Wavelets for progressive transmission/compression of images
The SPIHT method

WTBV WS 2016/17

January 25, 2017

- Methods for data compression, including wavelet methods for image compression, are treated in an encyclopedic way in

  D. Salomon, G. Motta, Handbook of Data Compression, Springer-Verlag London, 2010.

  1360 pages, more that 120 pages devoted to wavelet methods (Chap. 8).
  The SPIHT algorithm is in Section 8.14 (12 pages).
- A shorter presentation of both wavelet methods in general and SPIHT in particular is contained in Chapter 4 and Section 4.8 of

  D. Salomon, A Guide to Data Compression Methods, Springer-Verlag New York, 2002.

- Another recent source is

  K. Sayood, Introduction to Data Compression, 4-th ed., Elsevier, 2012.

  Chap. 12: Fourier techniques
  Chap. 13: Transform Coding
  Chap. 14: Sub-band coding (including filter banks)
  Chap. 15: Concise treatment of wavelets
  Chap. 16: Wavelet-based Image Compression
  SPIHT in Sec. 16.4.
  JPEG2000 detailed discussion in Sec. 16.5

- A wavelet book with special attention to data compression is

  R.M. Rao, A.S. Bopardikar, Wavelet Transforms, Introduction to Theory and Applications, Springer, Addison-Wesley, 1998.

  Chapter 5 is on Wavelet Transform and Data Compression, Sec. 5.3.2 presents the SPIHT algorithm, including an example that is reproduced in this lecture

- It is always worthwile to look into

  S. Mallat, A Wavelet Tour of Signal Processing, 3rd. ed., Elsevier, 2009.

  Mathematically more demanding than all the others. Chap. 10 is devoted to Compression. Sec. 10.4.2 gives a compact presentation of embedded coding and SPIHT

- The SPIHT algorithm is due to

  A. Said, W.A. Pearlman, A new, fast, and efficient image codec based on set partitioning in hierarchical trees, IEEE Trans. Circ. Syst. Video Technol. 6(3): 243–250, 1996.

  The authors have establishes a web page **cipr.rpi.edu** with lots of material (descriptions, demos, code, applications, . . .) and you can also download the mentioned article (very readable!). There have been many approaches to the use of wavelets in data compression. SPIHT undeniably belongs to the most efficient ones

- Introductory remarks:
  - SPIHT (= *set partitioning in hierarchical trees*) is prototypical for the application of wavelet transform methods in the field of image compression
  - SPIHT combines two ideas:
    - *progressive transmission*: frequently found in signal transmission and data compression methods: digital data (like pixel values of an image) are organized in *bit-planes*, thus giving preference in the transmission process to high significant bits over less significant bits
    - *embedded trees*, makes use of dependencies that exist in wavelet transformed data over several layers of resolution (sub-bands) so that pixels that relate to the same position in the original image are organized in quaternary trees

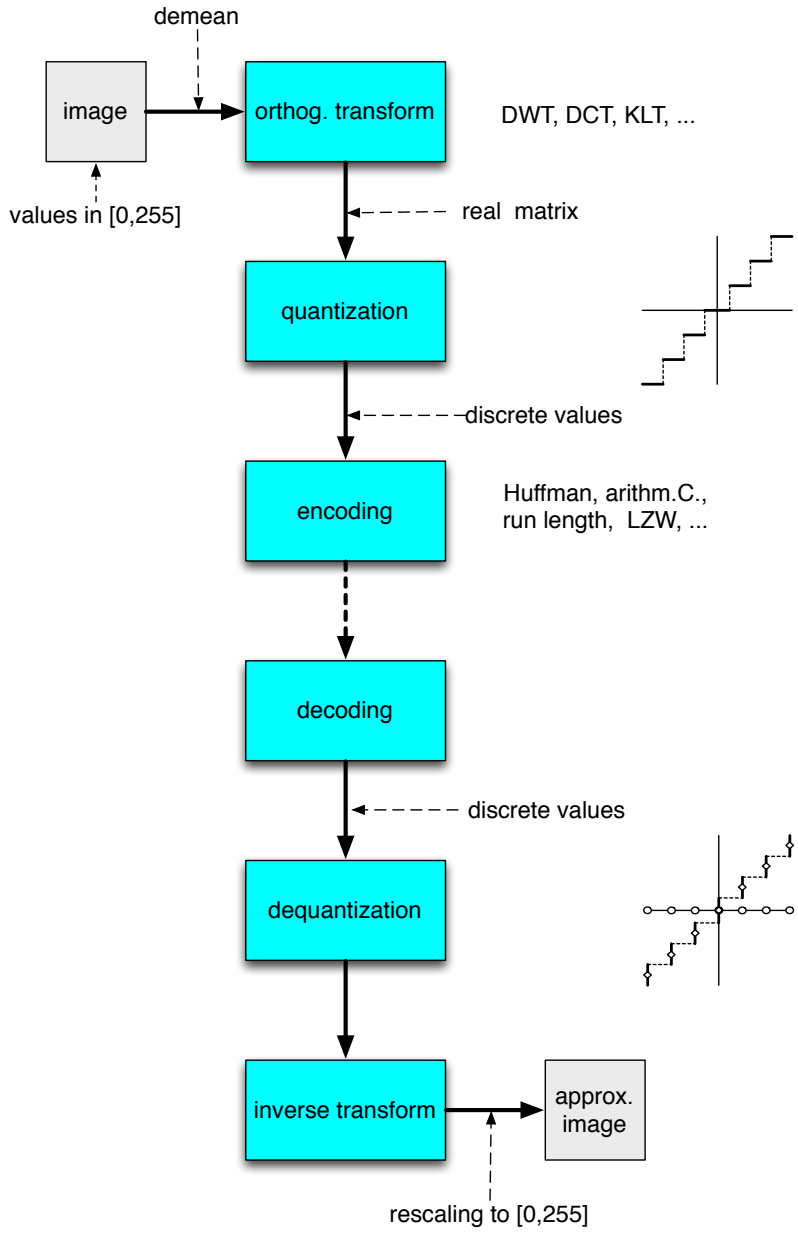


Figure: Structure of a wavelet transform image codec

- Heuristics for progressive transmission
  - $\boldsymbol{p} = \{p_{i,j}\}_{(i,j)\in I}$: an image, which is transform-coded by applying an orthogonal transform $\boldsymbol{T}$ to it, which yields the data $\boldsymbol{c} = \{c_{i,j}\}_{(i,j)\in I}$
  - $\widehat{\boldsymbol{c}} = \{\widehat{c}_{i,j}\}_{(i,j)\in I}$: an approximation of $\boldsymbol{c}$ (e.g., the k most significant bits of each coefficient) obtained by transforming and rearranging using a procedure $\boldsymbol{S}$
  - $\widehat{\boldsymbol{p}} = \{\widehat{p}_{i,j}\}_{(i,j)\in I}$: an approximation of the original input $\boldsymbol{p}$ obtained by coefficient-wise back-transforming using $\boldsymbol{T}^{-1}$
  - Schematically

$$\boldsymbol{p} \xrightarrow{\boldsymbol{T}} \boldsymbol{c} \xrightarrow{\boldsymbol{S}} \widehat{\boldsymbol{c}} \xrightarrow{\boldsymbol{T}^{-1}} \widehat{\boldsymbol{p}}$$

  - By orthogonality of T,

$$\| \boldsymbol{p} - \widehat{\boldsymbol{p}} \|^2 = \sum_{(i,j)\in I} |p_{i,j} - \widehat{p}_{i,j}|^2 = \sum_{(i,j)\in I} |c_{i,j} - \widehat{c}_{i,j}|^2 = \| \boldsymbol{c} - \widehat{\boldsymbol{c}} \|^2$$

  - The mean-squared error of approximation:

$$D_{MSE} = \frac{1}{N} \| \boldsymbol{p} - \widehat{\boldsymbol{p}} \|^2 = \frac{1}{N} \| \boldsymbol{c} - \widehat{\boldsymbol{c}} \|^2$$

    where $N = \sharp I$ is the total number of coefficients

- Heuristics for progressive transmission (contd.)
  - Now assume that for a subset $J \subseteq I$ one has

$$\widehat{c}_{i,j} = \begin{cases} c_{i,j} & \text{if } (i,j) \in J \\ 0 & \text{if } (i,j) \notin J \end{cases}$$

    Then

$$D_{MSE} = \frac{1}{N} \sum_{i,j \notin J} |c_{i,j}|^2$$

  - Considering all subsets J of I of the same cardinality N_0 the one(s) which yield(s) the minimum D_{MSE} is obtained by
    - sorting the coefficients $c_{i,j}$ ($(i,j) \in I$) according to their (decreasing) absolute value
    - taking for J the initial N_0 positions of the sorted sequence
- In practice:
  1. order the elements $c_{i,j}$ according to the bitlength $\lfloor \log_2 |c_{i,j}| \rfloor + 1$ of their absolute values
  2. transmit positions and signs of the elements of maximal bitlength r
  3. for $k = r-1, r-2, \ldots$
     - 3.1 transmit positions and signs of the elements of bitlength k
     - 3.2 transmit the k-th bit of all elements of bitlength $> k$ (i.e., transmitted in earlier rounds)

Progressive transmission, the procedures

procedure *scan*(L,C,N,LIP,LSP)

(* (L,C): a list $(C(\xi))_{\xi \in L}$ of integers
N: significance level
LIP, LSP: lists that are updated during the execution*)

```
for ξ in LIP do
   if |C(ξ)| ≥ N then
      output(ξ,sgn(C(ξ)))
      remove(LIP,ξ)
      append(LSP,ξ)
   end if
end for
return (LIP,LSP)
```

procedure *progtrans*(L,C) (* (L,C): a list $(C(\xi))_{\xi \in L}$ of integers*)

(*initialization*)
$r \leftarrow \lfloor \log_2 \max_{\xi \in L} |C(\xi)| \rfloor$
output(r)
$N \leftarrow 2^r$
LIP $\leftarrow$ L
LSP $\leftarrow$ empty list
(*sorting*)
(LIP,LSP) $\leftarrow$ *scan*(L,C,N,LIP,LSP)
oldLSP $\leftarrow$ LSP
repeat
$r \leftarrow r - 1$, $N \leftarrow N/2$
(*sorting*)
(LIP,LSP) $\leftarrow$ *scan*(L,C,N,LIP,LSP)
(*refinement*)
for $\xi \in$ oldLSP **do**
output(r-th bit of $|C(\xi)|$)
end for
oldLSP $\leftarrow$ LSP
until "as needed"

Example for progressive transmission:

Output of *progtrans*(3, -12, 4, 9, -5, 0, -1, -8, 8, 0)

round	position	sign	bit values
1	$\{2, 4, 8, 9\}$	$\{1, 0, 1, 0\}$	$\{\}$
2	$\{3, 5\}$	$\{0, 1\}$	$\{1, 0, 0, 0\}$
3	$\{1\}$	$\{0\}$	$\{0, 0, 0, 0, 0, 0\}$
4	$\{7\}$	$\{1\}$	$\{0, 1, 0, 0, 0, 1, 1\}$

Array representation of progressive transmission

pos	sgn	bit_1	bit_2	bit_3	bit_4
2	1	1	1	0	0
4	0	1	0	0	1
8	1	1	0	0	0
9	0	1	0	0	0
3	0	0	1	0	0
5	1	0	1	0	1
1	0	0	0	1	1
7	1	0	0	0	1
6	0	0	0	0	0
10	0	0	0	0	0

Successive approximations

	1	2	3	4	5	6	7	8	9	10	error
1 bit	0	−11	0	11	0	0	0	−11	11	0	8.60233
2 bit	0	−13	5	9	−5	0	0	−9	9	0	3.74166
3 bit	2	−12	4	8	−4	0	0	−8	8	0	2.00000
4 bit	3	−12	4	9	−5	0	−1	−8	8	0	0

Zig-zag ordering

- Progressive transmission is a *sequential* procedure, so when applying it to image data, one has to *serialize* the 2-dimensional data somehow
- There are many ways to do that...
- *zig-zag-ordering* of the elements of $\mathbb{N} \times \mathbb{N}$ is a particular convenient method, that is adapted to work well with DWT
- Notation for initial 2D-segments of $\mathbb{N} \times \mathbb{N}$

$$\square_n = \{0, \ldots, 2^n - 1\} \times \{0, \ldots, 2^n - 1\}$$

and for $0 \leq i, j \leq 1$

$$\square_n^{i,j} = \{2^{i \cdot n} + 0, \ldots, 2^{i \cdot n} + 2^n - 1\} \times \{2^{j \cdot n} + 0, \ldots, 2^{j \cdot n} + 2^n - 1\},$$

so that schematically

$$\square_{n+1} = \begin{bmatrix} \square_n^{0,0} & \square_n^{0,1} \\ \\ \square_n^{1,0} & \square_n^{1,1} \end{bmatrix}$$

- $\square_n$ is identified with the upper-left square $\square_{n+1}^{0,0}$ of $\square_{n+1}$. Thus one has an order-compatible inclusion chain

$$\square_1 \subset \square_2 \subset \square_3 \ldots \nearrow \mathbb{N} \times \mathbb{N}$$

- It suffices to define the zig-zag ordering on each $\square_n$
  1. Order the elements of $\square_1 = \{0,1\} \times \{0,1\}$ in Z-form

	0		1
0	0	$\rightarrow$	1
		$\swarrow$	
1	2	$\rightarrow$	3

  2. If the ordering has been defined for $\square_n$ for some $n \geq 1$. Then the ordering on $\square_{n+1}$ is given by
     - 2.1 The ordering on each of $\square_{n+1}^{i,j}$ $(0 \leq i,j \leq 1)$ is "the same" as the ordering of $\square_n$
     - 2.2 The ordering among the elements of different $\square_{n+1}^{i,j}$ is again given by zig-zag, as symbolized by

$$\square_{n+1}^{0,0} < \square_{n+1}^{0,1} < \square_{n+1}^{1,0} < \square_{n+1}^{1,1}$$

- So the zig-zag ordering of $\square_2$ is

$$\begin{array}{cccc} 0 & 1 & 4 & 5 \\ 2 & 3 & 6 & 7 \\ 8 & 9 & 12 & 13 \\ 10 & 11 & 14 & 15 \end{array}$$

- The functions
  - $m2l : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $m2l(i, j)$ is the ordinal number of (i, j) in the zig-zag ordering, and its inverse
  - $l2m = \mathbb{N} \to \mathbb{N} \times \mathbb{N}$, where $l2m(k)$ gives the position of the element with ordinal number k in $\mathbb{N} \times \mathbb{N}$,

  can easily described in terms of the binary expansions of the numbers involved

- Embedded trees in DWT-data
  - The 1D-case

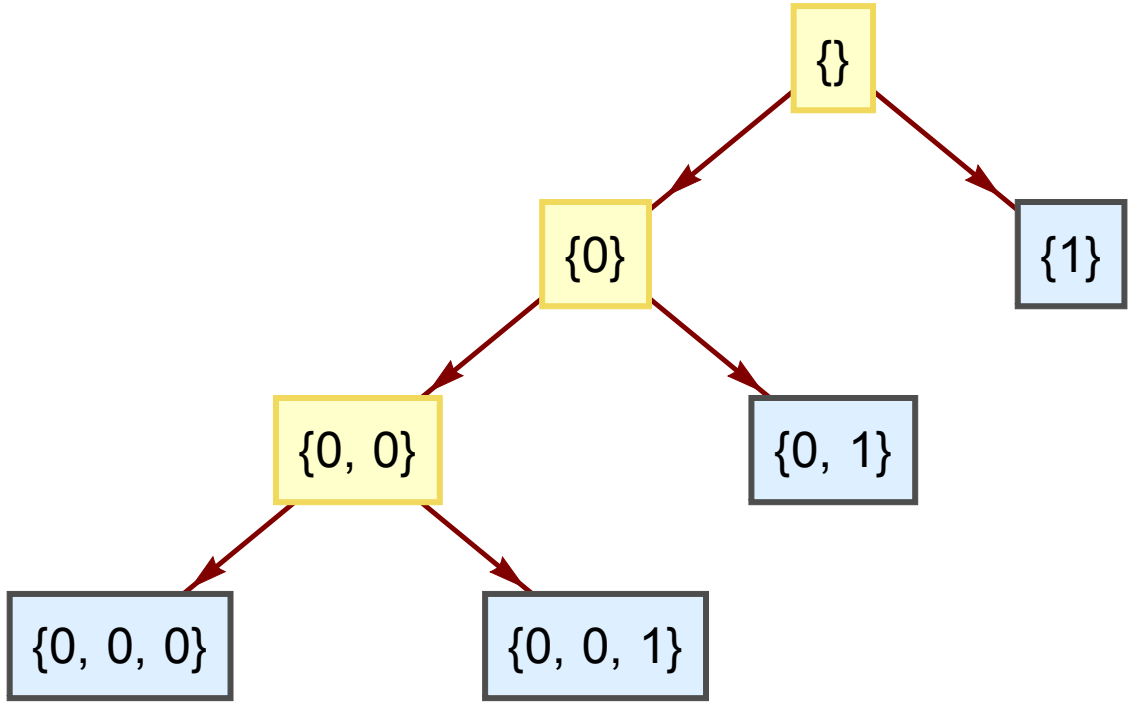


Figure: The decomposition scheme of a 3-level 1D-DWT (analysis)

  - The various parts of the wavelet decomposition tree are henceforth named *sub-bands* and denoted as indicated as sub-bands 0^k or 0^k1.

- The embedded trees in the case of a data vector of length 32 for a 3-level transform.

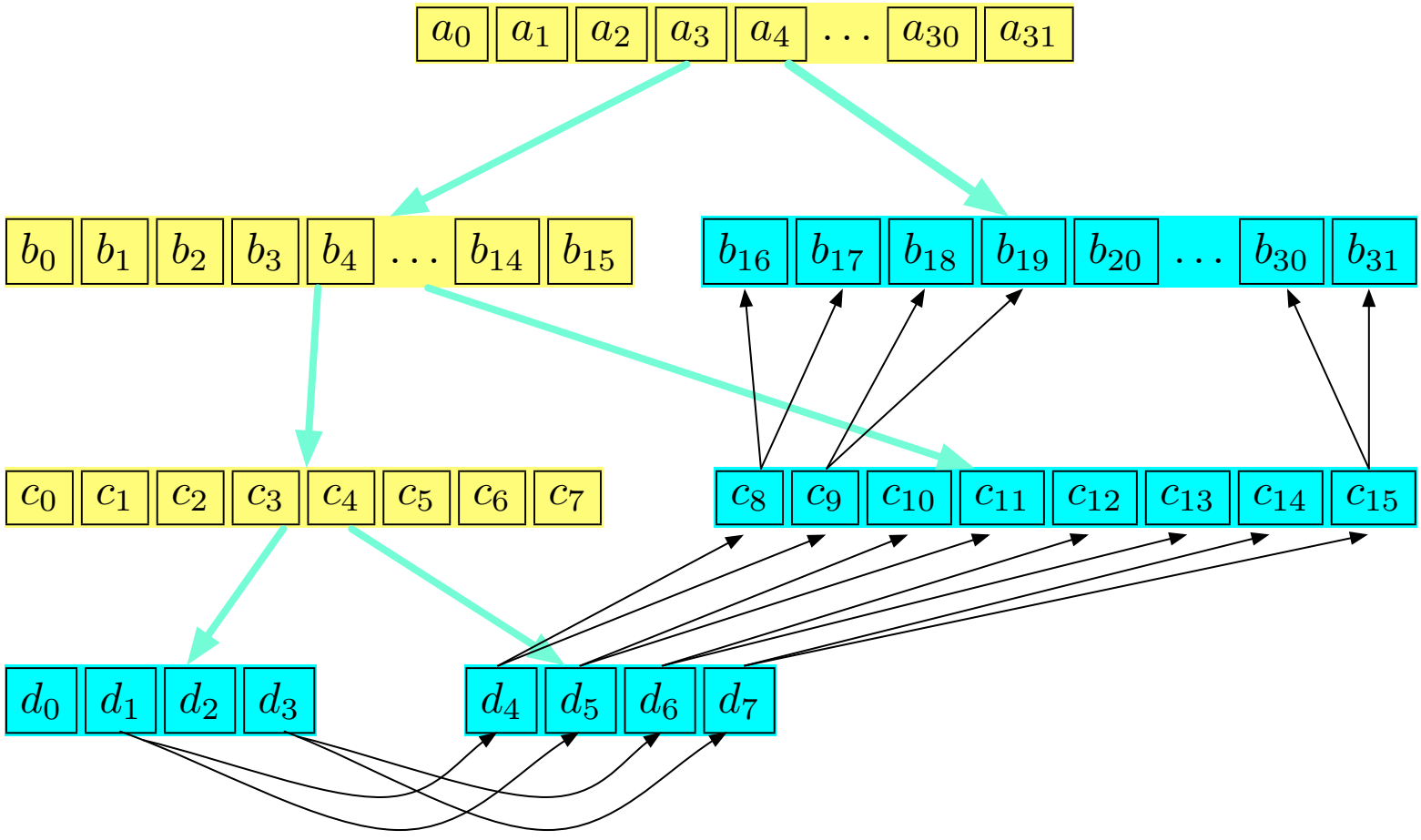


Figure: Embedded binary trees in context

- Motivation
  - Positions joined by black arrows refer to same position in the original data, hence the wavelet coefficients should be related. E.g., if d_5 is small, then most probably its descendants c_{10}, c_{11} and also $b_{20}, \ldots, b_{23}$ will be small too.
    In the case of smooth signal data there should be strongly decreasing tendency as the resolution increases.
    In particular: if a wavelet coefficient is 0, the most probably the coefficients of all its descendants will be 0 too.
    This phenomenon can be used profitably in tasks like data compression and progressive transmission.

- The embedded trees graph as drawn by Mathematica.

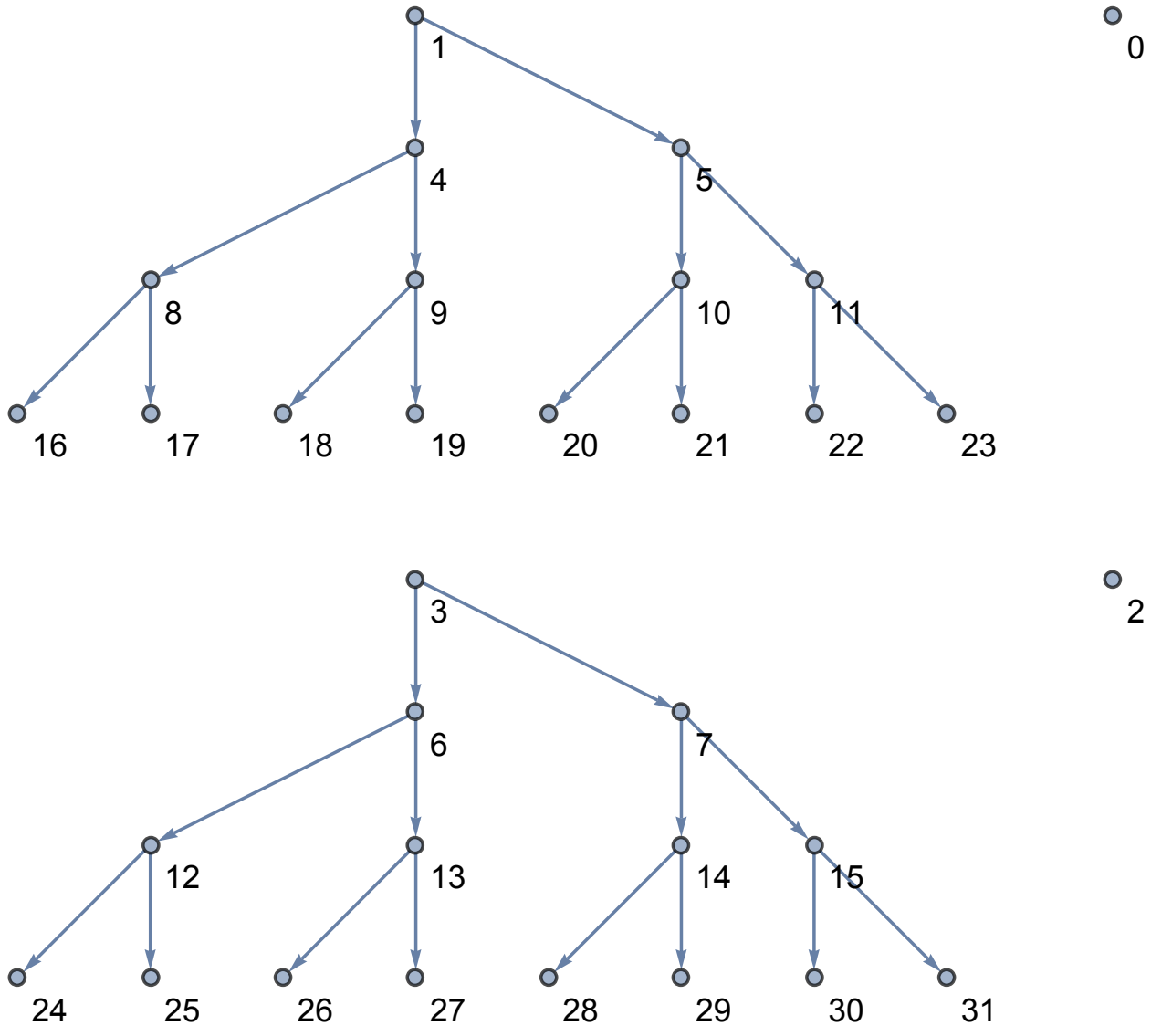


Figure: Embedded binary trees

- Such a forest can be described linearly by giving a list of predecessor labels (i.e., reversing the arrows, with the convention that each root label is its own predecessor):

$$0, 0, 0, 0, 1, 1, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9,$$
$$10, 10, 11, 11, 12, 12, 13, 13, 14, 14, 15, 15$$

- This list is just the bottom row in

$$\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & \dots & 31 & 32 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \dots & \downarrow & \downarrow \\ 0 & 0 & 0 & 0 & 1 & 1 & 3 & \dots & 15 & 15 \end{array}$$

- General rule for constructing these embedded trees for a data vector of length N that represents data from an ℓ-level DWT:
  - Index the data from 0 to $N-1$
  - Set $a = N/2^{\ell}$: this is the length of the 0^{ℓ} and of the $0^{\ell-1}1$ components (sub-bands) – this must be an even integer
  - All even indexed positions in the 0^{ℓ} sub-band will be isolated vertices
  - All odd indexed positions $2k+1$ in the 0^{ℓ} sub-band have successors $a+2k$ and $a+2k+1$ in the $0^{\ell-1}1$ sub-band $(0 \leq k < \lfloor a/2 \rfloor)$
  - Each position k for the transformed data which does not belong to the 0^{ℓ} sub-band or to the 1-sub-band has $2k$ and $2k+1$ as its successors.

  This procedure gives $a/2$ complete binary trees of depth ℓ.

- Embedded trees for the 2D-DWT (*spatial orientation trees*)

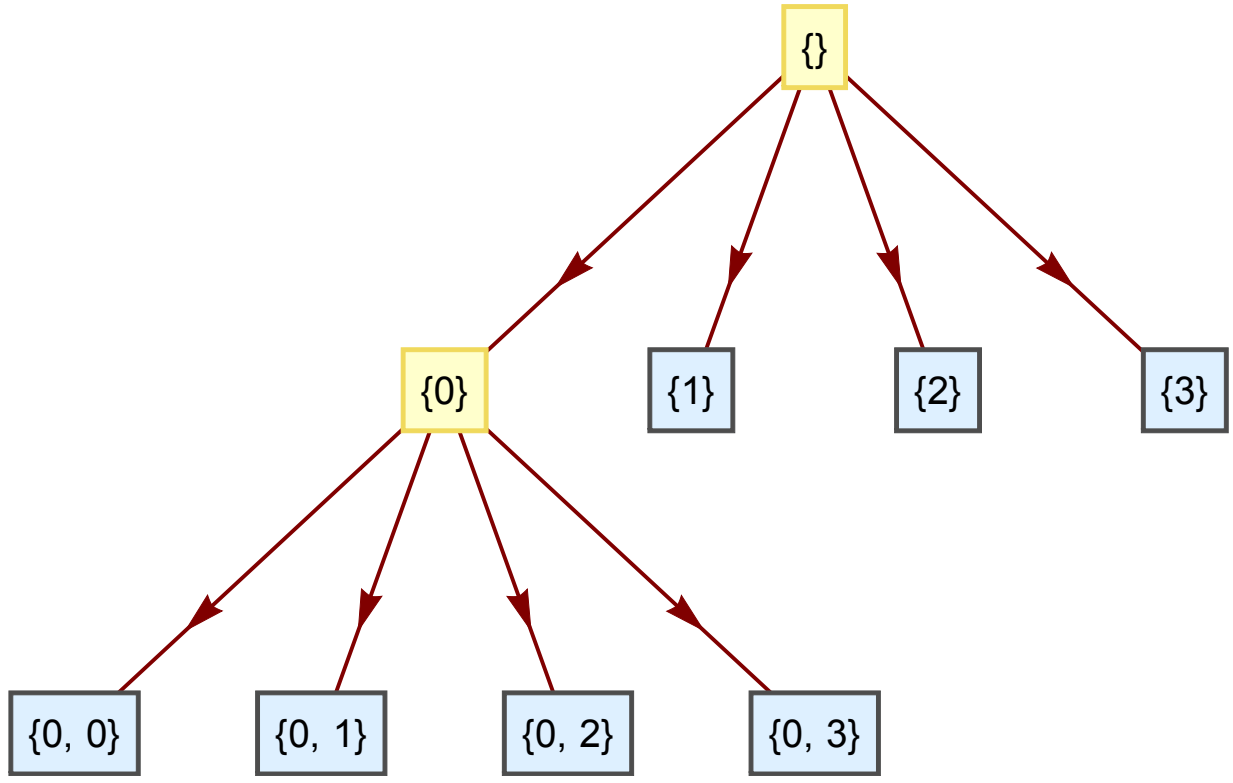


Figure: Decomposition scheme for a 2-level 2D-DWT (analysis)

- Construction of the embedded quaternary trees
  - The image is assumed to be square of size $2^n \times 2^n$.
    Numbering of rows and columns starts at 0.
  - ℓ : the number of levels of a 2D-DWT
    Then the four sub-bands on the lowest level 0^ℓ and
    $0^{\ell-1}1, 0^{\ell-1}2, 0^{\ell-1}3$ have size $a \times a$ with $a = 2^{n-\ell}$.
    a is assumed to be even, i.e., $\ell < n$.
  - Partition the 0^ℓ-sub-band into 2×2 squares.
    The upper left position (i, j) with $0 \leq i, j < a$ and i, j even of each square is an isolated vertex.

- Construction contd.

  (A) All other positions in the 0^ℓ sub-band become roots of quaternary trees of height ℓ. These roots have their four successors in the sub-bands $0^{\ell-1}1$, $0^{\ell-1}2$, $0^{\ell-1}3$.
  Precisely:

$$(i,j) \longrightarrow \begin{cases} (i, a+j-1)+\varepsilon & \text{if } i \text{ even and } j \text{ odd} \\ (i+a-1, j)+\varepsilon & \text{if } i \text{ odd and } j \text{ even} \\ (i+a-1, j+a-1)+\varepsilon & \text{if } i \text{ and } j \text{ odd} \end{cases}$$

  where $\varepsilon \in \{(0,0),(0,1),(1,0),(1,1)\}$.

  (B) From then on, each vertex in a sub-band (other than the sub-bands 1, 2, 3, where positions are "leaf" positions, i.e., positions without successors) has four successors in one of the sub-bands of the next higher frequency "in the same relative position", as indicated in the graphical sketch.
  Precisely

$$(i,j) \longrightarrow (2i, 2j)+\varepsilon$$

An example with $n = 4, \ell = 2, a = 4$

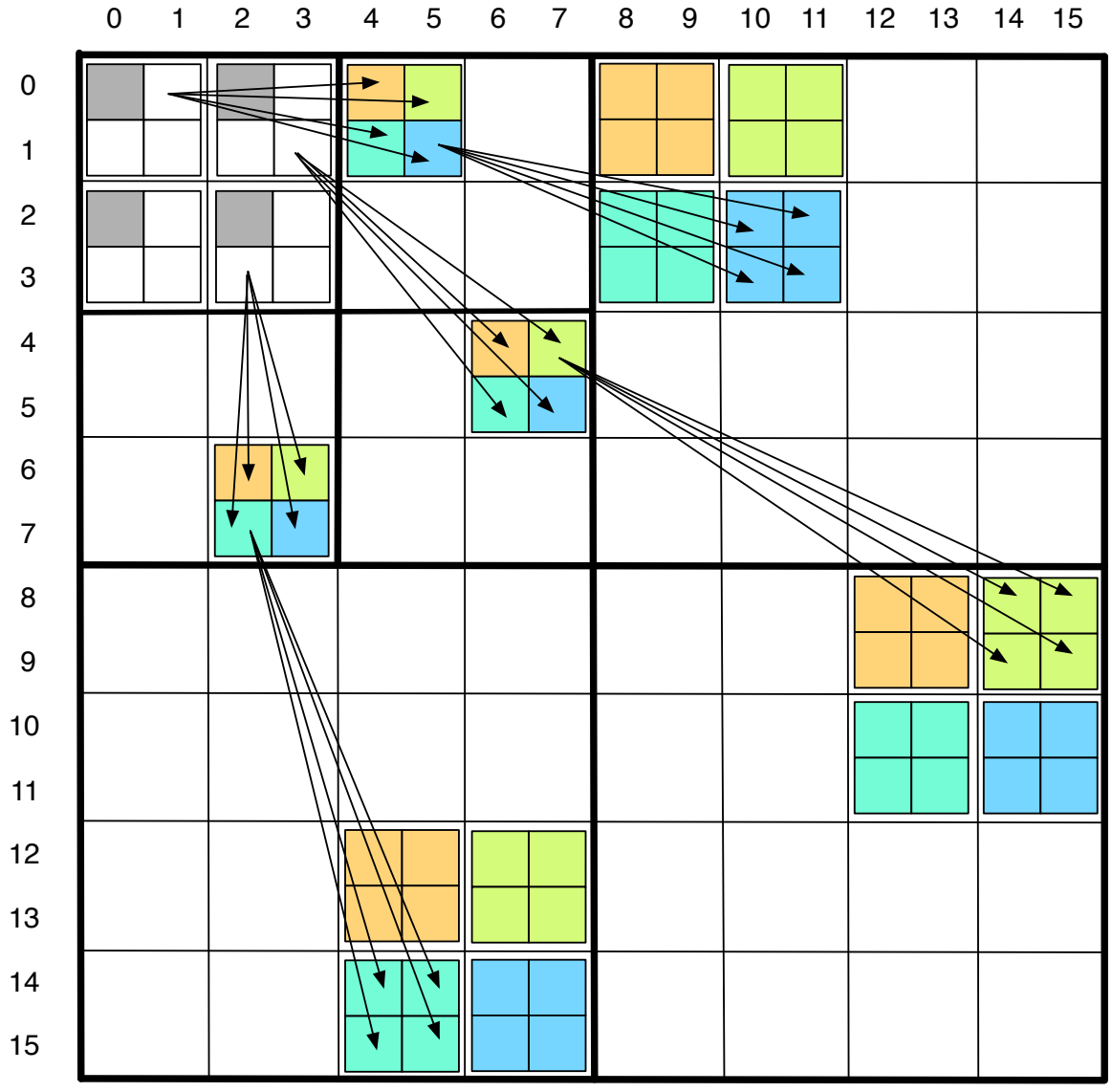


Figure: Embedded spatial orientation trees

The exemplary arrows drawn in the example figure are

$$
\begin{array}{lcll}
(0,1) & \longrightarrow & (0,4)+\varepsilon & = \{(0,4),(0,5),(1,4),(1,5)\} \quad \text{by (A)} \\
(1,3) & \longrightarrow & (4,6)+\varepsilon & = \{(4,6),(4,7),(5,5),(5,7)\} \quad \text{by (A)} \\
(1,5) & \longrightarrow & (2,10)+\varepsilon & = \{(2,10),(2,11),(3,10),(3,11)\} \quad \text{by (B)} \\
(3,2) & \longrightarrow & (6,2)+\varepsilon & = \{(6,2),(6,3),(7,2),(7,3)\} \quad \text{by (A)} \\
(4,7) & \longrightarrow & (8,14)+\varepsilon & = \{(8,14),(8,15),(9,14),(9,15)\} \quad \text{by (B)} \\
(7,2) & \longrightarrow & (14,4)+\varepsilon & = \{(14,4),(14,5),(15,4),(15,5)\} \quad \text{by (B)}
\end{array}
$$

Besides the 4 isolated vertices $(0,0),(0,2),(2,0),(2,2)$ (shaded in the figure) there are 12 quaternary trees of height 2, rooted in the remaining 12 vertices of the 0^2-sub-band.

A convenient way to linearly encode the structure of these embedded trees is by referring to the zigzag-ordering.

As an example: in an (8×8)-array the positions are numbered by the zigzag-ordering as shown here:

$$\begin{pmatrix} 0 & 1 & 4 & 5 & 16 & 17 & 20 & 21 \\ 2 & 3 & 6 & 7 & 18 & 19 & 22 & 23 \\ 8 & 9 & 12 & 13 & 24 & 25 & 28 & 29 \\ 10 & 11 & 14 & 15 & 26 & 27 & 30 & 31 \\ 32 & 33 & 36 & 37 & 48 & 49 & 52 & 53 \\ 34 & 35 & 38 & 39 & 50 & 51 & 54 & 55 \\ 40 & 41 & 44 & 45 & 56 & 57 & 60 & 61 \\ 42 & 43 & 46 & 47 & 58 & 59 & 62 & 63 \end{pmatrix}$$

The embedded trees relevant for a 2-level DWT can be displayed in a matrix that for each position gives the position of its tree-predecessor vertex

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 4 & 4 & 5 & 5 \\ 2 & 3 & 1 & 1 & 4 & 4 & 5 & 5 \\ 2 & 2 & 3 & 3 & 6 & 6 & 7 & 7 \\ 2 & 2 & 3 & 3 & 6 & 6 & 7 & 7 \\ 8 & 8 & 9 & 9 & 12 & 12 & 13 & 13 \\ 8 & 8 & 9 & 9 & 12 & 12 & 13 & 13 \\ 10 & 10 & 11 & 11 & 14 & 14 & 15 & 15 \\ 10 & 10 & 11 & 11 & 14 & 14 & 15 & 15 \end{pmatrix}$$

and the zigzag-ordering would give a 1-dimensional representation of this information

$$0, 1, 2, 3, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, \ldots, 15, 15, 15, 15$$

- A nice and useful feature of the zigzag-ordering is apparent:

  positions that are offsprings of the same position w.r.t. the embedded quaternary trees a neighbors in the matrix and the linear version of the predecessor information.

- The SPIHT method: General description (1)
  - The data
    - The procedures *scantree* and *spiht* operate on forests T of rooted trees.
    - T is specified by a list of root vertices and a function which to every vertex lists its immediate successors.
    - The vertices of T carry an integer valuation given by a function C.
    - C-values are thought of as binary strings of some common length r together with an extra sign bit.
  - The purpose of *spith*
    - is to compute a signed k-bit approximation of C for some value $k \leq r$ that may or may not be fixed in advance.
    - *scantree* is used to compute a $(k+1)$-bit approximation from a k-bit approximation (for $k = 1, 2, \ldots$) using information accumulated during the computation of the latter.
    - The idea of *progressive transmission* is used, together with a systematic exploration of the tree structures of T in view of the mapping C.

- The SPIHT method: General description (2)
  - Encoding and decoding
    - Running *spith* produces a bit stream as *output*.
    - This stream contains *significance* and *sign* information about (T, C), which encode the branchings of *spith* and its procedure *scantree* when exploring T
    - This stream contains all the information for exactly simulating the action of *spith* by just knowing the tree (or forest) structure T and reading the stream, but not knowing C
    - If one considers *spith* as an *encoder*, then replacing all *output*-commands in *spith* and *scantree* by *input*-commands (thus reading elements of the bit stream precisely where in *spith* they where produced) gives the view of a *decoder* who wants to compute C to a certain degree of accuracy with as little information as possible. This is where the aspect of *data compression* comes into play

- The SPIHT method: General description (3)
  - The *spith* algorithm is given below in an abstract form, i.e., the structure of T occurs only implicitly by referring to a successor map which is assumed to be known, but not further specified
  - SPITH for images and wavelets
    - The main application of *spith* is with images, notably their wavelet transforms with the structure of *embedded trees* made explicit
    - In this situation T is a forest consisting of a certain number of isolated points and a set of quaternary trees (all of the same height)
    - The typical properties of coefficient value arrays produced with DWT are exploited profitably when using these embedded trees in the *spiht* algorithm

- Data for and verbal description of the SPIHT-algorithm (1)
  - For each vertex ξ of T one defines
    - $\mathcal{O}(\xi)$: the (ordered) set of *offsprings* (immediate successors) of ξ
    - $\mathcal{D}(\xi)$: the set of *descendants* of ξ (i.e., offsprings and their descendants)
    - $\mathcal{L}(\xi) = \mathcal{D}(\xi) \setminus \mathcal{O}(\xi)$: the set of grandchildren of ξ and their descendants

- Data for and verbal description of the SPIHT-algorithm (2)
  - Given a significance level N, the *significance map* σ_N is defined as

$$\sigma_N : \xi \mapsto \begin{cases} 1 & \text{if } |C(\xi)| \geq N \\ 0 & \text{if } |C(\xi)| < N \end{cases}$$

  - Using σ_N one defines *significance* (w.r.t the significance level N) for sets $\mathcal{D}(\xi)$ and $\mathcal{L}(\xi)$ by mappings δ_N and λ_N:

$$\delta_N : \xi \mapsto \begin{cases} 1 & \text{if } \sigma_N(\eta) = 1 \text{ for at least one } \eta \in \mathcal{D}(\xi) \\ 0 & \text{otherwise} \end{cases}$$

$$\lambda_N : \xi \mapsto \begin{cases} 1 & \text{if } \sigma_N(\eta) = 1 \text{ for at least one } \eta \in \mathcal{L}(\xi) \\ 0 & \text{otherwise} \end{cases}$$

- Data for and verbal description of the SPIHT-algorithm (3)
  - The functions σ_N, δ_N and λ_N can be efficiently computed: for σ_N this is clear, for δ_N and λ_N this can be done inductively in a bottom-up manner starting at the leaves of T:

$$\delta_N(\xi) = \lambda_N(\xi) = 0 \quad \text{if } \xi \text{ is a leaf of } T, \text{ otherwise}$$

$$\lambda_N(\xi) = \max_{\eta \in \mathcal{O}(\xi)} \delta_N(\eta)$$

$$\delta_N(\xi) = \max\{\, \lambda_N(\xi), \max_{\eta \in \mathcal{O}(\xi)} \sigma_N(\eta) \,\}$$

- Data for and verbal description of the SPIHT-algorithm (4)
  - The lists
    - LIS (*list of insignificant sets*)
    - LIP (*list of insignificant points*)
    - LSP (*list of significant points*)

    are used for bookkeeping during the execution of the procedures *scantree* and *spiht*
    - $\xi \in$ LSP means : vertex ξ has been inspected by the algorithm and has been classified as *significant* (w.r.t. the current significance level, i.e., $\sigma_N(\xi) = 1$))
    - $\xi \in$ LIP means : vertex ξ has been inspected by the algorithm and has been classified as *insignificant* (w.r.t. the current significance level, i.e., $\sigma_N(\xi) = 0$)
    - entries in LIS are either of type (ξ, A) or of type (ξ, B), where (ξ, A) represents the set $\mathcal{D}(\xi)$ (ξ, B) represents the set $\mathcal{L}(\xi)$

- Data for and verbal description of the SPIHT-algorithm (5)
  - The current entries of LIS, LIP and LSP during execution of *scantree* constitute a partition of the *vertex set* of T into disjoint subsets:
    - entries $\xi \in$ LIP and $\xi \in$ LSP represent singleton sets $\{\xi\}$
    - an entry $(\xi, A) \in$ LIS represents the set $\mathcal{D}(\xi)$
    - an entry $(\xi, B) \in$ LIS represents the set $\mathcal{L}(\xi)$
  - The partition and the three lists are updated (i.e., the partition is refined) during the execution of *scantree*:

- Data for and verbal description of the SPIHT-algorithm (6)
  - If the top element of LIS is (ξ, A) and if the set $\mathcal{D}(\xi)$ is *significant*, i.e., $\delta_N(\xi)=1$, then
    - the offsprings $\eta \in \mathcal{O}(\xi)$ are examined for significance
    - the set $\mathcal{D}(\xi)$ is split into

$$\biguplus_{\eta \in \mathcal{O}(\xi)} \{\eta\} \uplus \mathcal{L}(\xi)$$

    - a new entry (ξ, B) added to LIS provided $\mathcal{L}(\xi) \neq 0$
  - If the top element of LIS is (ξ, B) and if the set $\mathcal{L}(\xi)$ is *significant*, i.e., $\lambda_N(\xi)=1$, then
    - the set $\mathcal{L}(\xi)$ is replaced by

$$\biguplus_{\eta \in \mathcal{O}(\xi)} \mathcal{D}(\eta),$$

    - entries (η, A) are added to LIS for all $\eta \in \mathcal{O}(\xi)$
  - Elements (ξ, B) or (ξ, A) thus added to LIS have to be examined in the same execution round of scantree.

- Data for and verbal description of the SPIHT-algorithm (7)
  - Algorithm *spiht* iterates the sorting procedure *scantree* together with a refinement procedure in the spirit of *progressive transmission*
  - At the beginning of each execution of *scantree*
    - the current elements of the list LIP are tested for significance w.r.t. to the actual significance level N, which is supposed to be lower than the level used in the previous execution of *scantree*
    - In case a vertex is found *N-significant*, it will be moved from LIP to LSP
  - During the execution of *scantree*
    - information about the significance of vertices (and the corresponding sign in the case of vertices found *significant*) and $\mathcal{D}$- and $\mathcal{L}$-sets is encoded in the bit stream *output*

- Data for and verbal description of the SPIHT-algorithm (8)
  - *spith* is initialized by
    - computing the maximal number r of bits needed to represent the absolute values of the $C(\xi)$,
    - setting the significance level for the first run of *scantree* to $N = 2^r$
      (so that there are indeed entries $C(\xi)$ with $2^r \leq |C(\xi)| < 2^{r+1}$ and no entries with $2^{r+1} \leq |C(\xi)|$ in T)
    - LIS and LIP are initialized using the root vertices of T, LSP is initially empty
  - The first call of *scantree* examines T with $N = 2^r$ as significance level.
    T is explored only to the extent needed for discovering all N-significant vertices of (T, C).
    After this first *sorting phase* there is *no refinement phase* necessary since LSP was empty before calling *scantree*

- Data for and verbal description of the SPIHT-algorithm (9)
  - The exploration of (T, C) is then done by iteratively by
    - *sorting*: calling *scantree* with the significance level halved at every iteration
    - *refinement*: enhancing the bit information of previous added elements of LSP in a *progressive transmission* way

    Each iteration uses the information accumulated in the lists LIS, LIP, and LSP, and updates these lists
  - Execution of *spith* can be stopped after every *sorting-refinement* round.
    The output stream produced up to this point is sufficient to reconstruct the respective k-bit approximation of C.

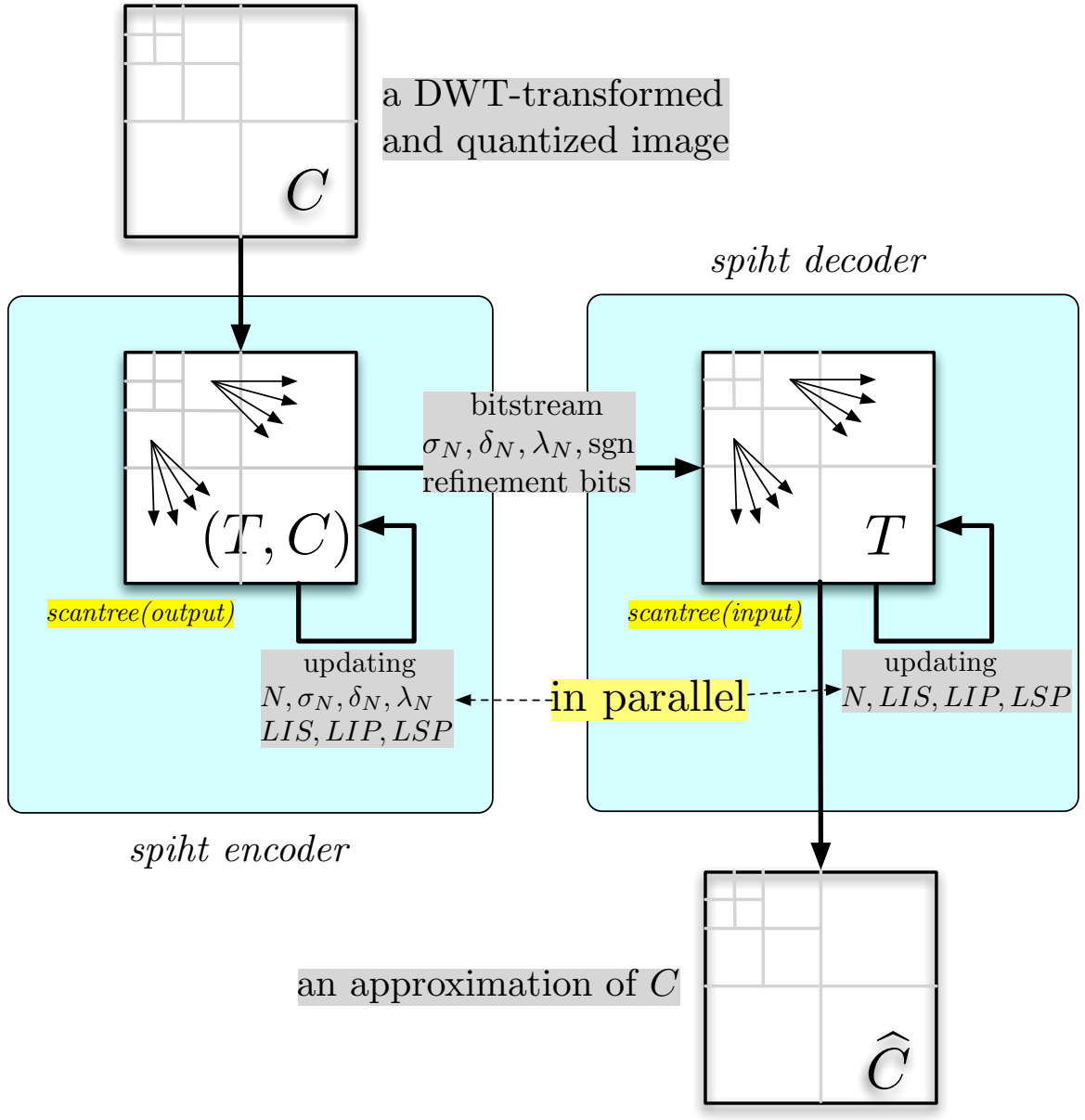


Figure: The SPIHT encoding-decoding scheme

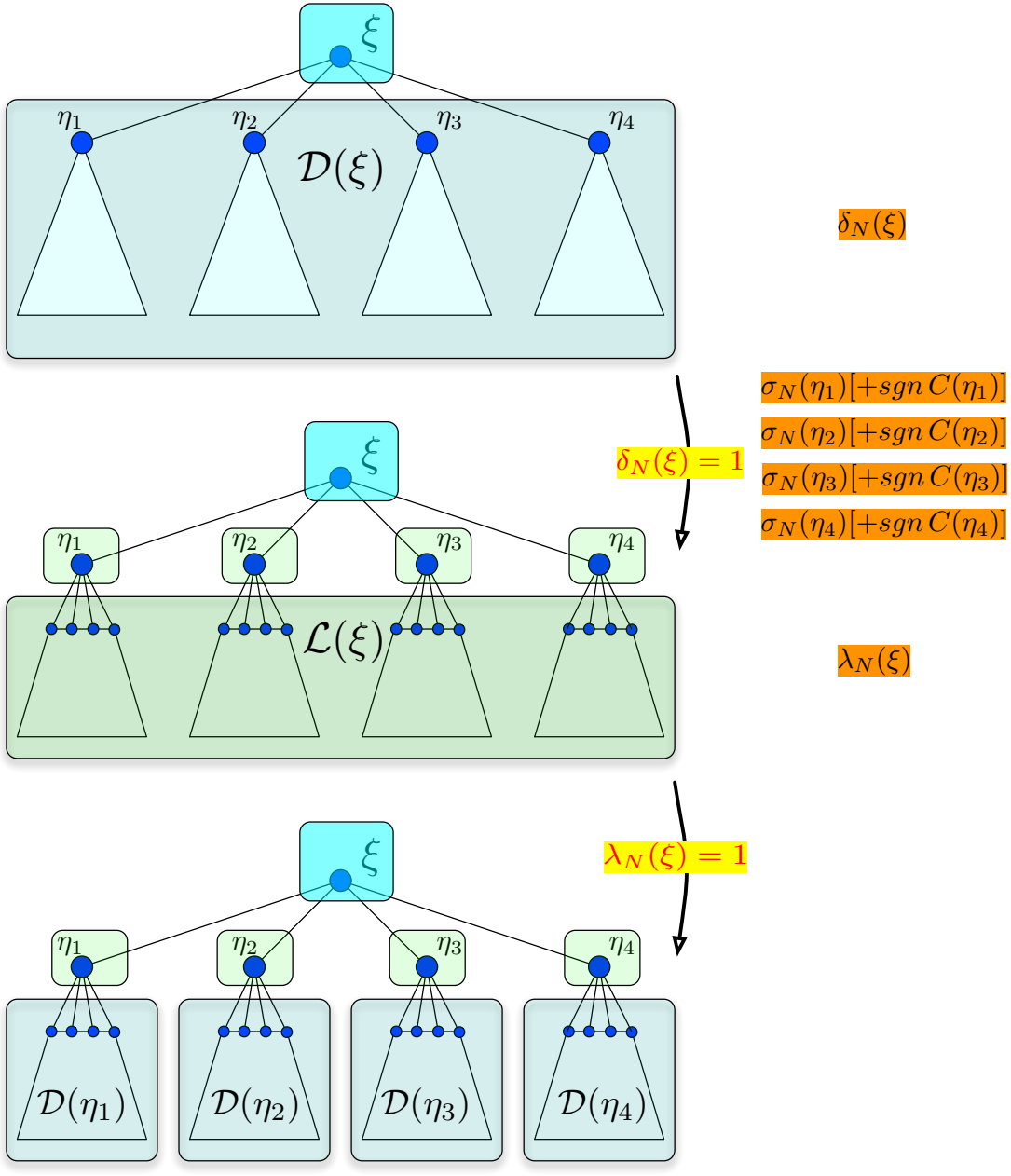


Figure: The set partitioning strategy of SPIHT

procedure *scantree*(T,C,N,LIS,LIP,LSP)

(* T: a forest of rooted trees, C: an integer valuation of the vertices of T, N: significance level
LIS, LIP, LSP: lists that are updated during the execution*)

```
for ξ in LIP do
    output(σ_N(ξ))
    if σ_N(ξ) = 1 then
        output(sgn(C(ξ))), remove(LIP,ξ), append(LSP,ξ)
    end if
end for
for ξ in LIS do
    if type(ξ,A) then
        output(δ_N(ξ))
        if δ_N(ξ) = 1 then
            for η in O(ξ) do
                output(σ_N(η))
                if σ_N(η) = 1 then
                    output(sgn(C(η))), append(LSP,η)
                else
                    append(LIP,η)
                end if
            end for
            remove(LIS,ξ-A)
            if L(ξ) ≠ ∅ then
                append(LIS,ξ-B)
            end if
        end if
    end if
    if type(ξ,B) then
        output(λ_N(ξ))
        if λ_N(ξ) = 1 then
            remove(LIS,ξ-B)
            for η ∈ O(ξ) do
                append(LIS,η-A)
            end for
        end if
    end if
end for
return (LIS,LIP,LSP)
```

procedure *spiht*(T,C)

(* T: a forest of rooted trees
C: an integer valuation of the vertices of T *)

(*initialization*)
$r \leftarrow \lfloor \log_2 \max_{\xi \in T} |C(\xi)| \rfloor$
output(r)
$N \leftarrow 2^r$
LIS $\leftarrow$ list of all ξ-A, where ξ is the root of a T-subtree with $\mathcal{D}(\xi) \neq \emptyset$
LIP $\leftarrow$ list of all ξ, which are roots of subtrees of T,
including isolated vertices
LSP $\leftarrow$ empty list
(*sorting*)
(LIS,LIP,LSP) $\leftarrow$ *scantree*(T,C,N,LIS,LIP,LSP)
oldLSP $\leftarrow$ LSP
repeat
$r \leftarrow r - 1$, $N \leftarrow N/2$
(*sorting*)
(LIS,LIP,LSP) $\leftarrow$ *scantree*(T,C,N,LIS,LIP,LSP)
(*refinement*)
for $\xi \in$ oldLSP **do**
output(r-th bit of $|C(\xi)|$)
end for
oldLSP $\leftarrow$ LSP
until "as needed"

- Ex. 1

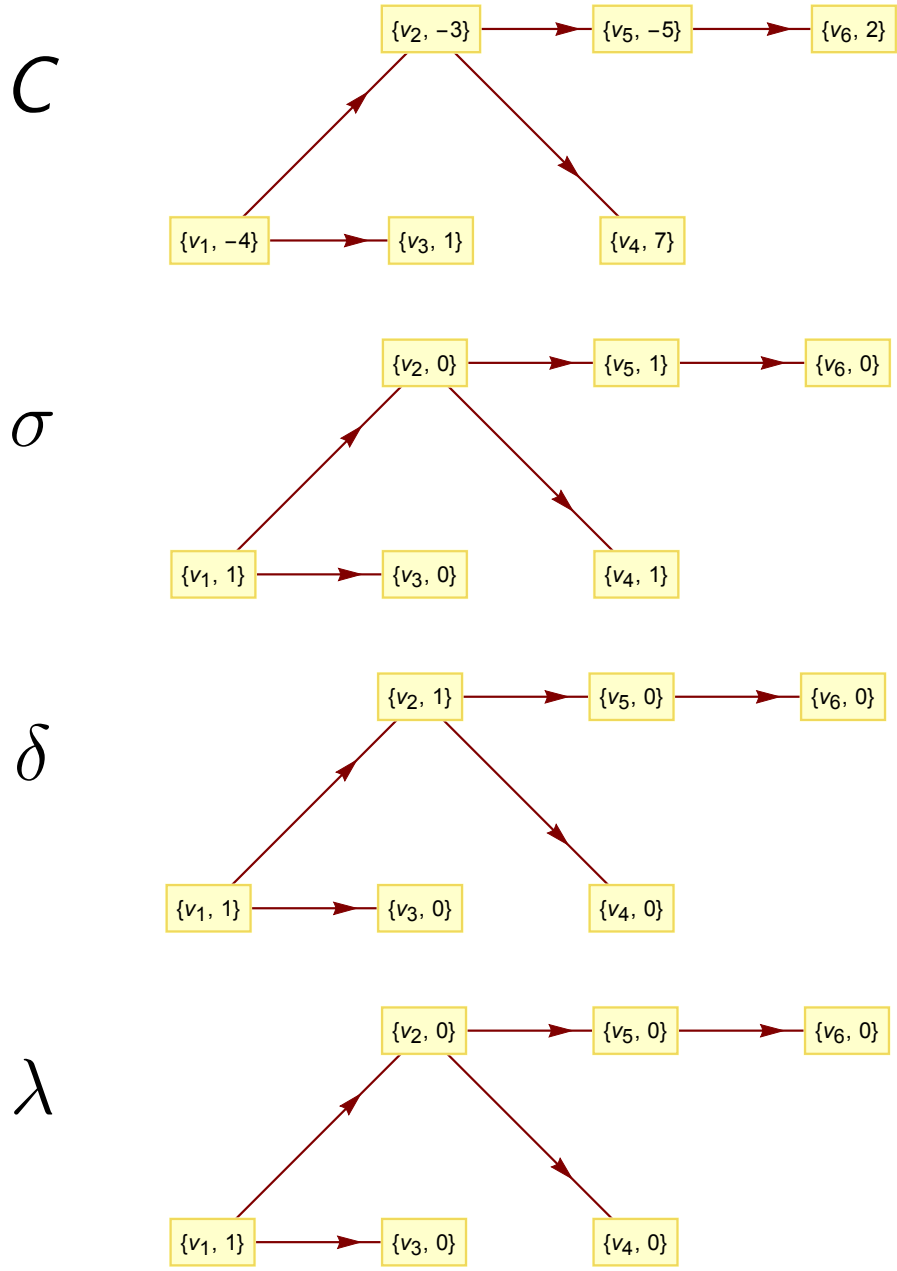
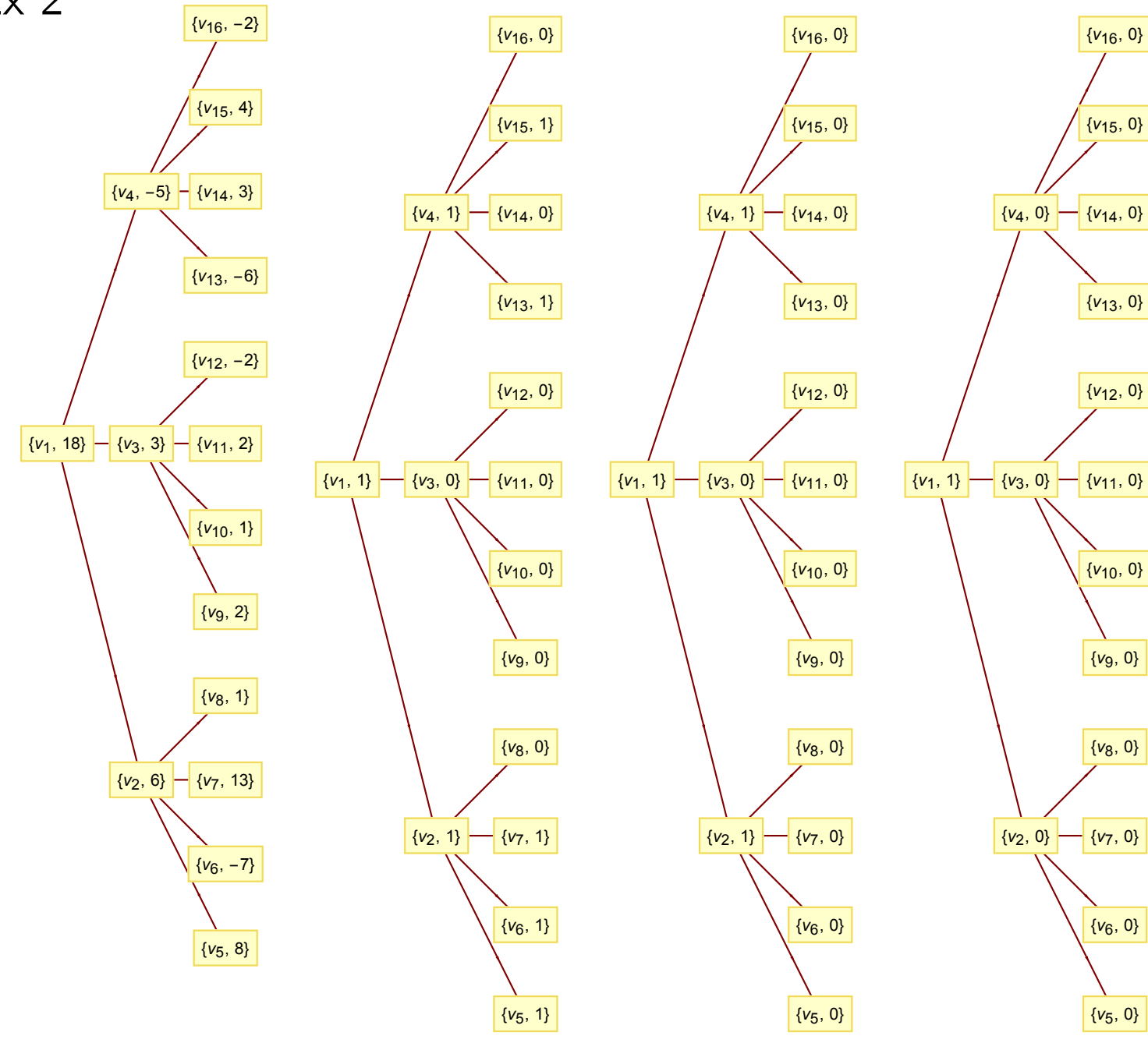


Figure: A tree T, its valuation C and the σ, δ, λ mappings of (T, C) for significance level $N = 4$

Execution of *scantree*[T, C, 4, {{1, A}}, {1}, {}, 1]
out: $\sigma[1] = 1$
out: *sign*$[-4]$
LSP$= \{1\}$ LIP$= \{\}$
LIS: $\{\{1, A\}\}$
out: $\delta[1] = 1$
out: $\sigma[2] = 0$
LSP$= \{1\}$ LIP$= \{2\}$
out: $\sigma[3] = 0$
LSP$= \{1\}$ LIP$= \{2, 3\}$
LIS: $\{\{1, B\}\}$
out: $\lambda[1] = 1$
LIS: $\{\{2, A\}, \{3, A\}\}$
out: $\delta[2] = 1$
out: $\sigma[4] = 1$
out: *sign*$[7]$
LSP$= \{1, 4\}$ LIP$= \{2, 3\}$
out: $\sigma[5] = 1$
out: *sign*$[-5]$
LSP$= \{1, 4, 5\}$ LIP$= \{2, 3\}$
LIS: $\{\{3, A\}, \{2, B\}\}$
out: $\delta[3] = 0$
LIS: $\{\{2, B\}\}$
out: $\lambda[2] = 0$
new LIS: $\{\{3, A\}, \{2, B\}\}$
new LIP: $\{2, 3\}$
new LSP: $\{1, 4, 5\}$
output: $\{1,$ *sign*$[-4], 1, 0, 0, 1, 1, 1,$ *sign*$[7], 1,$ *sign*$[-5], 0, 0\}$

- Ex 2


{v16, −2}
{v15, 4}
{v4, −5}
{v14, 3}
{v13, −6}
{v12, −2}
{v1, 18}
{v3, 3}
{v11, 2}
{v10, 1}
{v9, 2}
{v8, 1}
{v2, 6}
{v7, 13}
{v6, −7}
{v5, 8}
{v16, 0}
{v15, 1}
{v4, 1}
{v14, 0}
{v13, 1}
{v12, 0}
{v1, 1}
{v3, 0}
{v11, 0}
{v10, 0}
{v9, 0}
{v8, 0}
{v2, 1}
{v7, 1}
{v6, 1}
{v5, 1}
{v16, 0}
{v15, 0}
{v4, 1}
{v14, 0}
{v13, 0}
{v12, 0}
{v1, 1}
{v3, 0}
{v11, 0}
{v10, 0}
{v9, 0}
{v8, 0}
{v2, 1}
{v7, 0}
{v6, 0}
{v5, 0}
{v16, 0}
{v15, 0}
{v4, 0}
{v14, 0}
{v13, 0}
{v12, 0}
{v1, 1}
{v3, 0}
{v11, 0}
{v10, 0}
{v9, 0}
{v8, 0}
{v2, 0}
{v7, 0}
{v6, 0}
{v5, 0}

```
scantree[T2, C2, 16, {{1,A}}, {1}, {}, 1]

out: \[Sigma][1]= 1
out: sign[18]
LSP= {1} LIP= {}
LIS: {{1,A}}
out: \[Delta][1]= 0

new LIS: {{1,A}}
new LIP: {}
new LSP: {1}
output: {1,sign[18],0}
```

```
scantree[T2, C2, 8, {{1,A}}, {}, {1}, 1]

LIS: {{1,A}}
out: \[Delta][1]= 1
out: \[Sigma][2]= 0
LSP= {1} LIP= {2}
out: \[Sigma][3]= 0
LSP= {1} LIP= {2,3}
out: \[Sigma][4]= 0
LSP= {1} LIP= {2,3,4}
LIS: {{1,B}}
out: \[Lambda][1]= 1
LIS: {{2,A},{3,A},{4,A}}
out: \[Delta][2]= 1
out: \[Sigma][5]= 1
out: sign[8]
```

```
LSP= {1,5} LIP= {2,3,4}
out: \[Sigma][6]= 0
LSP= {1,5} LIP= {2,3,4,6}
out: \[Sigma][7]= 1
out: sign[13]
LSP= {1,5,7} LIP= {2,3,4,6}
out: \[Sigma][8]= 0
LSP= {1,5,7} LIP= {2,3,4,6,8}
LIS: {{3,A},{4,A}}
out: \[Delta][3]= 0
LIS: {{4,A}}
out: \[Delta][4]= 0

new LIS: {{3,A},{4,A}}
new LIP: {2,3,4,6,8}
new LSP: {1,5,7}
output: {1,0,0,0,1,1,1,sign[8],0,1,sign[13],0,0,0}
```

```
scantree[T2, C2, 4, {{3,A},{4,A}},{2,3,4,6,8},{1,5,7},1]

out: \[Sigma][2]= 1
out: sign[6]
LSP= {1,5,7,2} LIP= {}
out: \[Sigma][3]= 0
LSP= {1,5,7,2} LIP= {3}
out: \[Sigma][4]= 1
out: sign[-5]
LSP= {1,5,7,2,4} LIP= {3}
out: \[Sigma][6]= 1
out: sign[-7]
LSP= {1,5,7,2,4,6} LIP= {3}
out: \[Sigma][8]= 0
LSP= {1,5,7,2,4,6} LIP= {3,8}
LIS: {{3,A},{4,A}}
out: \[Delta][3]= 0
LIS: {{4,A}}
```

```
out: \[Delta][4]= 1
out: \[Sigma][13]= 1
out: sign[-6]
LSP= {1,5,7,2,4,6,13} LIP= {3,8}
out: \[Sigma][14]= 0
LSP= {1,5,7,2,4,6,13} LIP= {3,8,14}
out: \[Sigma][15]= 1
out: sign[4]
LSP= {1,5,7,2,4,6,13,15} LIP= {3,8,14}
out: \[Sigma][16]= 0
LSP= {1,5,7,2,4,6,13,15} LIP= {3,8,14,16}

new LIS: {{3,A}}
new LIP: {3,8,14,16}
new LSP: {1,5,7,2,4,6,13,15}

output: {1,sign[6],0,1,sign[-5],1,sign[-7],0,0,1,1,
         sign[-6],0,1,sign[4],0}
```

```
scantree[T2, C2, 2, {{3,A}}, {3,8,14,16},{1,5,7,2,
         4,6,13,15},1]

out: \[Sigma][3]= 1
out: sign[3]
LSP= {1,5,7,2,4,6,13,15,3} LIP= {}
out: \[Sigma][8]= 0
LSP= {1,5,7,2,4,6,13,15,3} LIP= {8}
out: \[Sigma][14]= 1
out: sign[3]
LSP= {1,5,7,2,4,6,13,15,3,14} LIP= {8}
out: \[Sigma][16]= 1
out: sign[-2]
LSP= {1,5,7,2,4,6,13,15,3,14,16} LIP= {8}
LIS: {{3,A}}
out: \[Delta][3]= 1
out: \[Sigma][9]= 1
out: sign[2]
```

```
LSP= {1,5,7,2,4,6,13,15,3,14,16,9} LIP= {8}
out: \[Sigma][10]= 0
LSP= {1,5,7,2,4,6,13,15,3,14,16,9} LIP= {8,10}
out: \[Sigma][11]= 1
out: sign[2]
LSP= {1,5,7,2,4,6,13,15,3,14,16,9,11} LIP= {8,10}
out: \[Sigma][12]= 1
out: sign[-2]
LSP= {1,5,7,2,4,6,13,15,3,14,16,9,11,12} LIP= {8,10}

new LIS: {}
new LIP: {8,10}
new LSP: {1,5,7,2,4,6,13,15,3,14,16,9,11,12}

output: {1,sign[3],0,1,sign[3],1,sign[-2],1,1,sign[2],
         0,1,sign[2],1,sign[-2]}
```

```
scantree[T2, C2, 1, {}, {8,10}, {1,5,7,2,4,6,13,15,
        3,14,16,9,11,12}, 1]

out: \[Sigma][8]= 1
out: sign[1]
LSP= {1,5,7,2,4,6,13,15,3,14,16,9,11,12,8} LIP= {}
out: \[Sigma][10]= 1
out: sign[1]
LSP= {1,5,7,2,4,6,13,15,3,14,16,9,11,12,8,10} LIP= {}
new LIS: {}
new LIP: {}
new LSP: {1,5,7,2,4,6,13,15,3,14,16,9,11,12,8,10}

output: {1,sign[1],1,sign[1]}
```

Ex 3 : execution of the SPIHT-algorithm

following Rao, Bopardikar: Wavelet Transforms, Addison-Wesley, 1998.

$$\begin{bmatrix} 62 & 34 & 18 & 17 & -4 & 1 & -2 & 6 \\ -31 & 24 & -15 & 14 & -11 & 0 & 4 & -1 \\ 42 & 29 & -35 & 10 & 29 & 10 & 6 & 9 \\ -12 & 15 & -9 & 15 & -1 & 9 & 5 & 13 \\ 4 & 45 & 13 & -1 & 26 & -21 & 3 & 1 \\ 3 & 0 & -2 & 21 & -1 & 0 & 7 & 9 \\ 0 & 13 & 4 & 5 & 4 & 5 & 6 & 0 \\ -1 & 7 & -11 & 3 & 0 & 8 & 2 & 7 \end{bmatrix}$$

- 2-level DWT-decomposition
- bound for coefficient size: $\lfloor \log_2(62) \rfloor = 5 \Rightarrow N = 2^5 = 32$

LIS	LIP	LSP
(0,1)-A	(0,0)	
(1,0)-A	(0,1)	
(1,1)-A	(1,0)	
	(1,1)	

Figure: Initializing the lists

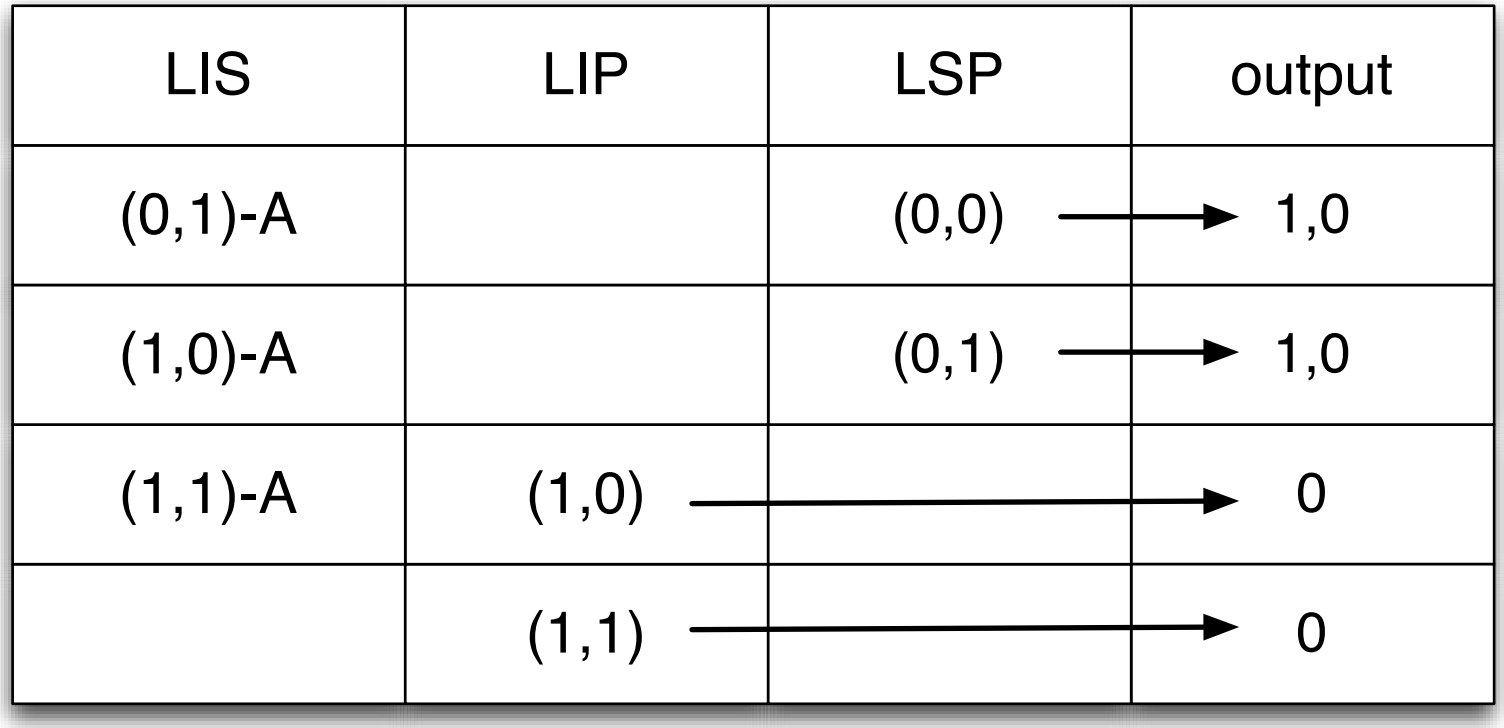


Figure: Lists after the first N-significance test for LIP

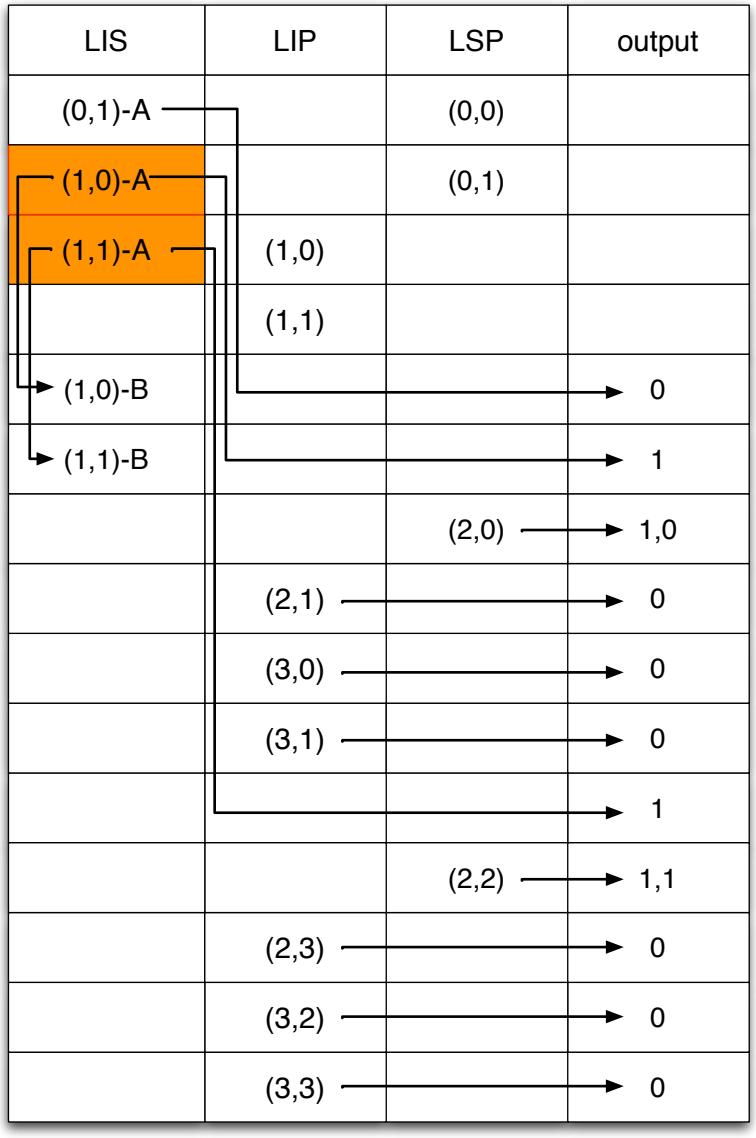


Figure: Lists after the first scanning of LIS-entries of type A

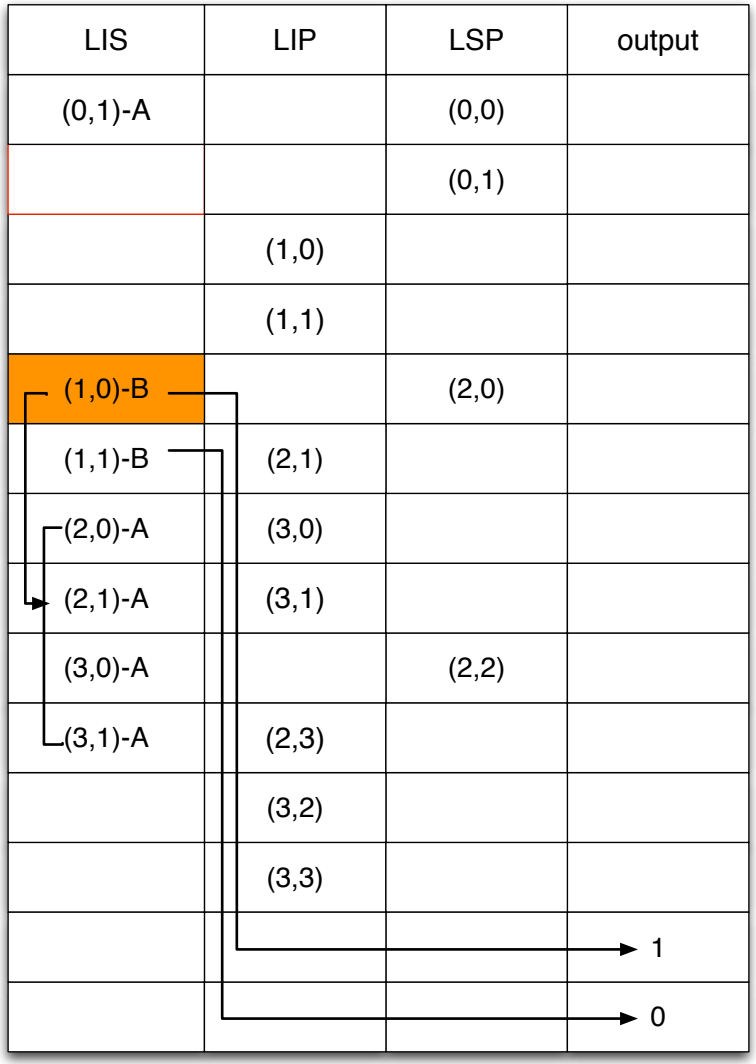


Figure: Lists after the first scanning of LIS-entries of type B

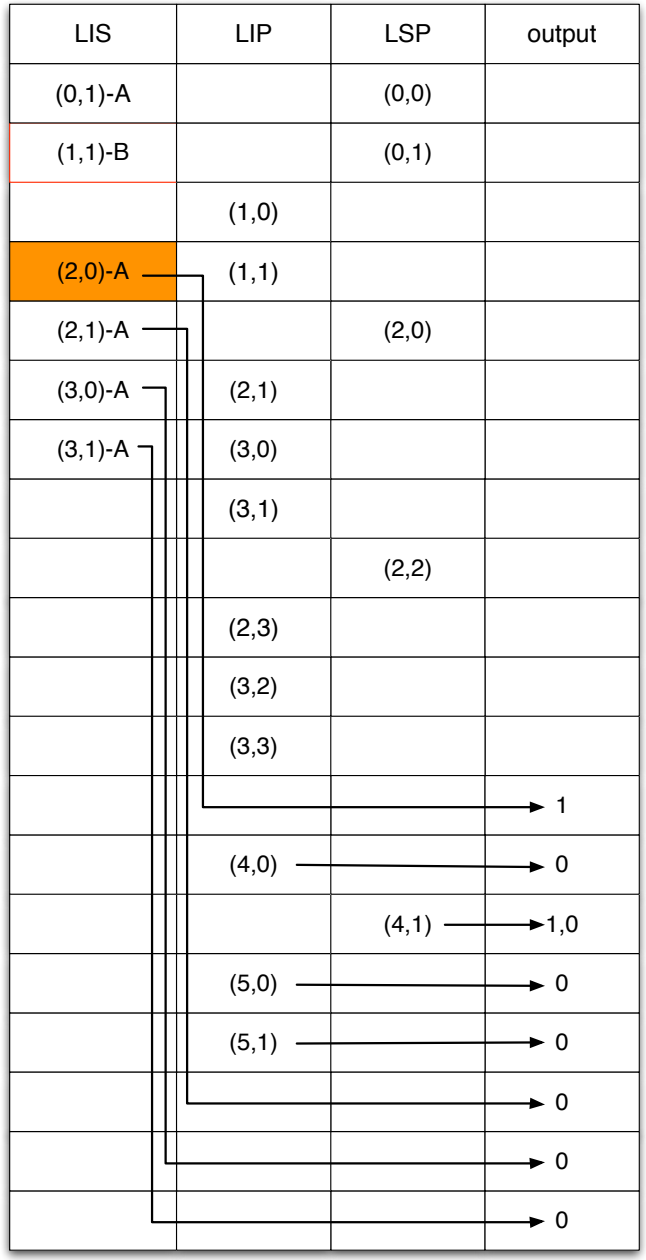


Figure: Lists after finishing the first execution of *scantree*

- The first execution of *scantree* is finished.
- After this round there is no *refinement* done!
- The significance threshold for the next round is set to $N = 16$.

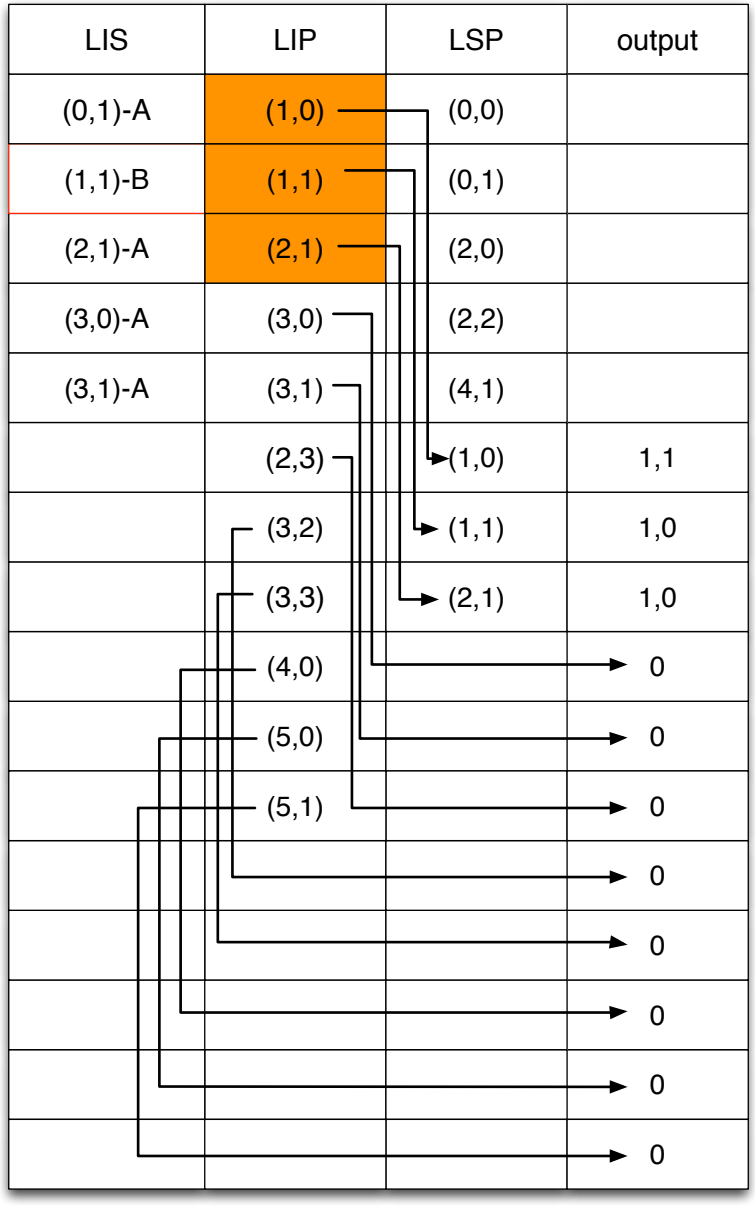
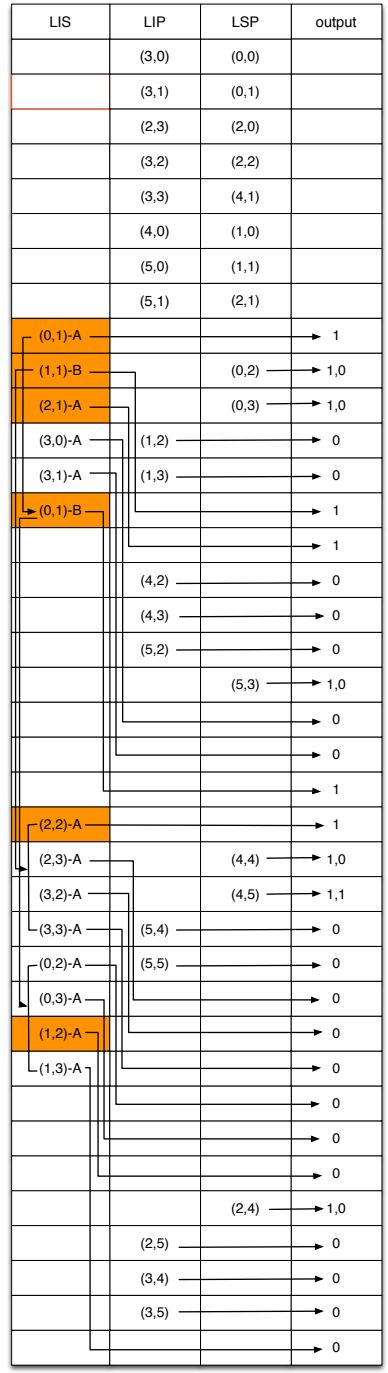


Figure: Lists after the significance test for LIP in the second round

- Following slide: Lists after the second round of *scantree*
- This is followed by *refinement*: output of the 4th bits of the LSP-entries from the first round: $1, 0, 1, 0, 1$.


LIS
LIP
LSP
output
(3,0)
(0,0)
(3,1)
(0,1)
(2,3)
(2,0)
(3,2)
(2,2)
(3,3)
(4,1)
(4,0)
(1,0)
(5,0)
(1,1)
(5,1)
(2,1)
(0,1)-A
1
(1,1)-B
(0,2)
1,0
(2,1)-A
(0,3)
1,0
(3,0)-A
(1,2)
0
(3,1)-A
(1,3)
0
(0,1)-B
1
1
(4,2)
0
(4,3)
0
(5,2)
0
(5,3)
1,0
0
0
1
(2,2)-A
1
(2,3)-A
(4,4)
1,0
(3,2)-A
(4,5)
1,1
(3,3)-A
(5,4)
0
(0,2)-A
(5,5)
0
(0,3)-A
0
(1,2)-A
0
(1,3)-A
0
0
0
0
(2,4)
1,0
(2,5)
0
(3,4)
0
(3,5)
0
0