Playing with Kruskal: algorithms for morphological trees in edge-weighted graphs

Laurent Najman Jean Cousty Benjamin Perret

Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge, A3SI, ESIEE

Tuesday 28 may 2013

Highlights

Results

- A quasi-linear algorithm that computes a *binary partition tree by altitude ordering*

- Three linear post-processing algorithms that compute
  - hierarchy of quasi-flat zones
    - also known as the α-tree
    - also known as the Fuzzy Connectedness hierarchy
  - (hierarchies of) watershed cuts
  - hierarchies by increasing attributes
    - constrained connectivity hierarchies or
    - watershed-based hierarchies.

Outline

Outline

Minimum Spanning Tree

The Minimum Spanning Tree (MST) T is a connected spanning graph of the graph G such that the weight of T:

$$F(T) := \sum_{e \in E(T)} F(e)$$

is the least possible weight for a connected spanning subgraph of G.

Kruskal algorithm for MST (High-Level View)

- create a forest $\mathcal{F}$ (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty and $\mathcal{F}$ is not yet a single tree
  - remove an edge with minimum weight from S
  - if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
  - otherwise discard that edge.

At the termination of the algorithm, the forest has only one component and forms a minimum spanning tree of the graph.

The *disjoint set* problem

The disjoint set problem consists in maintaining a collection $\mathcal{Q}$ of disjoint sets under the operation of union.
Each set Q in $\mathcal{Q}$ is represented by a unique element of Q, called the *canonical element*.

- $\texttt{MakeSet}(q_1)$
- $\texttt{FindCanonical}(q_1)$
- $\texttt{Union}(q_1, q_2)$

Kruskal algorithm for MST (Implementation)

Data: An edge-weighted graph (V, E, F).
Result: A minimum spanning tree MST
Result: A collection $\mathcal{Q}$

`// Collection` $\mathcal{Q}$ `is initialized to` $\emptyset$

$e := 0$
for *all* $x_i \in V$ **do** MakeSet(i);
for *all edges* $\{x, y\}$ *by (strict) increasing weight* $F(\{x, y\})$ **do**
    $c_x := \mathcal{Q}$.FindCanonical(x); $c_y := \mathcal{Q}$.FindCanonical(y)
    if $c_x \neq c_y$ **then**
        $\mathcal{Q}$.Union(c_x, c_y);
        MST[e] $:= \{x, y\}$; $e := e + 1$
    else DoSomething$(\{x, y\})$

Main question in Kruskal implementation

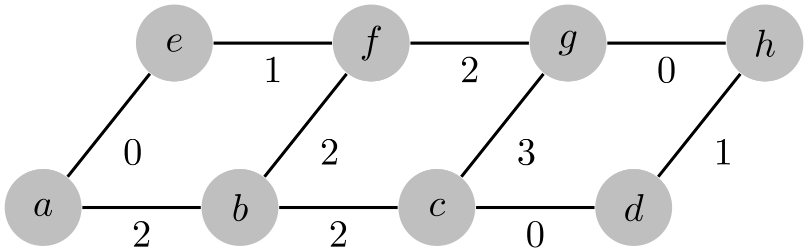
Question

How to represent and implement the collection $\mathcal{Q}$.

Answer

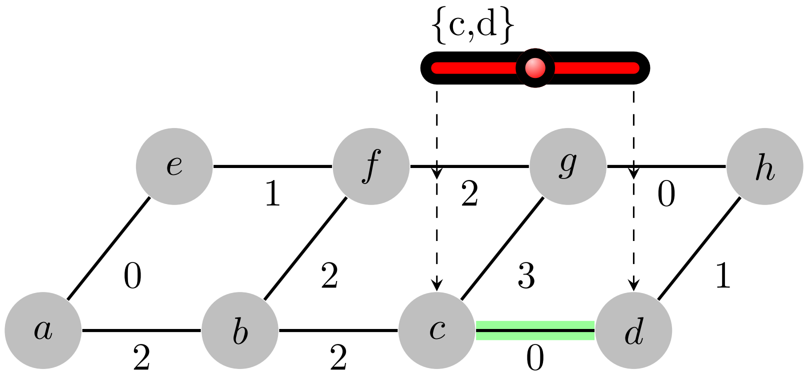
A good representation for $\mathcal{Q}$ is as a set of trees.

Binary Partition Tree by altitude ordering



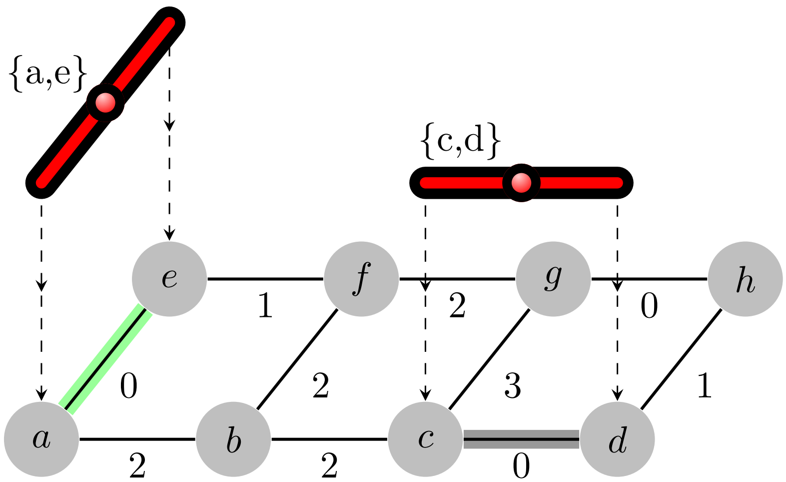
Edge-weighted graph

Binary Partition Tree by altitude ordering



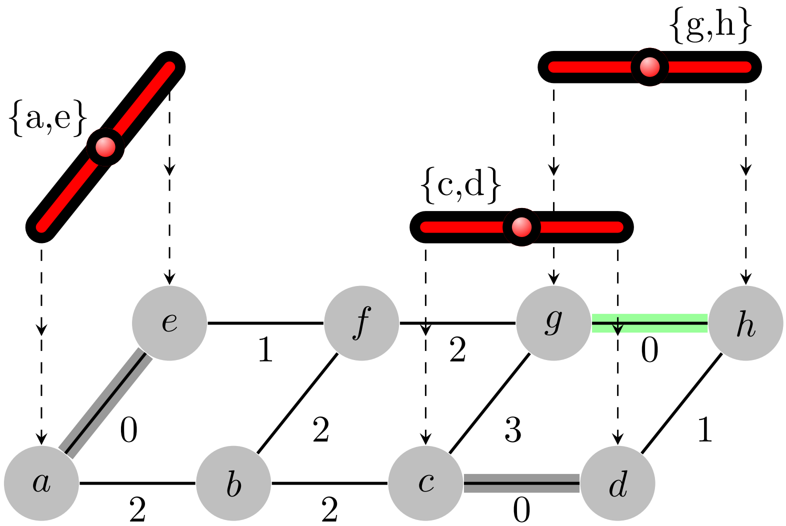
First edge-node

Binary Partition Tree by altitude ordering



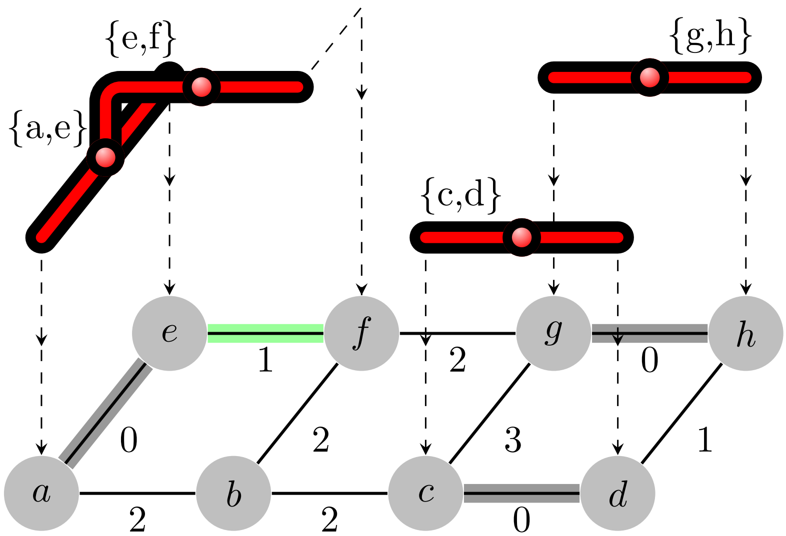
Second edge-node

Binary Partition Tree by altitude ordering



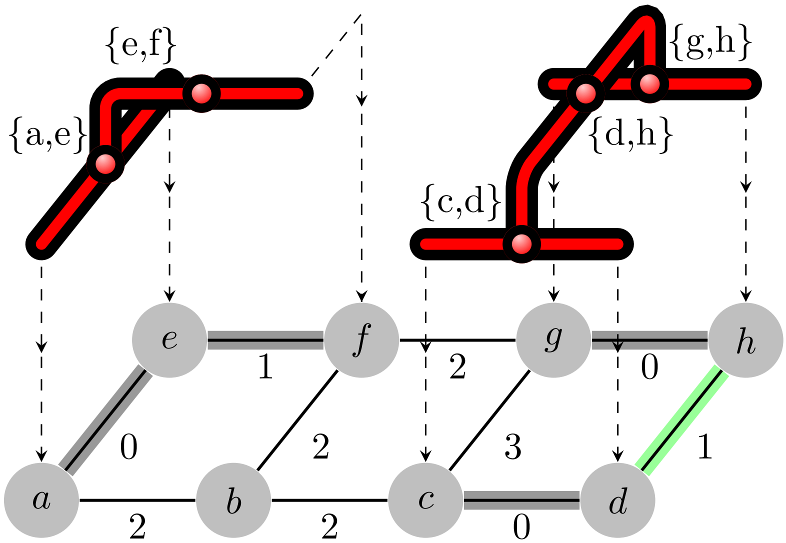
Third edge-node

Binary Partition Tree by altitude ordering



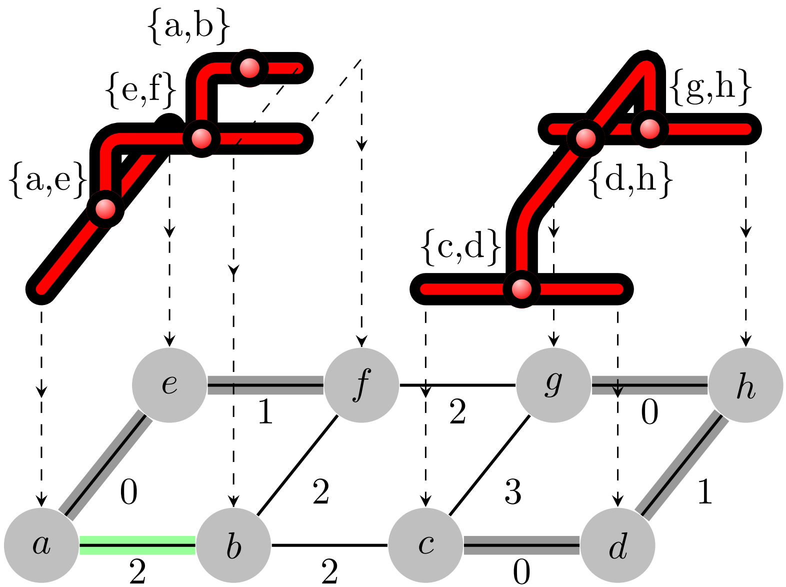
Fourth edge-node

Binary Partition Tree by altitude ordering



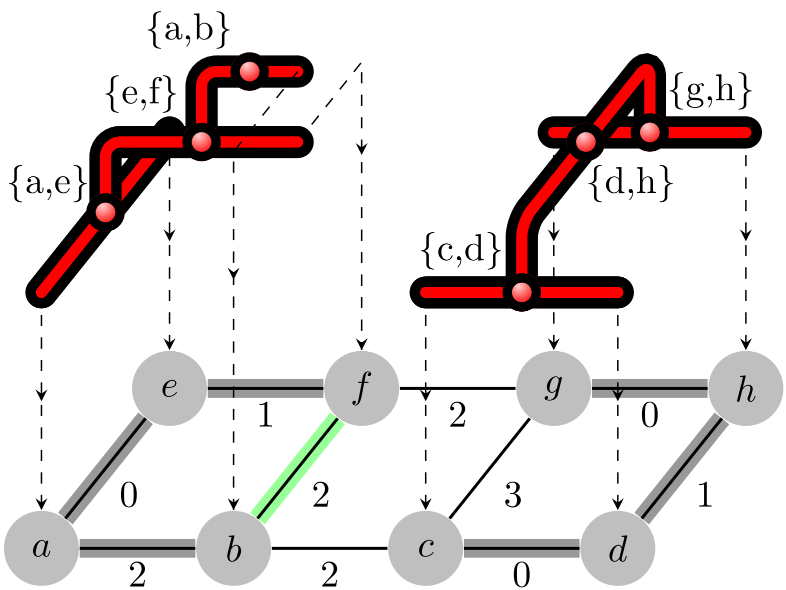
Fifth edge-node

Binary Partition Tree by altitude ordering



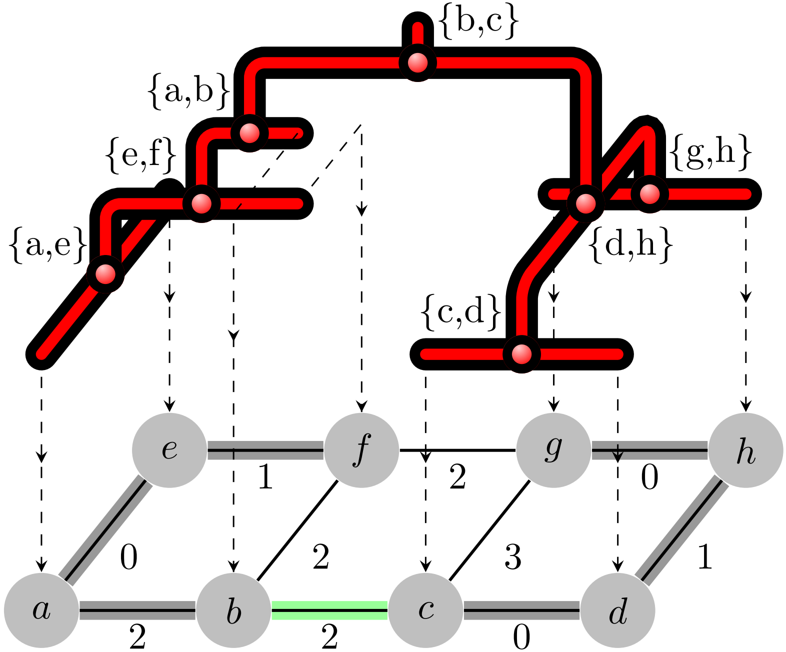
Sixth edge-node

Binary Partition Tree by altitude ordering



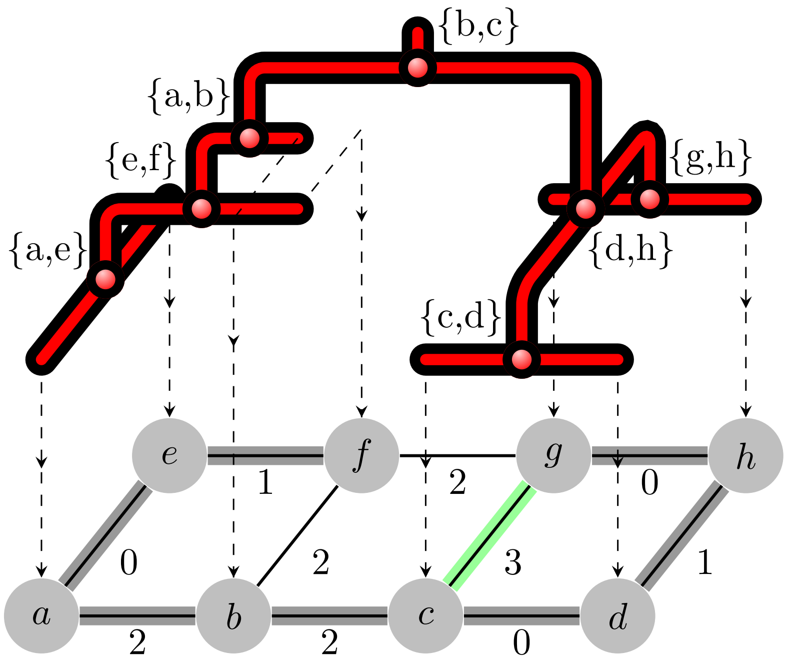
No new node

Binary Partition Tree by altitude ordering



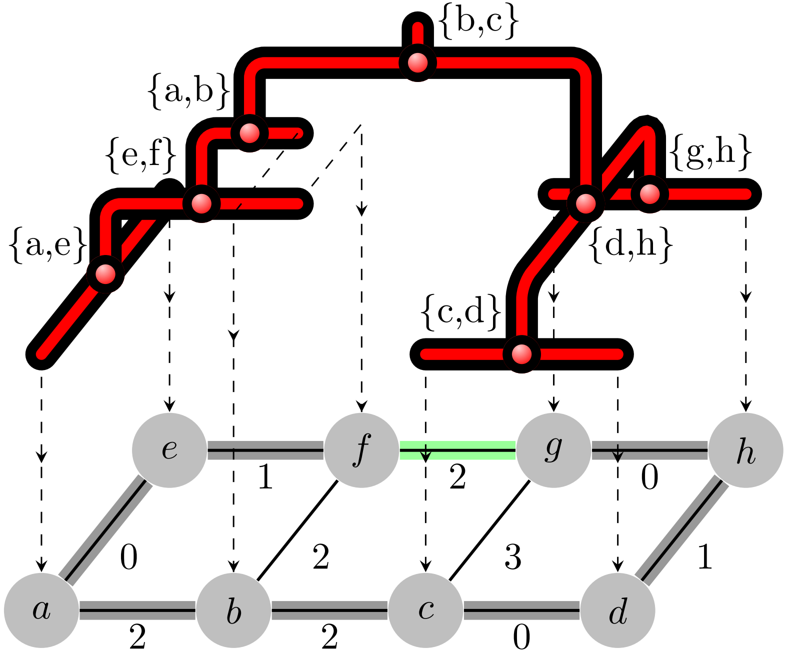
Seventh edge-node

Binary Partition Tree by altitude ordering



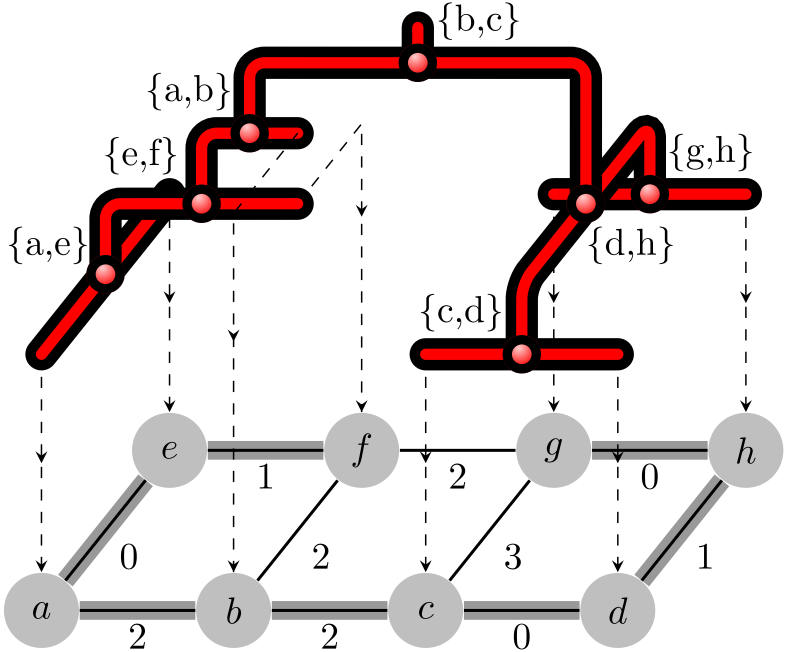
No new node

Binary Partition Tree by altitude ordering



No new node

Binary Partition Tree by altitude ordering



Final Q_{BT}

Q_{BT} Union-Find

Procedure Q_{BT}.MakeSet(q)

Q_{BT}.parent[q] $:= -1$; Q_{BT}.size $+= 1$;

Function Q_{BT}.FindCanonical(q)

while Q_{BT}*.parent[q]* ≥ 0 **do** $q :=$ Q_{BT}.parent[q];
return q;

Function Q_{BT}.Union(c_x, c_y)

Q_{BT}.parent[c_x]:=Q_{BT}.size; Q_{BT}.parent[c_y]:=Q_{BT}.size;
Q_{BT}.MakeSet(Q_{BT}.size);
return Q_{BT}*.size-1*;

Q_{BT} Union-Find

Interest

The produced tree is useful

Drawback

The algorithm is slow : $O(|V|^2)$

Tarjan Union-Find

Interest

Quasi-linear complexity

Drawback

The produced tree is not useful for our purpose

Tarjan Union-Find

Procedure $Q_T.\mathrm{MakeSet}(q)$

$Q_T.\mathrm{parent}[Q_T.\mathrm{size}] := -1;\ \ Q_T.\mathrm{Rnk}[Q_T.\mathrm{size}] := 0;\ Q_T.\mathrm{size} \mathrel{+}= 1;$

Function $Q_T.\mathrm{FindCanonical}(q)$

$r := q;$
while $Q_T.parent[r] \geq 0$ **do** $r := Q_T.\mathrm{parent}[r];$
while $Q_T.parent[q] \geq 0$ **do** $tmp := q; q := Q_T.\mathrm{parent}[q];$
$Q_T.\mathrm{parent}[tmp] := r;$

Function $Q_T.\mathrm{Union}(c_x,\ c_y)$

if $(Q_T.Rnk[c_x] > Q_T.Rnk[c_y])$ **then** $\mathrm{swap}(c_x, c_y);$
if $(Q_T.Rnk[c_x] == Q_T.Rnk[c_y])$ **then** $Q_T.\mathrm{Rnk}[c_y] \mathrel{+}= 1;$
$Q_T.\mathrm{parent}[c_x] := c_y;$
return $c_y;$

Q_{EBT}: Efficient Q_{BT} Union-Find

Interest

- Combination of both Q_{BT} and Q_T.
- Quasi-linear complexity.
- One of the produced trees, Q_{BT}, is useful.

Q_{EBT}: Efficient Q_{BT} Union-Find

Procedure Q_{EBT}.MakeSet(q)

Q_{EBT}.Root[q]:=q; Q_{BT}.MakeSet(q); Q_T.MakeSet(q);

Function Q_{EBT}.Union(c_x, c_y)

t_u:=Q_{EBT}.Root[c_x]; t_v := Q_{EBT}.Root[c_y];
Q_{BT}.parent[t_u] := Q_{BT}.parent[t_v] := Q_{BT}.size;
Q_{BT}.children[Q_{BT}.size].add($\{t_u\}$);
Q_{BT}.children[Q_{BT}.size].add($\{t_v\}$);
c:=Q_T.Union(c_x,c_y); `// Union in` Q_T `(with compression)`
Q_{EBT}.Root[c] := Q_{BT}.size; `// Update the root of` Q_{EBT}
Q_{BT}.MakeSet(Q_{BT}.size);
return Q_{BT}*.size-1*;

Function Q_{EBT}.FindCanonical(q)

return Q_T*.FindCanonical(q)*;

Outline

Some helper functions

Function getEdge(n)

Data: a (non-leaf) node n of Q_{BT}

Result: the edge e of the MST corresponding to the n^{th} node

return $n - |V|$;

Function weightNode(n)

Data: a (non-leaf) node of the tree

Result: the weight of the MST edge associated with the n^{th} node of Q_{BT}

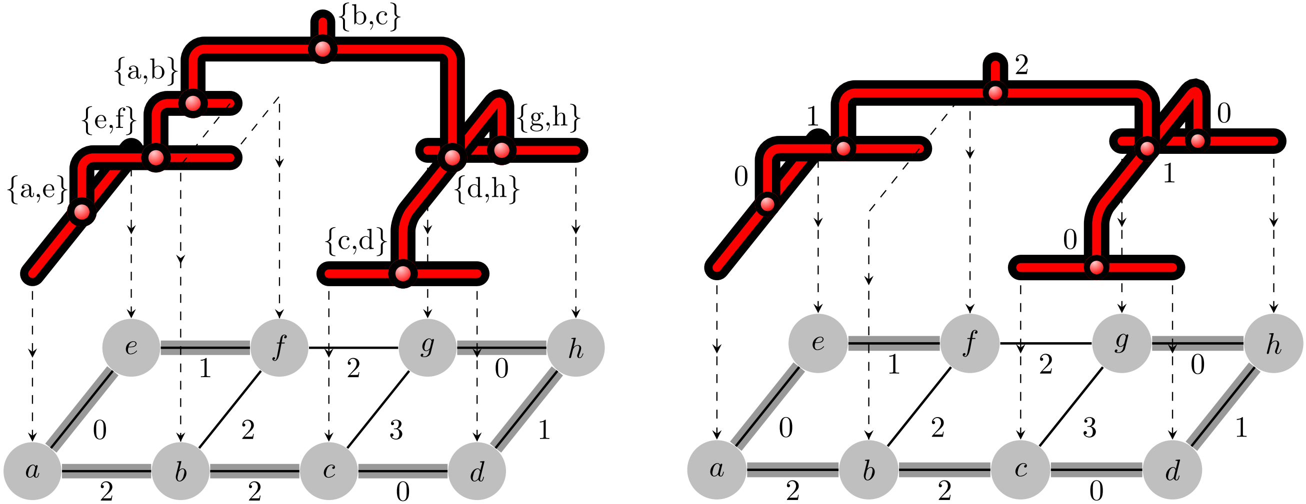
return $F(MST[getEdge(n)])$;

Outline

Q_{CT}: Quasi-flat zones hierarchy

- Also know as the α-tree.
- Also know as the Fuzzy Connectedness hierarchy.
- A quasi-linear algorithm: min-tree of the MST

Q_{CT}: Quasi-flat zones hierarchy



Q_{BT}

Q_{CT}

Quasi-flat zones hierarchy

Procedure CanonizeQ_{BT}

Data: Q_{BT}

Result: Q_{CT}, a canonized version of Q_{BT}

for *all nodes n of* Q_{BT} **do** Q_{CT}.parent[n]:=Q_{BT}.parent[n]; Q_{CT}.size+=1;

for *each non-leaf and non-root node n of* Q_{BT} *by decreasing order* **do**

  $p := Q_{CT}$.parent[n];

  if *(weightNode(p) == weightNode(n))* **then**

    for *all* $c \in Q_{BT}$*.children*[n] **do** Q_{CT}.parent[c]:=p;

    Q_{CT}.parent[n]:=n; `// Delete node` n `of` Q_{CT}

`// If needed, build the list of children`

for *all nodes n of* Q_{CT} **do**

  p:=Q_{CT}.parent[n]; **if** $p \geq 0$ *and* $p \neq n$ **then** Q_{CT}.children[p].add(n);

Quasi-flat zones hierarchy

Q_{BT} or Q_{CT}?

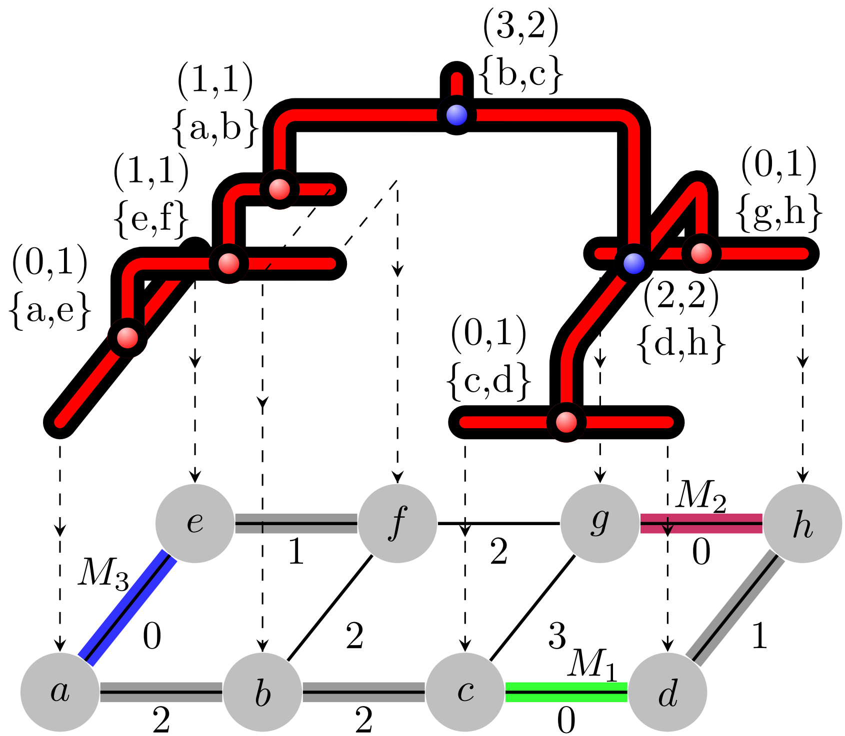
- It is possible to merge the min-tree algorithm with Kruskal's MST to obtain Q_{CT} in one step
- Q_{BT} contains more information than Q_{CT}
- The rest of the talk shows that computing Q_{CT} is not needed

Outline

Watershed cuts

- Partitions defined thanks to the *drop of water* principle
- Difficulty: non-uniqueness on flat zones (hence a choice)
- Also leads to a hierarchy (of watershed-cut partitions)
  - hierarchy by pass/connection value
  - (also known as Fuzzy connectedness)

Watershed cuts



Watershed cuts

Function watershed

Data: Q_{BT}

Result: A binary array *ws* indicating which MST edges are watershed

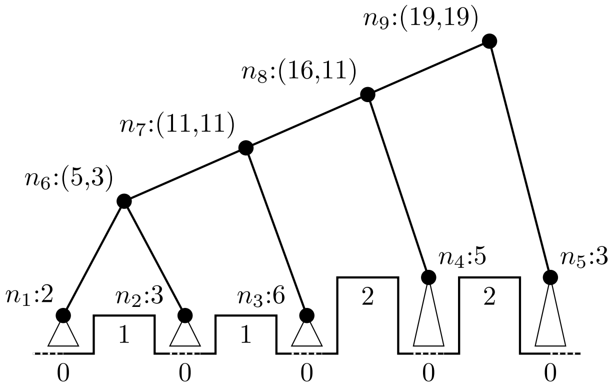
```
for all leaf-nodes n of Q_BT do minima[n]:=0;
for each non-leaf node n of Q_BT by increasing order do
    flag := TRUE; nb := 0;
    for all c ∈ Q_BT.children[n] do
        m := minima[c]; nb := nb + m;
        if (m == 0) then flag := FALSE;
    ws[getEdge(n)] := flag;
    if (nb ≠ 0) then minima[n] := nb;
    else
        if (n is the root of Q_BT) then minima[n] := 1;
        else
            p := Q_BT.parent[n];
            if (weightNode[n]<weightNode[p]) then minima[n] := 1;
            else minima[n]:=0;
```

Outline

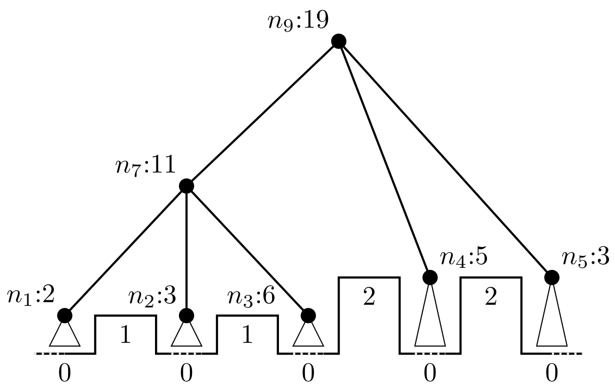
Attribute-based hierarchies

- Increasing attributes
- Watershed-cut framework
- Constrained-connectivity framework
  - The range criterion is indeed increasing
- Area-base, depth-based, volume-based hierarchies. . .
  - either from a watershed-cut hierarchy
  - or from a quasi-flat zone hierarchy

Attribute-based hierarchies



Q_{BT}



Q_{CT}

Attribute-based hierarchies

Function getAttribute(n)

Data: A node n of Q_{BT}

Result: The attribute at the time of the merging

```
if (n is the root) or (weightNode(parent[n]) ≠ weightNode(n)) then
    for all c children of n do  getAttribute(c);
    attribute[n] := attributeComp[n];
else
    max:=0;
    for all children c of n do
        v:=getAttribute(c);
        if v > max then max := v;
    attribute[n] := max;
return attribute[n];
```

Attribute-based hierarchies

Procedure ComputeMergeAttributeMST

Data: Q_{BT}

Result: a reweighted MST G corresponding to the attribute-based hierarchy

for *any non-leaf node n of* Q_{BT} **do**

$\quad a_1 := attribute[children[n].left]$;

$\quad a_2 := attribute[children[n].right]$;

$\quad G[getEdge(n)] := \min(a_1, a_2)$;

Conclusion

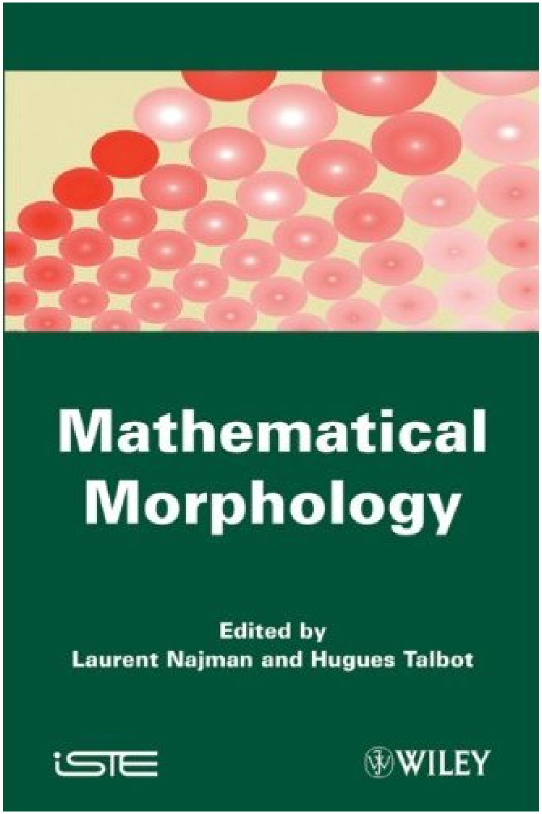
- Several elegant yet efficient algorithms for morphological trees
- Based on the Minimum Spanning Tree
- Other approaches than Kruskal can be used
- Unification theory in: Cousty, J., Najman, L., Perret, B.: *Constructive links between some morphological hierarchies on edge-weighted graphs*. (ISMM 2013).
- Source code at `http://www.esiee.fr/~info/sm/`

Summary of the poster content

	$\mathcal{PH}(G)$	$\mathcal{MH}(G)$	$\mathcal{PH}(T)$	$\mathcal{MH}(T)$	$\mathcal{Q}$	$\mathcal{B}_\prec$	$\mathcal{H}_S$
$\mathcal{PH}(G)$	$\Longleftrightarrow$	$\Longleftrightarrow$	$\Longrightarrow$	$\Longrightarrow$	$\Longrightarrow$	$\times$	$\times$
$\mathcal{MH}(G)$	$\Longleftrightarrow$	$\Longleftrightarrow$	$\Longrightarrow$	$\Longrightarrow$	$\Longrightarrow$	$\times$	$\times$
$\mathcal{PH}(T)$	$\Longleftarrow$	$\Longleftarrow$	$\Longleftrightarrow$	$\Longleftrightarrow$	$\Longrightarrow$	$\Longleftarrow$	$\times$
$\mathcal{MH}(T)$	$\Longleftarrow$	$\Longleftarrow$	$\Longleftrightarrow$	$\Longleftrightarrow$	$\Longrightarrow$	$\Longleftarrow$	$\times$
$\mathcal{Q}$	$\Longleftarrow$	$\Longleftarrow$	$\Longleftarrow$	$\Longleftarrow$	$\Longleftrightarrow$	$\Longrightarrow$	$\times$
$\mathcal{B}_\prec$	$\times$	$\times$	$\Longrightarrow$	$\Longrightarrow$	$\Longrightarrow$	$\Longleftrightarrow$	$\Longrightarrow$
$\mathcal{H}_S$	$\times$	$\times$	$\times$	$\times$	$\times$	$\Longleftarrow$	$\Longleftrightarrow$

Table 1: Summary of the main results.

Thank for your attention !



Pink: http://pinkhq.com
Olena: http://www.lrde.epita.fr/cgi-bin/twiki/view/Olena