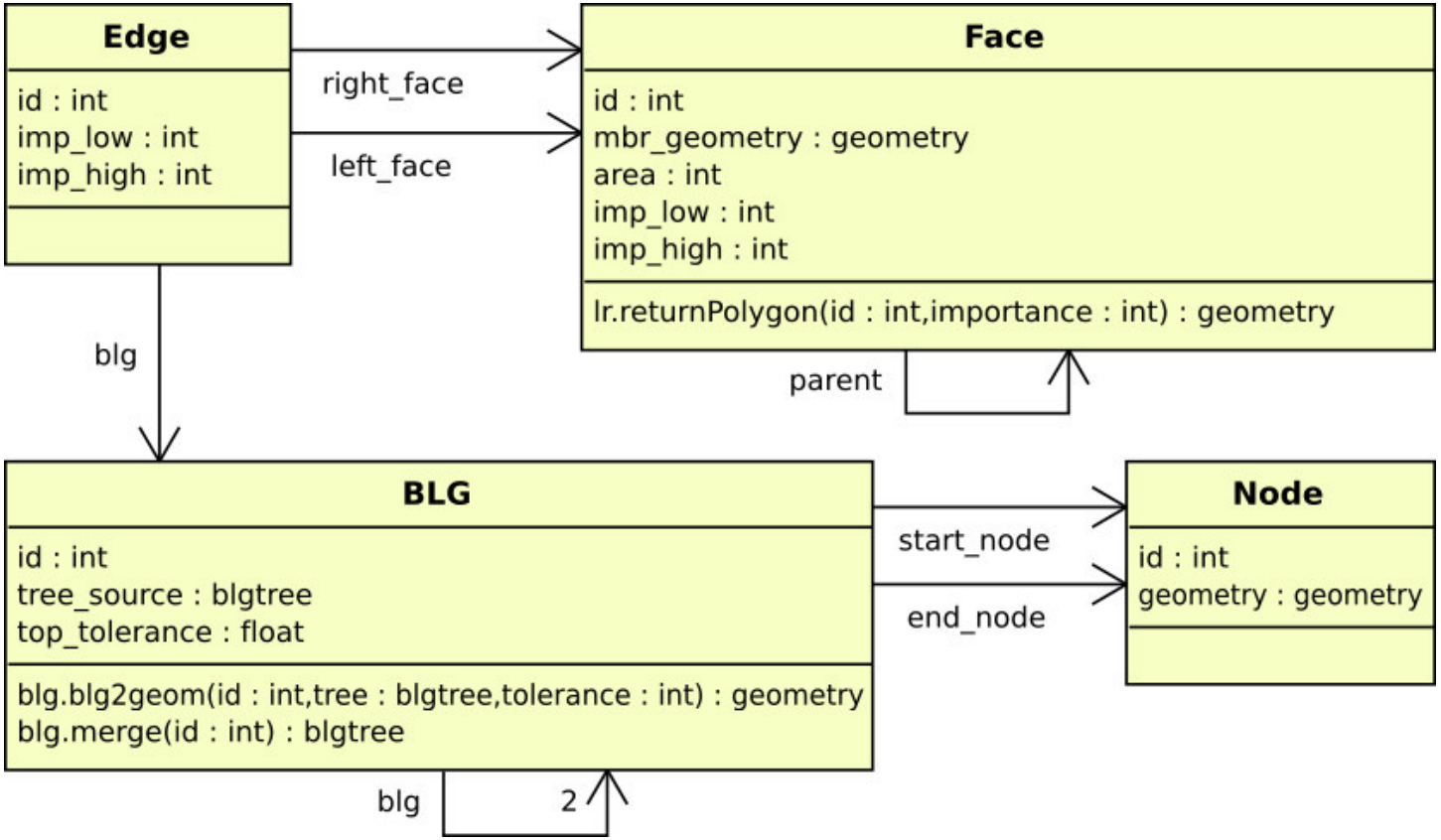


TU Delft



face id	imp l	imp h	area	mbr	parent
A	0	1	...	x_l, y_l, x_h, y_h	F
B	0	1	...	x_l, y_l, x_h, y_h	F
C	0	3	...	x_l, y_l, x_h, y_h	H
D	0	2	...	x_l, y_l, x_h, y_h	G
E	0	2	...	x_l, y_l, x_h, y_h	G
F	1	3	A+B	$A \cup B$	H
G	1	4	D+E	$D \cup E$	I
H	1	4	F+C	$F \cup C$	I
I	1	5	H+G	$H \cup G$	-

3. UML class diagram tGAP structure



3. tGAP storage requirements

- Several test datasets (small/medium/large): cadastral and topographic data (1:1.000-1:10.000)
- Plain (base scale) polygon storage 82 Mb
- Lean topology (base scale storage) 107 Mb (fact 1.3, note that Oracle spatial topology requires fact 3.0)
- Current tGAP (vario scale storage) 491 Mb (fact 6.0)

	#face/ Mb	*#edge/ Mb*	*#blg/ Mb*	*#node/ Mb*	*Total Mb*
Basic topology	170.368/ 2	418.530/ 94	-/ 0	281.216/ 11	107
tGAP structure	340.735/ 56	7.113.680/ 291	658.219/ 133	281.216/ 11	491

TU Delft

3. tGAP storage improvements

- tgap_face: less attributes; area, mbr, perhaps parent..
- tgap_edge 'explodes': 17 times more than base edges, many versions of same edge (at different imp levels). However only few attributes change left, right, imp → all versions of edge in same record+varray's for variable attributes
- tgap_blg: implicit BLG-tree or even polyline
- tgap_blg: separate table for leaf and non leaf records

- Expected size mean and lean tGAP: factor 3

TU Delft

3. tGAP initial visualization: polygons at arbitrary scale in Google Earth

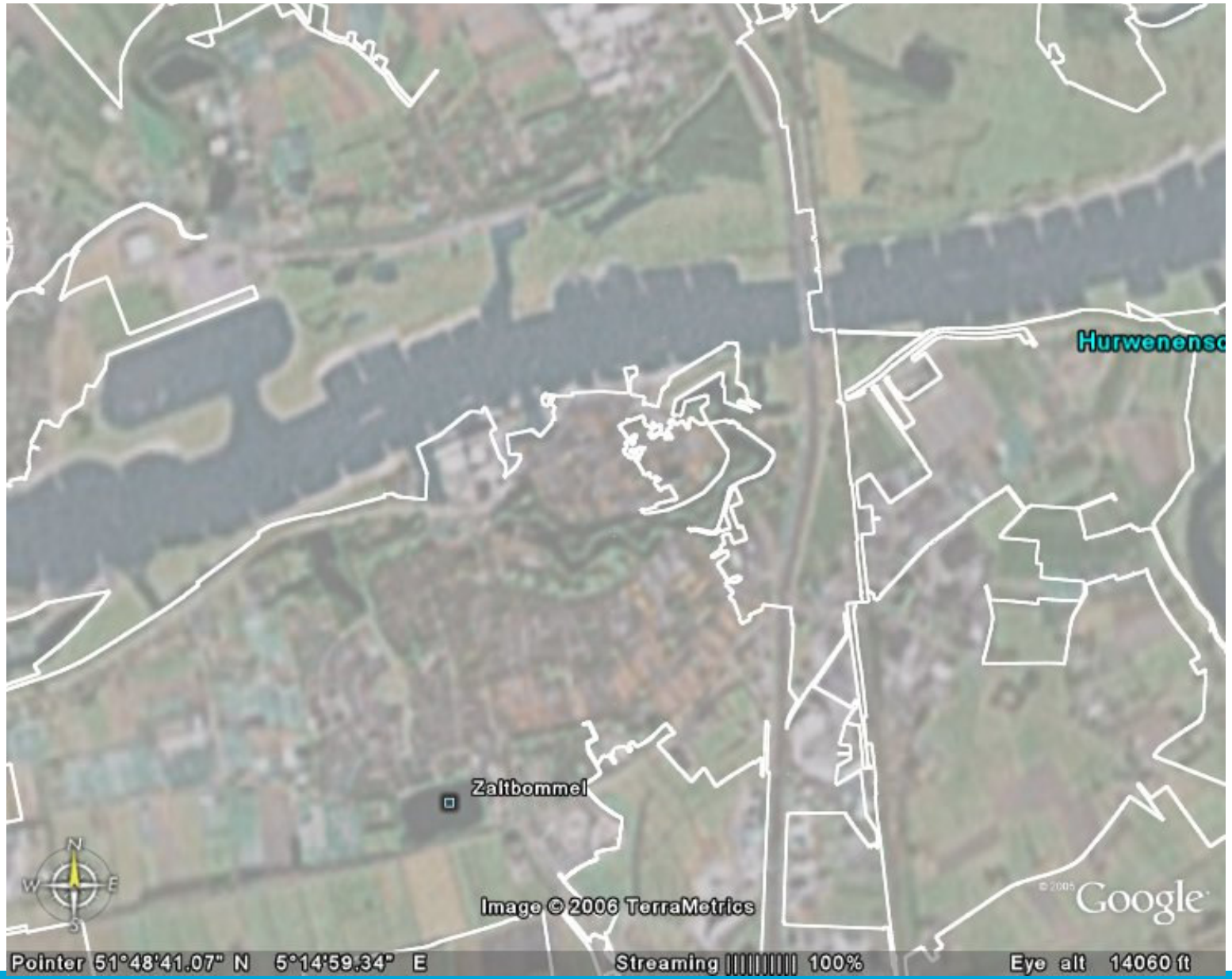
1. DBMS Server: Oracle spatial with tGAP as discussed → Polygons generated for arbitrary importance and tolerance (BLG-tree)
2. Middleware (Apache web server + Python, GDAL): coord transf, KML → Polygons transformed
3. Frontend: Google Earth → Polygons visualized

- Communication:
  - 2←→3: HTTP get/KML and
  - 1←→2: OCI (query, result set)

TU Delft


Image © 2006 GeoContent
Image © 2006 TerraMetrics
Google
Pointer 51°48'40.85" N 5°15'16.88" E
Streaming 100%
Eye alt 54.37 mi

TU Delft


Hurwenensc
Zaltbommel
N
W
E
S
Image © 2006 TerraMetrics
© 2005 Google
Pointer 51°48'41.07" N 5°14'59.34" E
Streaming 100%
Eye alt 14060 ft

TU Delft


Image © 2006 TerraMetrics
©2005 Google
Pointer 51°48'49.00" N 5°15'06.88" E
Streaming 100%
Eye alt 5483 ft

TU Delft


N
W
E
S
Image © 2006 TerraMetrics
©2005 Google
Pointer 51°48'40.95" N 5°15'17.50" E
Streaming 100%
Eye alt 1478 ft

TU Delft

Contents

1. Introduction
2. Background tGAP structure
3. First implementation
4. Client-server set-up and progressive refinement
5. Conclusions

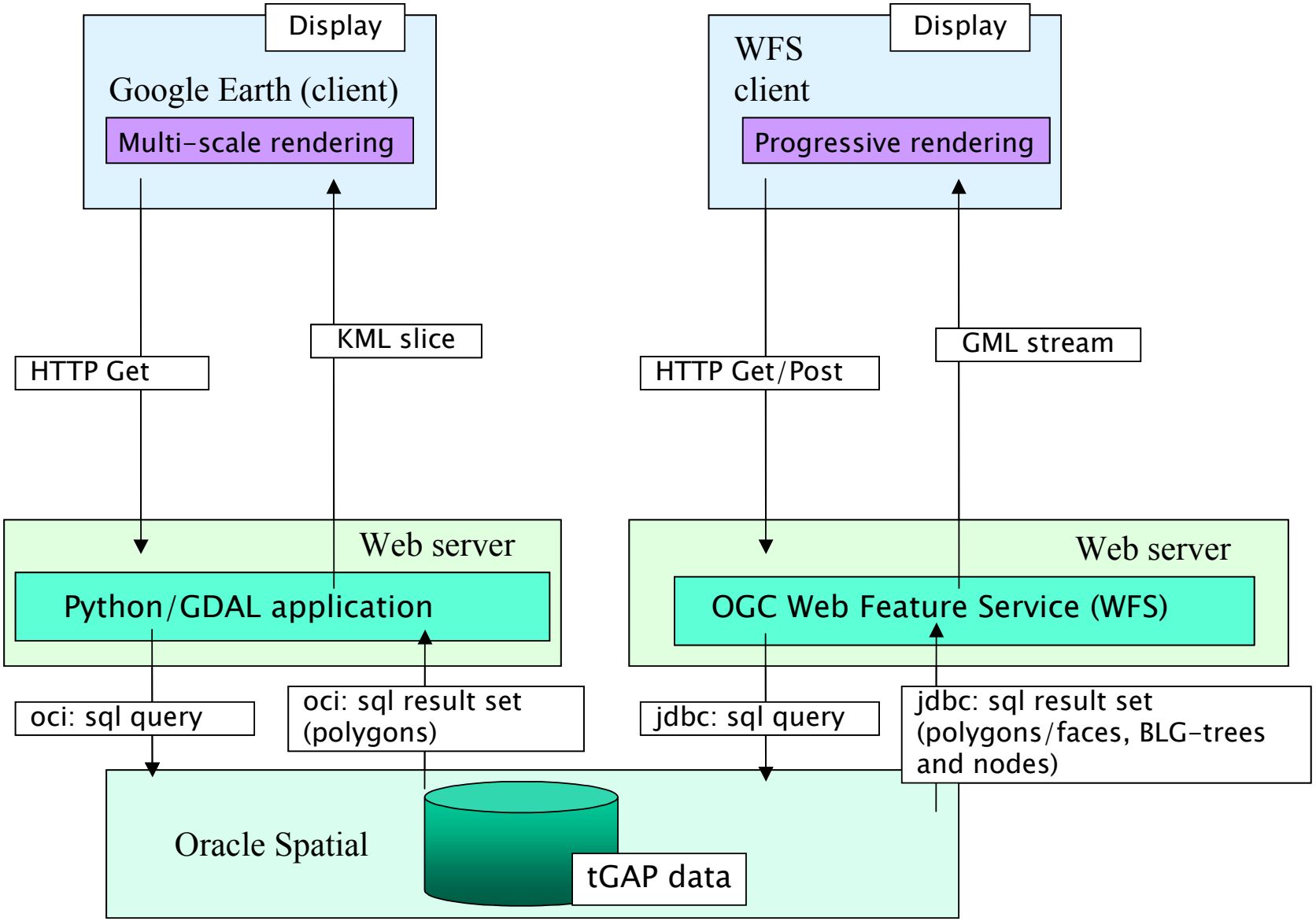
TU Delft

4. Polygons or structure?

- Current implementation has focus on server
- Client gets only polygons:
  1. No topology structure
  2. No progressive refinement

- Polygons are requested for every wanted scale (importance)
- Improvements for progressive transfer:
  1. Send importance range polygons (sorted) → smart client
  2. Send tGAP structure → needs smarter client (tGAP aware)

TU Delft


Display
Google Earth (client)
Multi-scale rendering
Display
WFS
client
Progressive rendering
HTTP Get
KML slice
HTTP Get/Post
GML stream
Web server
Python/GDAL application
Web server
OGC Web Feature Service (WFS)
oci: sql query
oci: sql result set
(polygons)
jdbc: sql query
jdbc: sql result set
(polygons/faces, BLG-trees
and nodes)
Oracle Spatial
tGAP data

TU Delft

4. Streaming of importance range (and first compared with a cut)

- A cut (or slice) of single importance

```
select face_id as id, '101' as impLevel,
  RETURN_POLYGON(face_id, 101) as geom
from tgap_face
where imp_low <= 101 and 101 < imp_high;
```

- A ordered range of importance values

```
select face_id as id, imp_high-1 as impLevel,
  imp_low, imp_high,
  RETURN_POLYGON(face_id,imp_high-1) as geom
from tgap_face
where imp_high > 90
order by imp_high desc;
```


TU Delft

4. Smart client for polygon range

- Alternatives:
  - Render step by step: start with most coarse polygon, then replace it by its two children. Repeat this step when receiving more detailed polygons.
  - Collect polygons for a while and render at a number of larger steps (and morph between steps).
  - Wait until everything has arrived and draw at appropriate scale
- The cached range (imp) of polygons can also be used at client side for (smart) zooming.
- Note no topology used and also no line simplification

4. Extension to OGC/ISO WFS

- It is possible to specify imp range in Filter part of GetFeature request and using ogc:SortBy
- Not ideal because it is not clear that this is about scale, streaming, progressive transfer/refinement
- Deeper integration in WFS (called WFS-R):
  1. GetCapabilities should indicate if server supports progressive refinement
  2. Reporting of the min and max imp of a theme
  3. New request type GetFeatureByImportance

TU Delft

4. Example WFS-R request

```
<wfs:GetFeatureByImportance service="WFS"
 version="1.0.0" outputFormat="GML2" ...>
 <wfs:Query typeName='tgap_face' minImp='5' maxImp=`8'>
  <ogc:Filter>
   <ogc:BBOX>
    <ogc:PropertyName>geom</ogc:PropertyName>
    <gml:Box srsName="…epsg.xml#28992">
     <gml:coordinates>
      136931,416574 139382,418904
     </gml:coordinates>
    </gml:Box>
   </ogc:BBOX>
  </ogc:Filter>
  <ogc:SortBy>gdmc:imp_high D</ogc:SortBy>
 </wfs:Query>
</wfs:GetFeatureByImportance>
```


TU Delft

Contents

1. Introduction
2. Background tGAP structure
3. First implementation
4. Client-server set-up and progressive refinement
5. Conclusions

TU Delft

5. Conclusions, main results

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

TU Delft

5. Conclusion: progressive refinement based on tGAP structure

- 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. Conclusions, improvements

- Generalization is application (task) dependent
  → more than 1 tGAP structure on same base topol (compare to multiple indices on same table)
- 'Bug': different edges of narrow features may cross when generalizing → avoid this during creation by tests (and state corresponding correct imp/tol value)
- Benchmarks 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

TU Delft

5. 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. Also combine 2 less important lines in 1 (e.g. after removal of least important branch)
- 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).

TU Delft