Minimal Component-Hypertrees

Alexandre Morimitsu
Wonder A. L. Alves   Dennis J. Silva
Charles F. Gobber   Ronaldo F. Hashimoto

alexandre.morimitsu@usp.br

University of São Paulo

Universidade Nove de Julho

March 27, 2019

Introduction and Related Works

Introduction

- Mathematical Morphology;

Introduction

- Mathematical Morphology;
- Connected components (CCs) can give information about the characteristics of an object;

Introduction

- Mathematical Morphology;
- Connected components (CCs) can give information about the characteristics of an object;

Introduction

Components Trees (Salembier et al., 1998)

- A graph (tree) that represents the inclusion relation of connected components of level sets of an image;

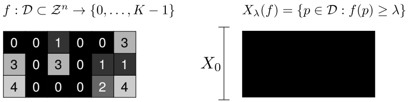
$f : \mathcal{D} \subset \mathcal{Z}^n \to \{0, \ldots, K-1\}$

0	0	1	0	0	3
3	0	3	0	1	1
4	0	0	0	2	4

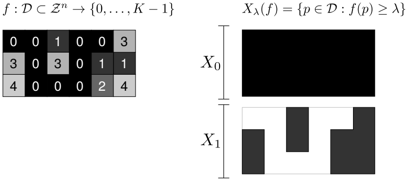
Introduction

Components Trees (Salembier et al., 1998)

- A graph (tree) that represents the inclusion relation of connected components of level sets of an image;



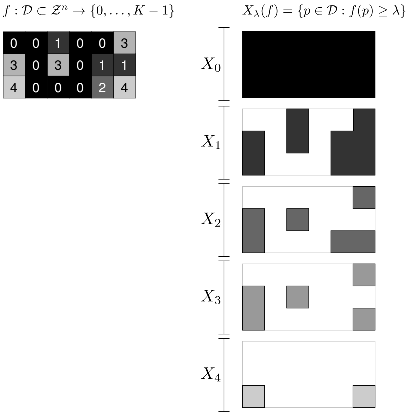
- A graph (tree) that represents the inclusion relation of connected components of level sets of an image;



Introduction

Components Trees (Salembier et al., 1998)

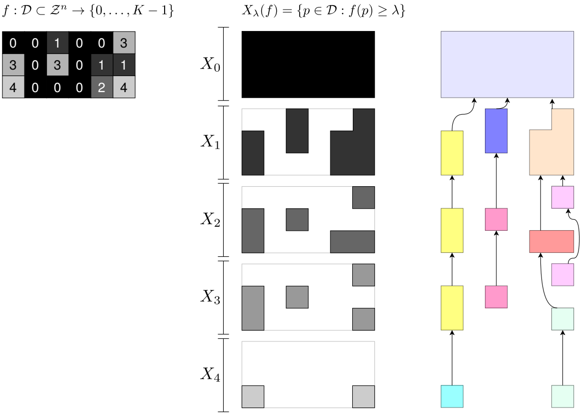
- A graph (tree) that represents the inclusion relation of connected components of level sets of an image;



Introduction

Components Trees (Salembier et al., 1998)

- A graph (tree) that represents the inclusion relation of connected components of level sets of an image;

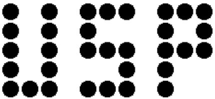


- Closely related to the the chosen connectivity;

Introduction

Connected Components

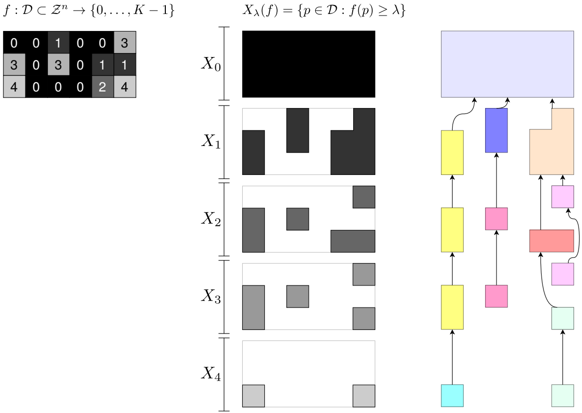
- Closely related to the the chosen connectivity;



- Groups of close objects can be considered as a single component;

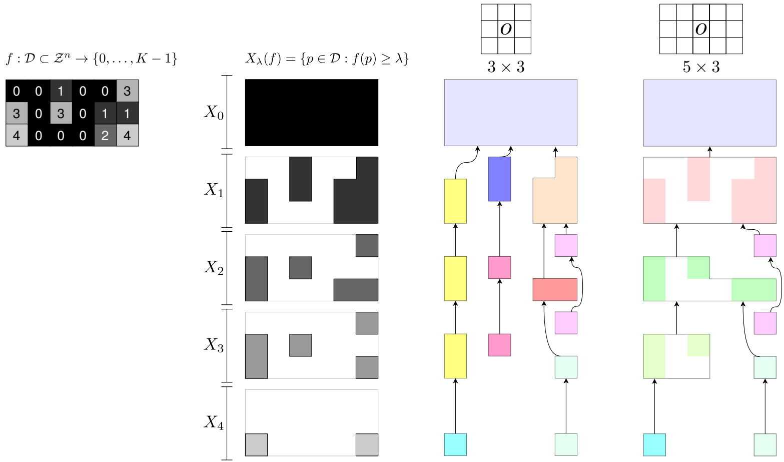
Introduction

Components Trees



Introduction

Components Trees



- Another hierarchy of connected components: multiple connectivities;

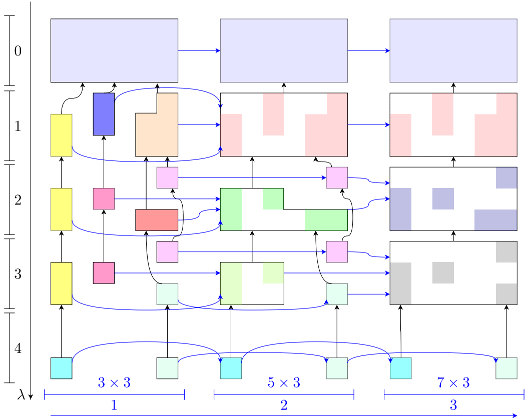
Introduction

Component-Hypertrees (Passat and Naegel, 2011)

- Another hierarchy of connected components: multiple connectivities;
- A bigger neighborhood may connect disjoint components built from a smaller neighborhood;

Introduction
Component-Hypertrees

- Graph that represents both inclusions based on level sets and neighborhoods: *component-hypertree*.



Introduction

Component-Hypertrees

- Not as widely adopted as component trees:

Introduction

Component-Hypertrees

- Not as widely adopted as component trees:
- Includes information of all individual component trees and inclusion of nodes from consecutive trees;

- Not as widely adopted as component trees:
- Includes information of all individual component trees and inclusion of nodes from consecutive trees;
- Cost in memory and processing times is multiplied by the number of neighborhoods;

Introduction

Component-Hypertrees

- Definition for mask-based connectivities (Passat and Naegel, 2011);
  - Focused on the theory of hypertrees;

Introduction

Component-Hypertrees

- Definition for mask-based connectivities (Passat and Naegel, 2011);
  - Focused on the theory of hypertrees;
- Efficient way of updating a tree for the next neighborhood (Morimitsu et al., 2015);

- Definition for mask-based connectivities (Passat and Naegel, 2011);
  - Focused on the theory of hypertrees;
- Efficient way of updating a tree for the next neighborhood (Morimitsu et al., 2015);
- To the best of our knowledge, there was no optimized way of storing hypertrees efficiently;

- Definition for mask-based connectivities (Passat and Naegel, 2011);
  - Focused on the theory of hypertrees;
- Efficient way of updating a tree for the next neighborhood (Morimitsu et al., 2015);
- To the best of our knowledge, there was no optimized way of storing hypertrees efficiently;
- This is the problem we want to solve;

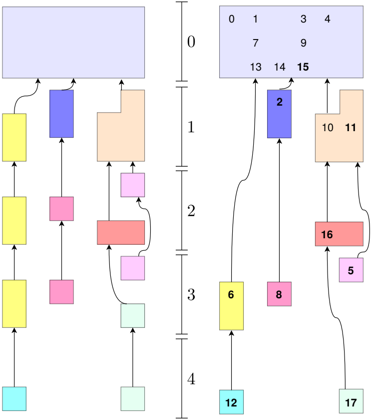
Background

- Max-tree: efficient way of implementing a component tree;

Theory

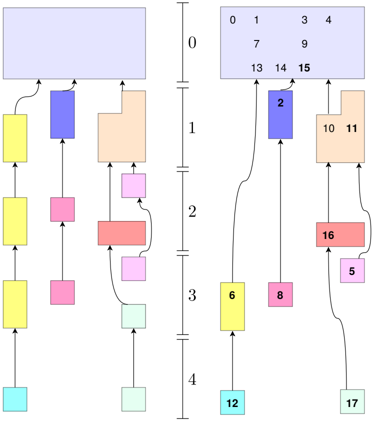
Max-tree

- Max-tree: efficient way of implementing a component tree;



Theory

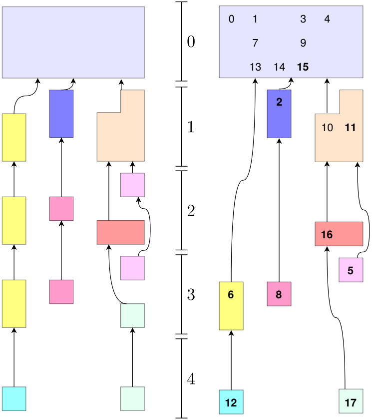
Max-tree



- Nodes are stored only once;
- Each pixel is stored only in the smallest node that contains it;
- Construction algorithm is optimized, i.e., it already allocate the nodes without repetition;

Theory

Max-tree



- Nodes are stored only once;
- Each pixel is stored only in the smallest node that contains it;
- Construction algorithm is optimized, i.e., it already allocate the nodes without repetition;

- We want a similar structure for component-hypertrees.

Naive approach

Simplified component-hypertree

- Naive approach: Build each max-tree separately;
- Merge nodes from consecutive trees;

Naive approach

Simplified component-hypertree

- Naive approach: Build each max-tree separately;
- Merge nodes from consecutive trees;
- Approach works, but it is not efficient:
  - Does not use previous computations;
  - Repeated nodes in different trees;

Proposed Method

Proposed approach

Simplified component-hypertree

- Proposed approach:

Proposed approach

Simplified component-hypertree

- Proposed approach:
  - Supposes a construction algorithm that uses previously computed max-tree and update them for the next neighborhood;
  - Keep track of changes to allocate only new nodes and arcs;

Algorithms

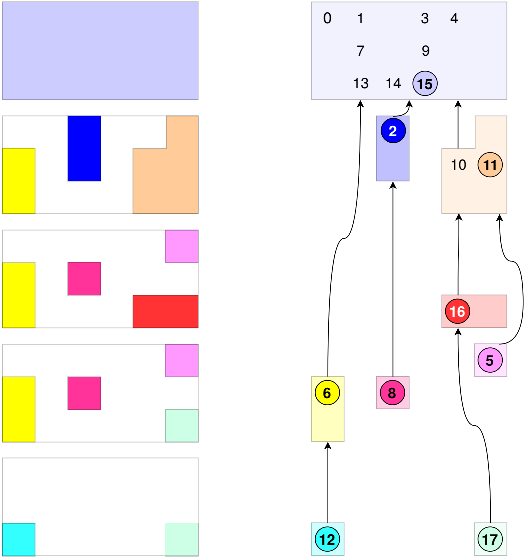
- Unordered union-find based version;

Algorithms

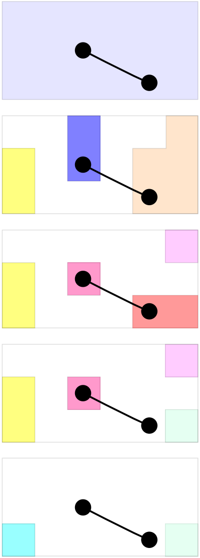
- Unordered union-find based version;
- Let $\mathcal{A}$ be a set of pair of neighboring pixels. Then, the algorithm follows this template:
  1. Initialize the max-tree with each pixel as a node; //makeset
  2. For each pair $(p, q) \in \mathcal{A}$:
     2.1 Connect p to q in the max-tree; //union (Wilkinson et al., 2008)

Algorithms

Union

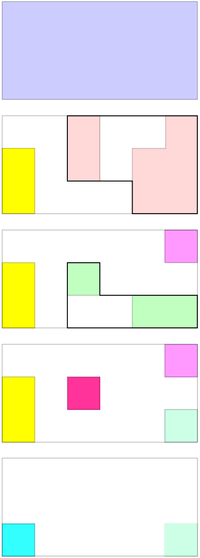


Algorithms
Union



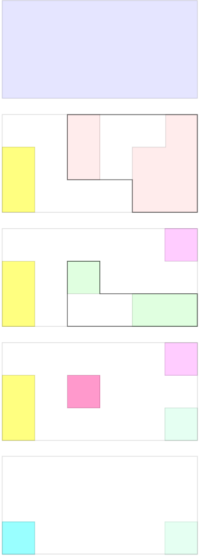
Algorithms

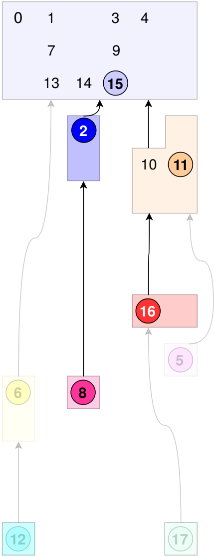
Union



Algorithms

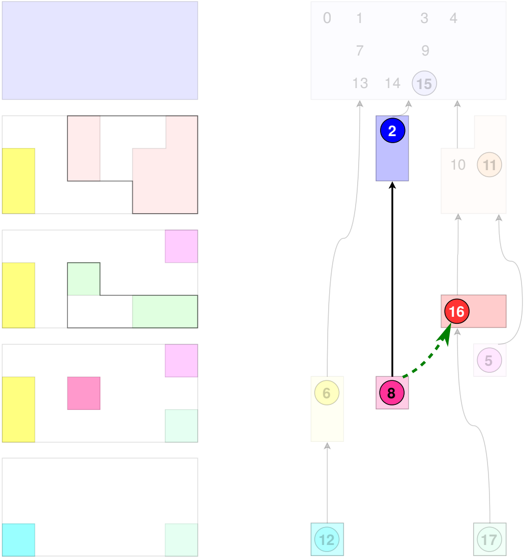
Union





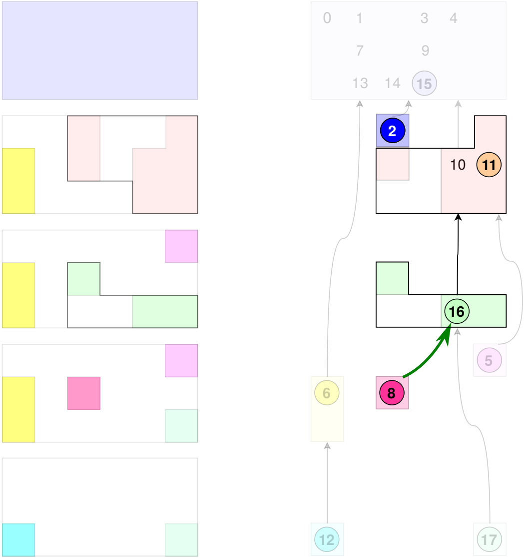
Algorithms

Union



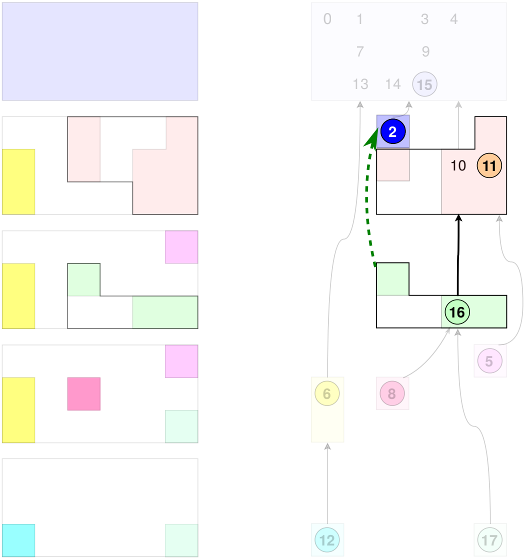
Algorithms

Union



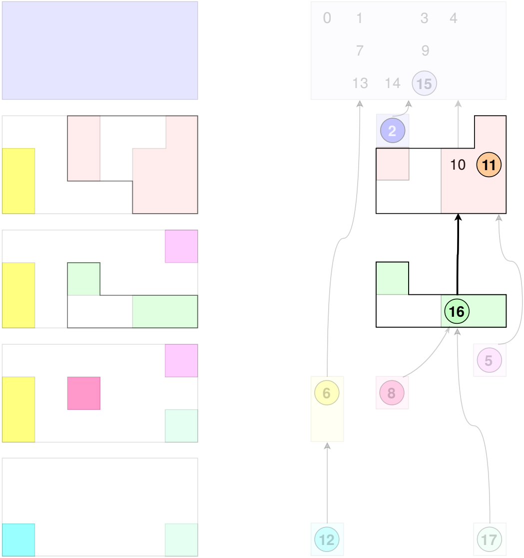
Algorithms

Union



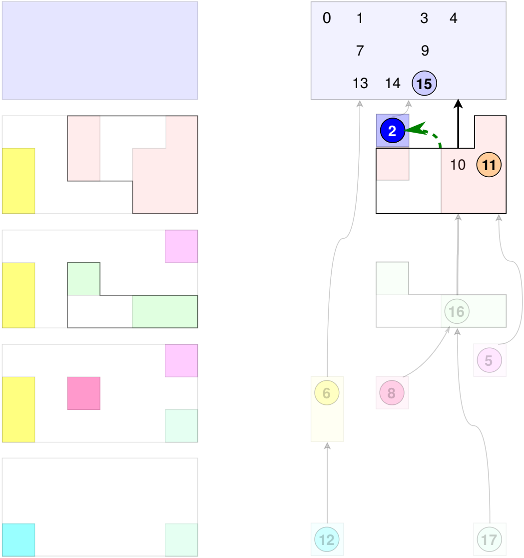
Algorithms

Union



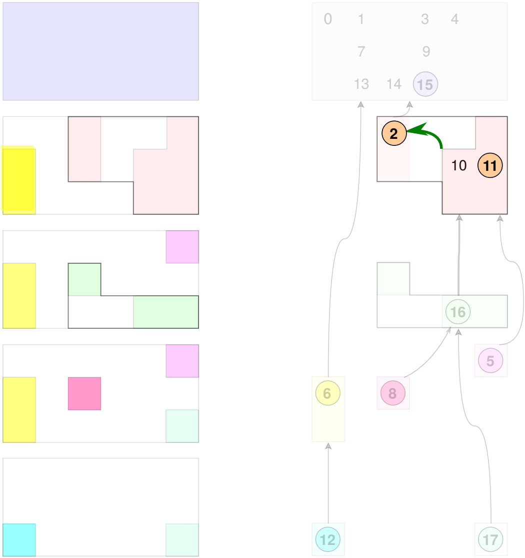
Algorithms

Union



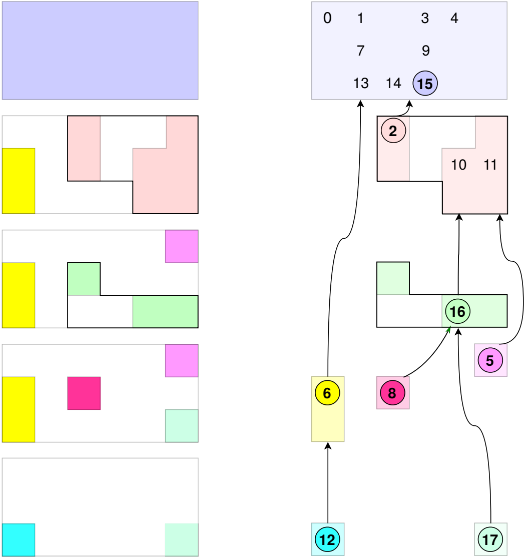
Algorithms

Union



Algorithms

Union



Algorithms
Union

- In the max-tree, connecting two pixels consists of merging two separate paths of the tree into one;

Algorithms

Hypertree Template

Let $\mathbb{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a sequence of n increasing sets of neighboring pixels. Then, the hypertree construction algorithm follows the template below:

1. Initialize the max-tree;
2. For $1 \leq i \leq n$:
   - 2.1 For (p, q) neighbors in $\mathcal{A}_i$:
     - 2.1.1 Connect p and q in the max-tree, looking for new nodes and arcs not present in step $i - 1$;
   - 2.2 Update the allocated hypertree based on new nodes and arcs;

Algorithms

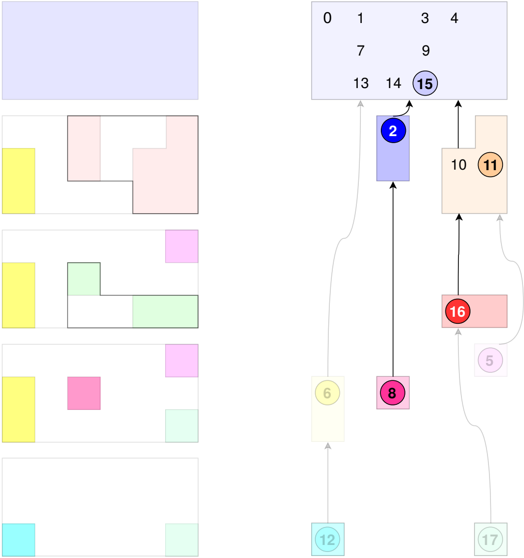
Hypertree Template

Let $\mathbb{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a sequence of n increasing sets of neighboring pixels. Then, the hypertree construction algorithm follows the template below:

1. Initialize the max-tree;
2. For $1 \leq i \leq n$:
   2.1 For **"relevant"** (p, q) in $\mathcal{A}_i$: //e.g. Morimitsu et al. (2015)
      2.1.1 Connect p and q in the max-tree, looking for new nodes and arcs not present in step $i - 1$;
   2.2 Update the allocated hypertree based on new nodes and arcs;

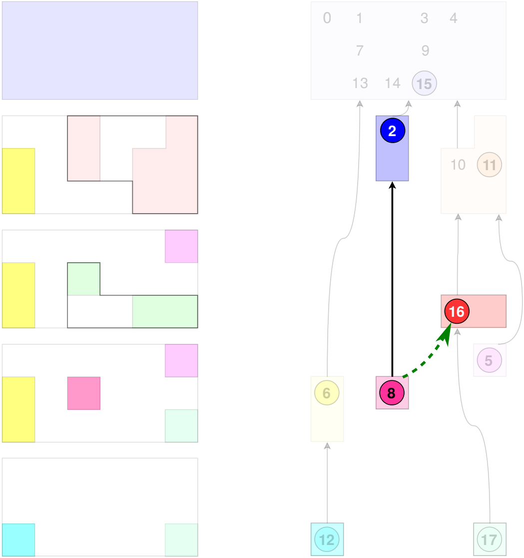
Algorithms

New nodes and arcs



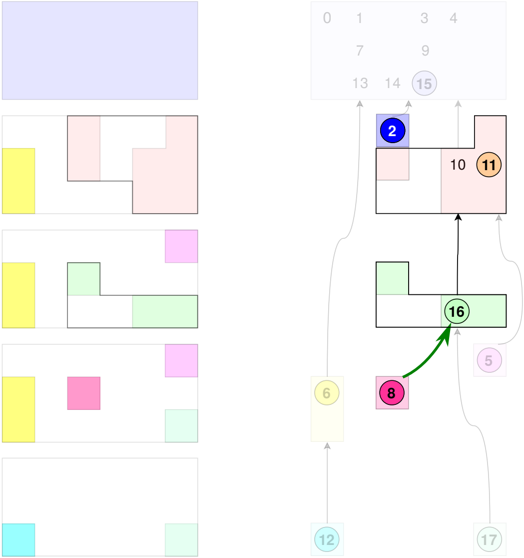
Algorithms

New nodes and arcs



Algorithms

New nodes and arcs



Algorithms

New nodes and arcs

- Detection of new nodes is found through changes in parent relation during the connect procedure;

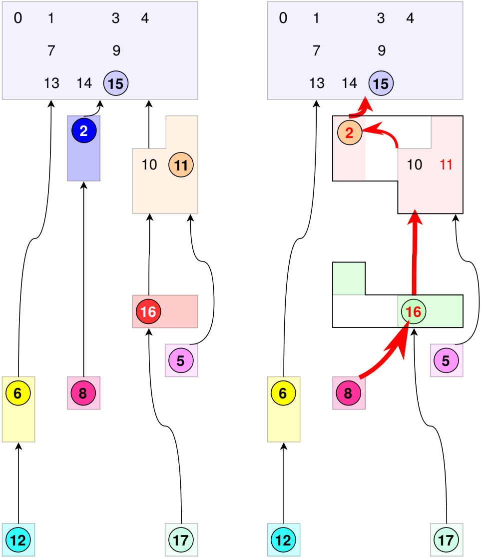
Algorithms

New nodes and arcs

- Detection of new nodes is found through changes in parent relation during the connect procedure;
- A node with a new child from the other path is a new node, since it contain at least a new pixel;
- All ancestors in this path, up to the common ancestor, will also now contain this new pixel and will be part of a new node.

Algorithms

New nodes and arcs



Algorithms

New nodes

- Mark all nodes (i.e., add their representative to a queue) in a path when a change in parenthood happens;
- Marked nodes are used to allocate new nodes;
  - Sometimes two marked nodes represent a same new node.
  - Usage of the *find* operation to avoid duplicated nodes;

Algorithms

New arcs

- Allocating arcs from new nodes:
  - For all new nodes, find their respective parent (from the max-tree) in hypertree and allocate these arcs;

Algorithms

New arcs

- Allocating arcs from new nodes:
  - For all new nodes, find their respective parent (from the max-tree) in hypertree and allocate these arcs;
- Allocating arcs pointing from old nodes to new nodes:
  - Allocate arcs that trigger a change in parenthood in the connect procedure;

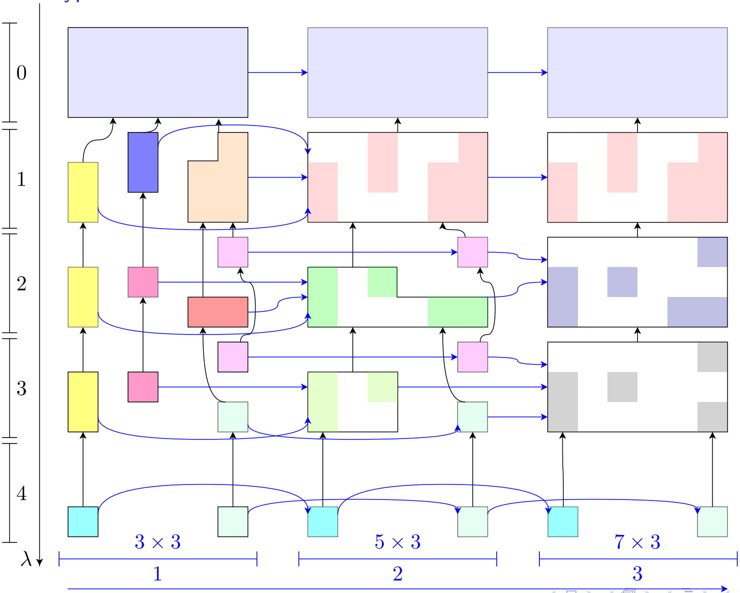
Algorithms

New arcs

- Allocating arcs from new nodes:
  - For all new nodes, find their respective parent (from the max-tree) in hypertree and allocate these arcs;
- Allocating arcs pointing from old nodes to new nodes:
  - Allocate arcs that trigger a change in parenthood in the connect procedure;
  - Find all nodes from the previous tree with the same representative as the new nodes and link them with an arc;

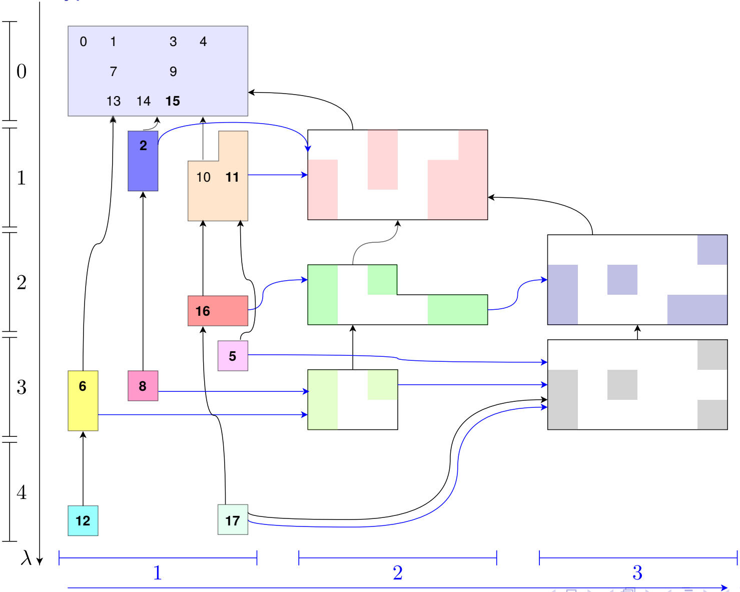
Theory

Obtained-hypertree



Theory

Obtained-hypertree



Theory

Max-trees vs. Obtained Component-Hypertrees

- Nodes are stored only once;
- Each pixel is stored only in the smallest node that contains it;
- Construction algorithm does not allocate repeated nodes.

- ✓;
- Each pixel is stored only in the smallest node that contains in the first tree;
- ✓;

Theory

Max-trees vs. Obtained Component-Hypertrees

- Nodes are stored only once;
- Each pixel is stored only in the smallest node that contains it;
- Construction algorithm does not allocate repeated nodes.
- All arcs give relevant information regarding inclusion relation.

- $\checkmark$;
- Each pixel is stored only in the smallest node that contains in the first tree;
- $\checkmark$;

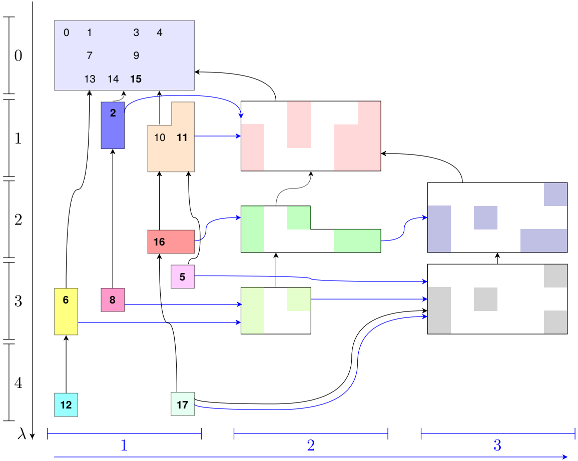
Theory

Max-trees vs. Obtained Component-Hypertrees

▶ Nodes are stored only once;	▶ ✓;
▶ Each pixel is stored only in the smallest node that contains it;	▶ Each pixel is stored only in the smallest node that contains in the first tree;
▶ Construction algorithm does not allocate repeated nodes.	▶ ✓;
▶ All arcs give relevant information regarding inclusion relation.	▶ Removes most arcs that give redundant information regarding inclusion relation.

Theory

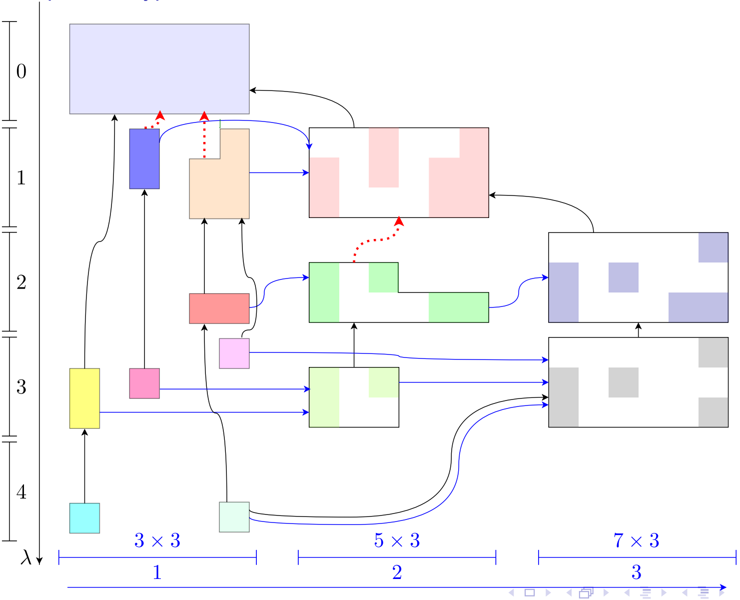
Minimal Component-Hypertrees



- Minimum hypertree?

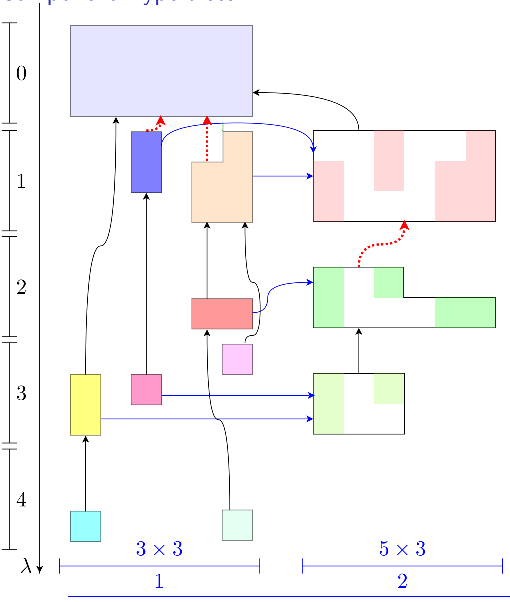
Theory

Minimal Component-Hypertrees



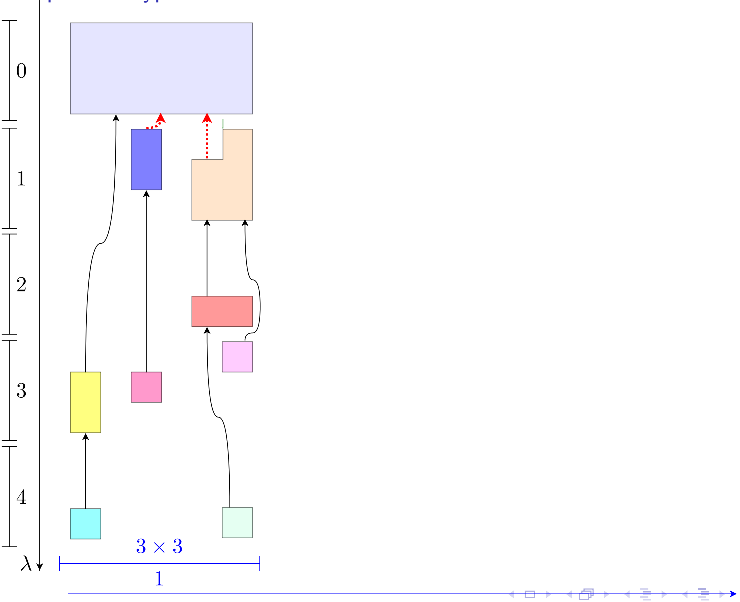
Theory

Minimal Component-Hypertrees



Theory

Minimal Component-Hypertrees



- The obtained hypertree has the smallest[1] number of nodes and arcs such that:

[1] except for double arcs

- The obtained hypertree has the smallest[1] number of nodes and arcs such that:
  1. All original inclusion relations are preserved;
  2. All nodes can be reconstructed without depending on nodes with higher connectivity index;

[1] except for double arcs

Experiments

Time consumption

- Updating the max-tree is the most time consuming step;

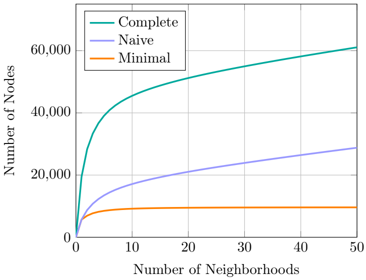
Experiments

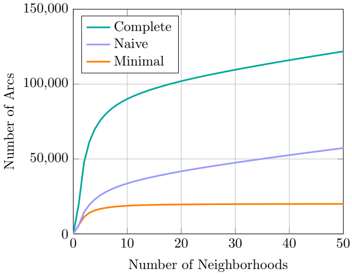
Time consumption

- Updating the max-tree is the most time consuming step;
- For an optimized implementation using 50 square neighborhoods:
  - Total time ranging from 1 (0.25 mega-pixels) to 60s (8 mega-pixels);
  - Only 3% to 6% of time used to allocate structures;

Experiments

Memory saving





Experiments

Memory saving

- Memory saved, in average, for $n = 10$:
  - about 80% compared to the complete representation;
  - about 50% compared to the naive implementation;

Experiments

Memory saving

- Memory saved, in average, for $n = 10$:
  - about 80% compared to the complete representation;
  - about 50% compared to the naive implementation;
- This percentage increases as n increases since the number of new nodes decreases;

Conclusion

Conclusion

- Algorithms for efficient storage of component-hypertrees was proposed;
- Big saves in storage compared to other approaches;
- Allocation of nodes and arcs is fast;
- Computation of attributes will be presented in a later date (ISMM 2019);

Last Remarks

- Thank you!
- Questions? You can also check our poster;

Acknowledgements

This study was financed in part by the CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Finance Code 001); FAPESP - Fundação de Amparo a Pesquisa do Estado de São Paulo (Proc. 2018/15652-7); CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico (Proc. 428720/2018-8).

References

Alexandre Morimitsu, Wonder AL Alves, and Ronaldo F Hashimoto. Incremental and efficient computation of families of component trees. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 681–692. Springer, 2015.

Nicolas Passat and Benoît Naegel. Component-hypertrees for image segmentation. In *ISMM*, volume 6671, pages 284–295. Springer, 2011.

Philippe Salembier, Albert Oliveras, and Luis Garrido. Antiextensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7(4): 555–570, 1998.

Michael HF Wilkinson, Hui Gao, Wim H Hesselink, Jan-Eppo Jonker, and Arnold Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 (10):1800–1813, 2008.