Tree Decomposition methods

Chapter 9

Graph concepts reviews: Hyper graphs and dual graphs

- **A hypergraph** is $H = (V,S)$, $V = \{v_1,..,v_n\}$ and a set of subsets **Hyperegdes**: $S = \{S_1, ..., S_l\}$.

- **Dual graphs** of a hypergaph: The nodes are the hyperedges and a pair of nodes is connected if they share vertices in V. The arc is labeled by the shared vertices.

- **A primal graph** of a hypergraph $H = (V,S)$ has V as its nodes, and any two nodes are connected by an arc if they appear in the same hyperedge.

- if all the constraints of a network R are binary, then its hypergraph is identical to its primal graph.
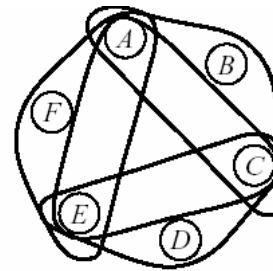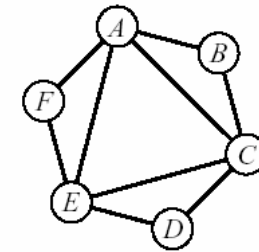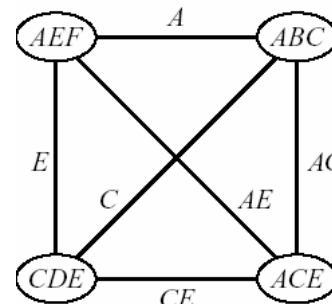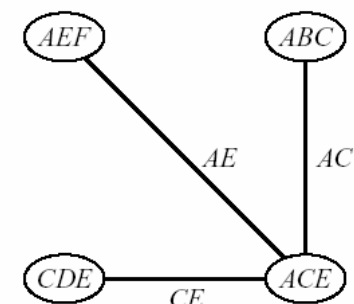
Acyclic Networks

- **The running intersection property (connectedness):** An arc can be removed from the dual graph if the variables labeling the arcs are shared along an alternative path between the two endpoints.

- **Join graph**: An arc subgraph of the dual graph that satisfies the connectedness property.

- **Join-tree:** a join-graph with no cycles

- **Hypertree:** A hypergraph whose dual graph has a join-tree.

- **Acyclic network:** is one whose hypergraph is a hypertree.
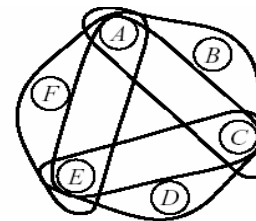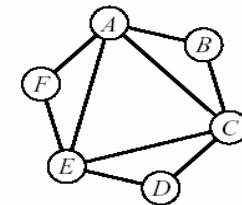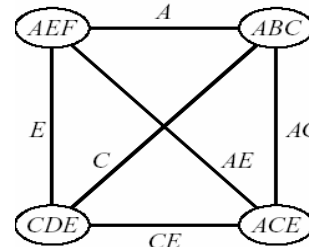
Solving acyclic networks

- Algorithm **_acyclic-solving_** applies a tree algorithm to the join-tree. It applies directional relational arc-consistency from leaves to root.

- **Complexity**: acyclic-solving is $O(r\ l\ log\ l)$ steps, where r is the number of constraints and l bounds the number of tuples in each constraint relation

Example

- Constraints are:
- $R\_\{ABC\} = R\_\{AEF\} = R\_\{CDE\} = \{(0,0,1)\ (0,1,0)(1,0,0)\}$
- $R\_\{ACE\} = \{\ (1,1,0)\ (0,1,1)\ (1,0,1)\ \}$.

- d= (R_{ACE},R_{CDE},R_{AEF},R_{ABC}).
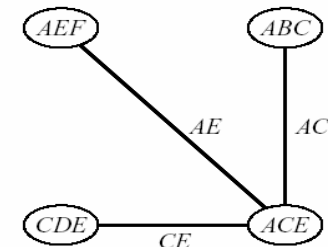   - When processing R_{ABC}, its parent relation is R_{ACE};

    $$R_{ACE} = \pi_{ACE}(R_{ACE} \otimes R_{ABC}) = \{(0,1,1)(1,0,1)\}$$

    - processing R_{AEF} we generate relation

    $$R_{ACE} = \pi_{ACE}(R_{ACE} \otimes R_{AEF}) = \{(0,1,1)\}$$

        - processing R_{CDE} we generate:
        - R_{ACE} = \pi_{ACE} (R_{ACE} x R_{CDE}) = {(0,1,1)}.
- A solution is generated by picking the only allowed tuple for R_{ACE}, A=0,C=1,E=1, extending it with a value for D that satisfies R_{CDE}, which is only D=0, and then similarly extending the assignment to F=0 and B=0, to satisfy R_{AEF} and R_{ABC}.

Recognizing acyclic networks

- **Dual-based recognition**:
  - Perform maximal spanning tree over the dual graph and check connectedness of the resulting tree.
  - Dual-acyclicity complexity is $O(e^3)$
- **Primal-based recognition**:
  - **Theorem** (Maier 83): A hypergraph has a join-tree iff its primal graph is chordal and conformal.
  - A chordal primal graph is **conformal** relative to a constraint hypergraph iff there is a one-to-one mapping between maximal cliques and scopes of constraints.

Primal-based recognition
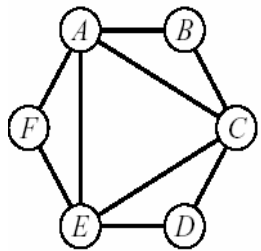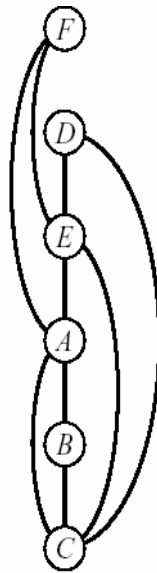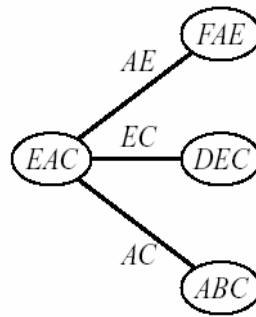
(a) (b) (c)

- Check chordality using max-cardinality ordering.
- Test conformality
- Create a join-tree: connect every clique to an earlier clique sharing maximal number of variables.

Tree-based clustering

- Convert a constraint problem to an acyclic-one: group subset of constraints to clusters until we get an acyclic problem.

- Hypertree embedding of a hypergraph $H = (X,H)$ is a hypertree $S = (X, S)$ s.t., for every h in H there is h_1 in S s.t. h is included in h_1.

- This yield algorithm join-tree clustering

Join-tree clustering

- **Input**: A constraint problem R =(X,D,C) and its primal graph G = (X,E).
- **Output**: An equivalent acyclic constraint problem and its join-tree: T= (X,D, {C '})
- 1. Select an d = (x_1,...,x_n)
- 2. **Triangulation**(create the induced graph along d and call it G^*:)
-     for j=n to 1 by -1 do
-         E ← E U {(i,k)| (i,j) in E,(k,j) in E }
- 3. **Create a join-tree of the induced graph G^*:**
-     a. Identify all maximal cliques (each variable and its parents is a clique).
-         Let C_1,...,C_t be all such cliques,
-   b. Create a tree-structure T over the cliques:
-      Connect each C_{i} to a C_{j} (j < I) with whom it shares largest subset of variables.
- 4. Place each input constraint in one clique containing its scope, and let
-   P_i be the constraint subproblem associated with C_i.
- 5. Solve P_i and let {R'}_i $ be its set of solutions.
- 6. Return C' = {R'}_1,..., {R'}_t
- the new set of constraints and their join-tree, T.

- Size of maximal clique - 1 is the Induced width.

Example of tree-clustering
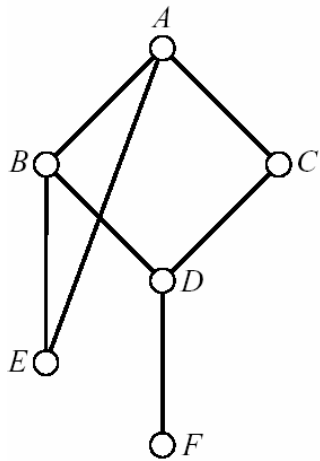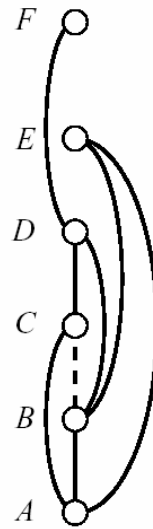
(a)   (b)   (c)
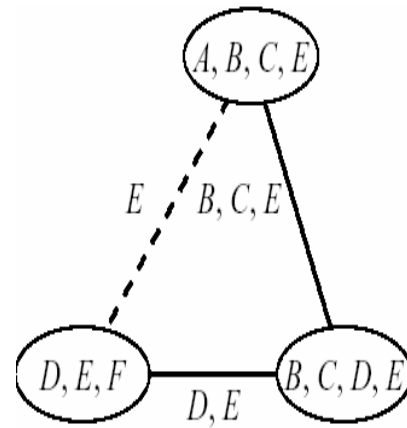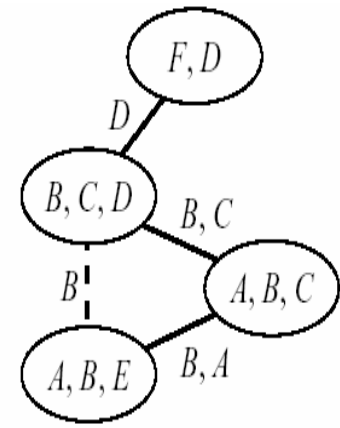
(a)   (b)

Unifying tree-decompositions

- A tree-decomposition of R = (X, D, C) is a triple $<T, \chi, \psi>$ where $T = <V, E>$ is a tree, and χ and ψ are sets of functions .
  - For each constraint $R_i \in C$ there is at least one vertex v in T such that $R_i \in \psi(v)$ and $scope(R_i) \subseteq \chi(v)$
  - For each variable x in X, the set $\{v \in V \mid x \in \chi(v)\}$ induces a connected subtree of T. (This is the connectedness property.)
- **tree-width** = max number of vars in a cluster
- **hyper-width** = is max functions in a cluster
- the **separator of u and v**: the intersection between variables in u and v.

Example of two join-trees again

(a)

(b)

$\{R_{AB}, R_{AC}, R_{BE}\}$

$\{R_{FD}\}$

A, B, C, E

F, D

D, E, F

B, C, D $\{R_{BD}, R_{CD}\}$

B, C, D, E

$\{R_{DF}\}$ $\{R_{BD}, R_{DC}, R_{AE}\}$

$\{R_{AB}, R_{AC}\}$ A, B, C

A, B, E

$\{R_{BE}, R_{AE}\}$
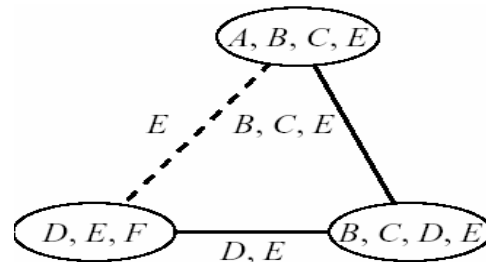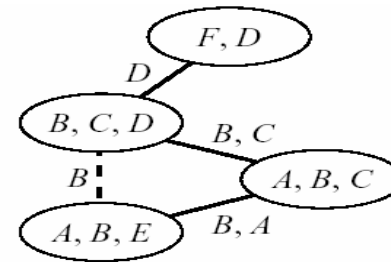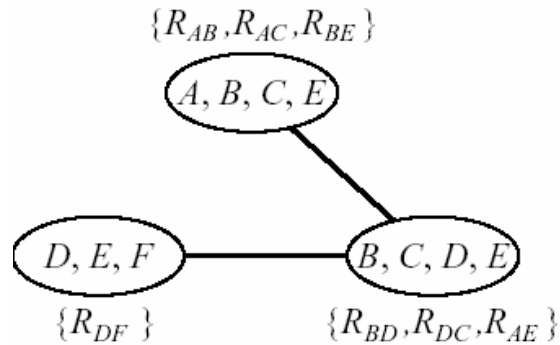
(a)
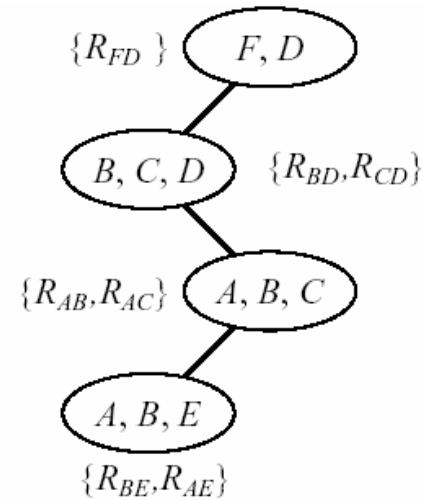
(b)

Cluster Tree Elimination

- Cluster Tree Elimination *(CTE)* works by passing messages along a tree-decomposition

- Basic idea:
  - Each node sends one message to each of its neighbors
  - Node u sends a message to its neighbor v only when u received messages from all its other neighbors

Constraint Propagation
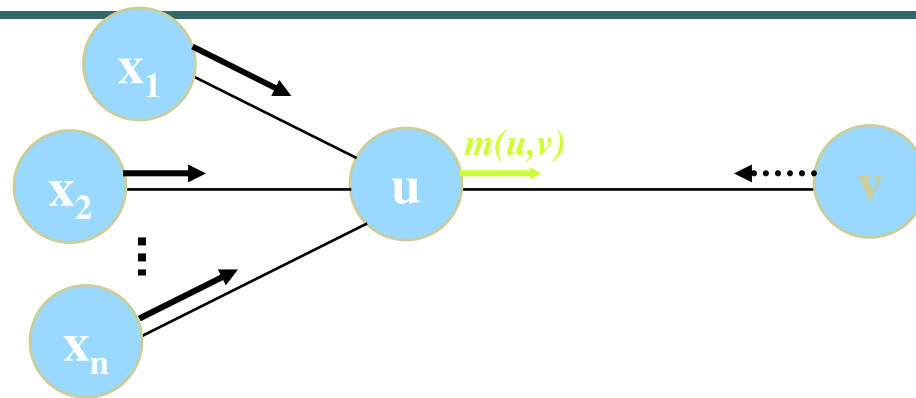
$$cluster(u) = \psi(u) \cup \{m(x_1, u), m(x_2, u), ..., m(x_n, u), m(v, u)\}$$

Compute the message :

$$m_{(u,v)} = \pi_{sep(u,v)} (\otimes_{R_i \in cluster(u)} R_i)$$

Example of CTE message propagation

Diagram of clique tree nodes:

- **1**: F, D
- **2**: B, C, D
- **3**: A, B, C
- **4**: A, B, E

$$m_{(1,2)}(D) = \pi_D(R_{FD})$$

$$m_{(2,1)}(D) = \pi_D(R_{BD} \bowtie R_{CD} \bowtie m_{(3,2)})$$

$$m_{(2,3)}(B,C) = \pi_{BC}(R_{BD} \bowtie R_{CD} \bowtie m_{(1,2)})$$

$$m_{(3,2)}(B,C) = \pi_{BC}(R_{AB} \bowtie R_{AC} \bowtie m_{(4,3)})$$

$$m_{(3,4)}(A,B) = \pi_{AB}(R_{AB} \bowtie R_{AC} \bowtie m_{(2,3)})$$

$$m_{(4,3)}(A,B) = \pi_{AB}(R_{BE} \bowtie R_{AE})$$
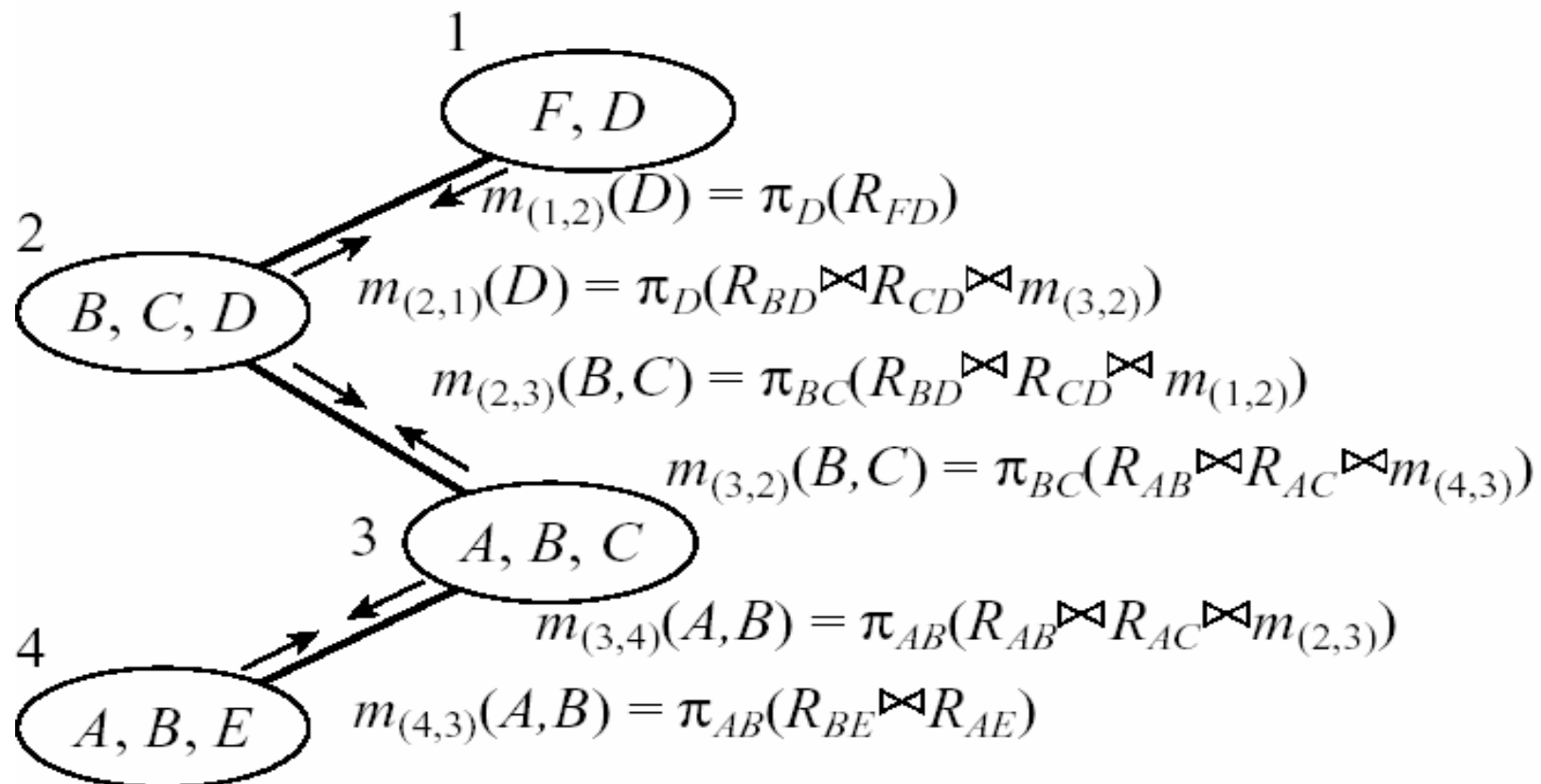
Distributed relational arc-consistency example

The message that R2 sends to R1 is

R1 updates its relation and domains and sends messages to neighbors

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2

R_1

A
1
2
3

R_3

A	C
1	2
3	2

R_5

B	C	F
1	2	3
3	2	1

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_6

D	F	G
1	2	3
2	1	3
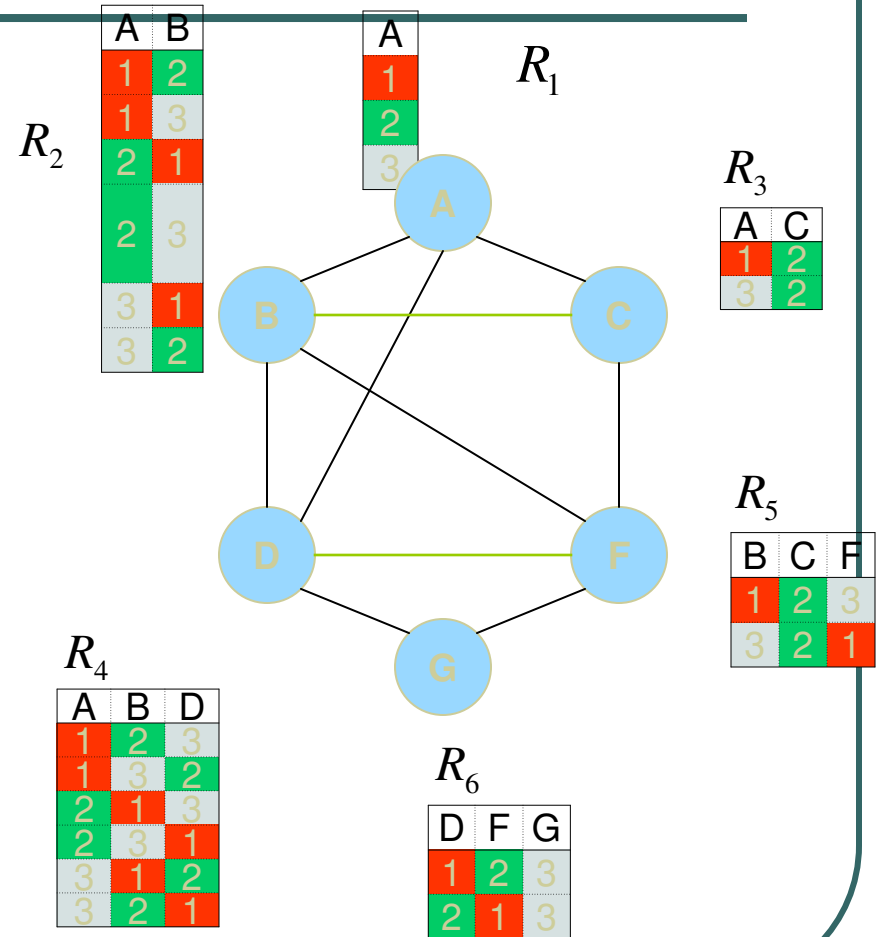
Distributed Arc-Consistency

DR-AC can be applied to the dual problem of any constraint network.

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \qquad (1)$$

$$D_i \leftarrow D_i \cap (\bowtie_{k \in ne(i)} D_k^i) \qquad (2)$$
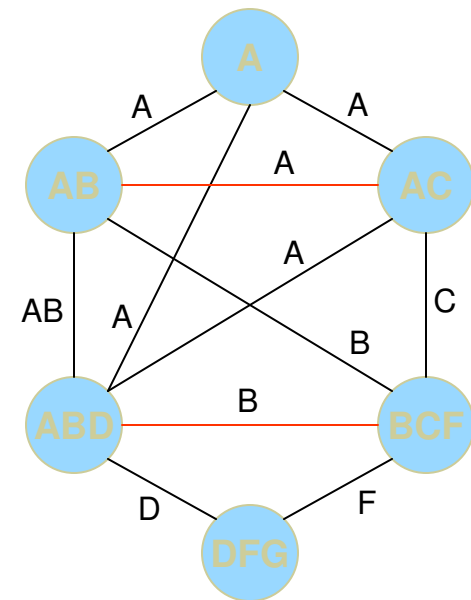
b) Constraint network

DR-AC on a dual join-graph

R_1

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2

R_3

A	C
1	2
3	2

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
1	2	3
2	1	3

Iteration 1

R_1

A
1
2
3

h_2^1

A
1
2
3

h_3^1

A
1
3

h_4^1

A
1
2
3

h_5^2

B
1
3

h_4^2

B
1
2
3

h_1^2

A
1
2
3

R_2

A	B
1	2
1	3
2	1
2	3
3	1
3	2

R_3

A	C
1	2
3	2

h_1^3

A
1
2
3

h_5^3

C
2

h_6^4

D
1
2

h_2^4

B
1
2
3

h_1^4

A
1
2
3

R_4

A	B	D
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

h_2^5

B
1
2
3

h_3^5

C
2

h_6^5

F
1
3

R_6

D	F	G
1	2	3
2	1	3

h_4^6

D
1
2
3

h_5^6

F
1
3

Graph nodes:
- 1: A
- 2: AB
- 3: AC
- 4: ABD
- 5: BCF
- 6: DFG

Edges labeled: A, A, B, A, B, C, D, F
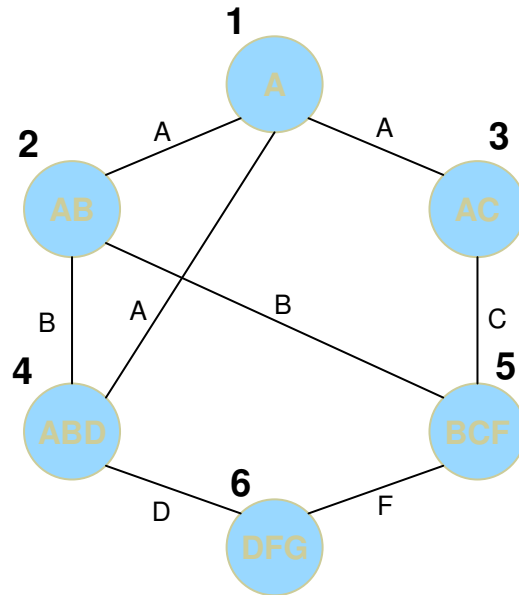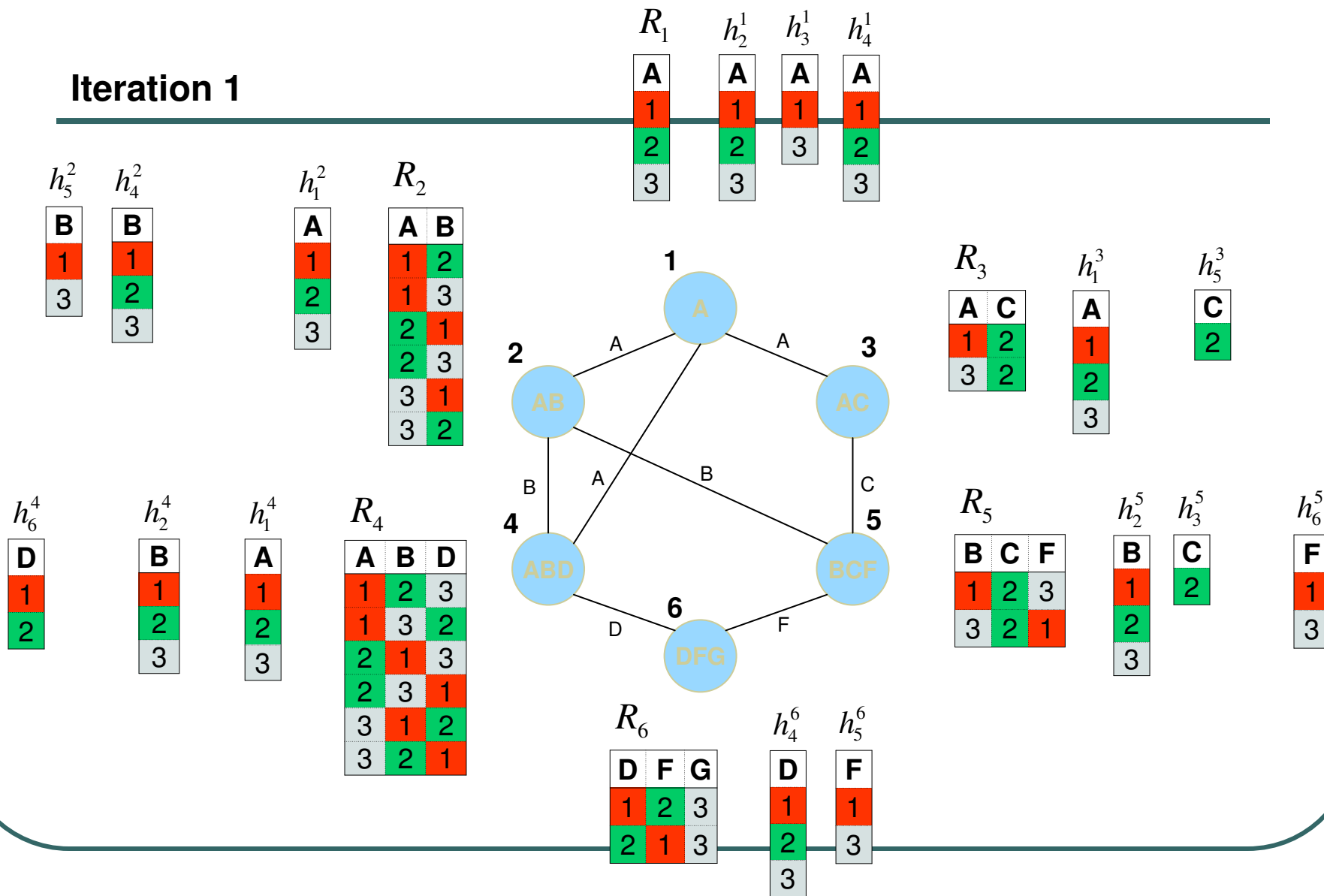
$$R_i \leftarrow R_i \cap \left(\bowtie_{k \in ne(i)} h_k^i \right)$$

Iteration 1

R_1

A
1
3

R_2

A	B
1	3
2	1
2	3
3	1

R_3

A	C
1	2
3	2

R_4

A	B	D
1	3	2
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

R_6

D	F	G
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \qquad (1)$$

Iteration 2

R_1

A
1
3

h_2^1

A
1
2
3

h_3^1

A
1
3

h_4^1

A
1
2
3

h_5^2

B
1
3

h_4^2

B
1
2
3

h_1^2

A
1
3

R_2

A	B
1	3
2	1
2	3
3	1

R_3

A	C
1	2
3	2

h_1^3

A
1
3

h_5^3

C
2

h_6^4

D
2

h_2^4

B
1
3

h_1^4

A
1
3

R_4

A	B	D
1	3	2
2	3	1
3	1	2
3	2	1

R_5

B	C	F
1	2	3
3	2	1

h_2^5

B
1
3

h_3^5

C
2

h_6^5

F
1

R_6

D	F	G
2	1	3

h_4^6

D
1
2

h_5^6

F
1
3

Graph:

1 — A
2 — AB
3 — AC
4 — ABD
5 — BCF
6 — DFG

Edges: 1–2 (A), 1–3 (A), 2–4 (B), 2–5 (B), 4–1 (A), 3–5 (C), 4–6 (D), 5–6 (F)
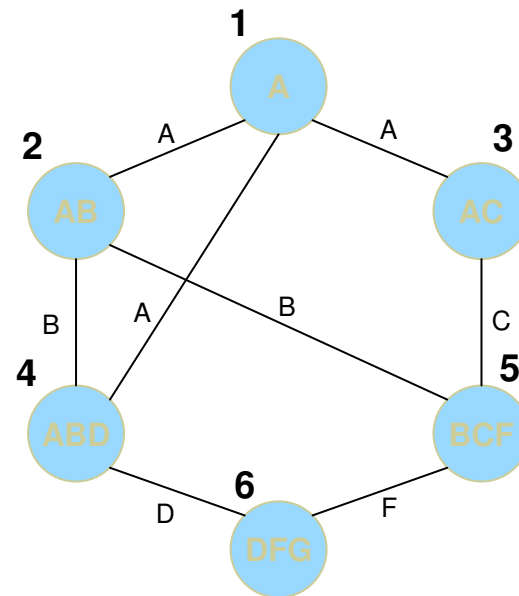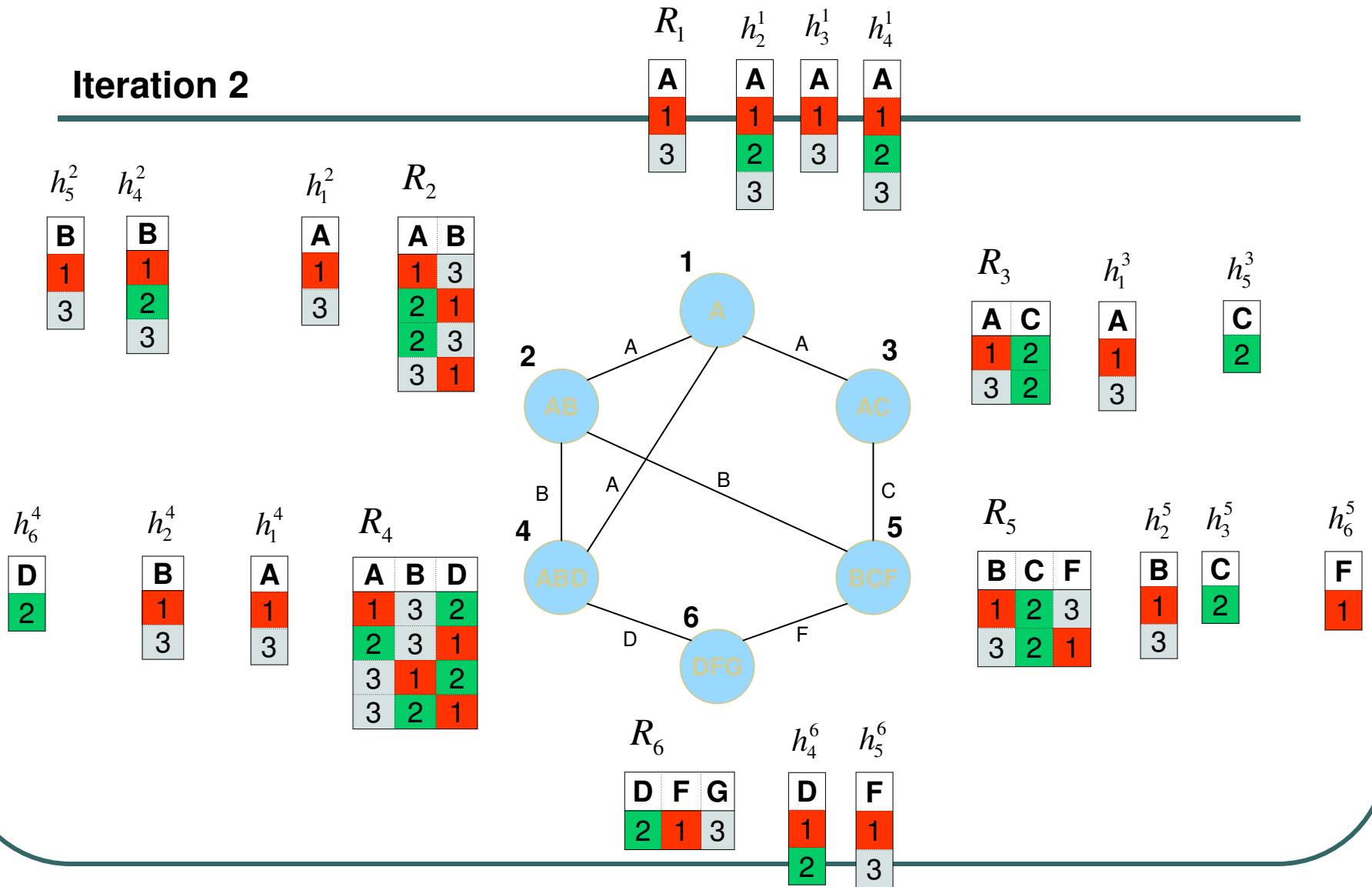
Iteration 2

R_1

A
1
3

R_2

A	B
1	3
3	1

R_3

A	C
1	2
3	2

R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

1 A

2 AB

3 AC

4 ABD

5 BCF

6 DFG

A A A A B B C B D F

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \qquad (1)$$

Iteration 3

R_1

A
1
3

h_2^1

A
1
3

h_3^1

A
1
3

h_4^1

A
1
3

h_5^2

B
3

h_4^2

B
1
3

h_1^2

A
1
3

R_2

A	B
1	3
3	1

R_3

A	C
1	2
3	2

h_1^3

A
1
3

h_5^3

C
2

h_6^4

D
2

h_2^4

B
1
3

h_1^4

A
1
3

R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

h_2^5

B
1
3

h_3^5

C
2

h_6^5

F
1

Graph nodes:
1 — A
2 — AB
3 — AC
4 — ABD
5 — BCF
6 — DFG

Edge labels: A, A, B, A, B, C, D, F

R_6

D	F	G
2	1	3

h_4^6

D
2

h_5^6

F
1
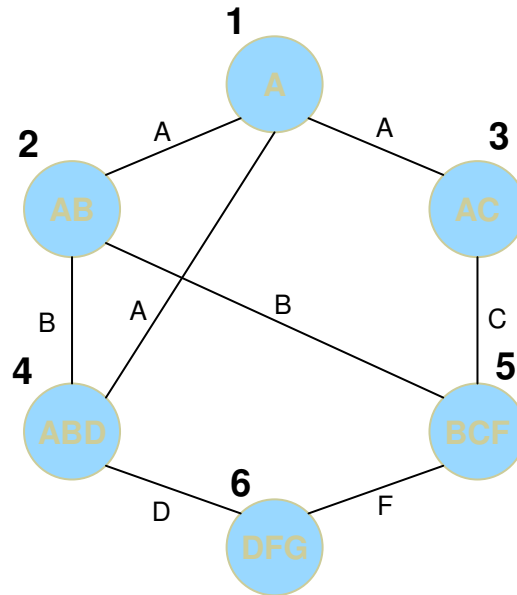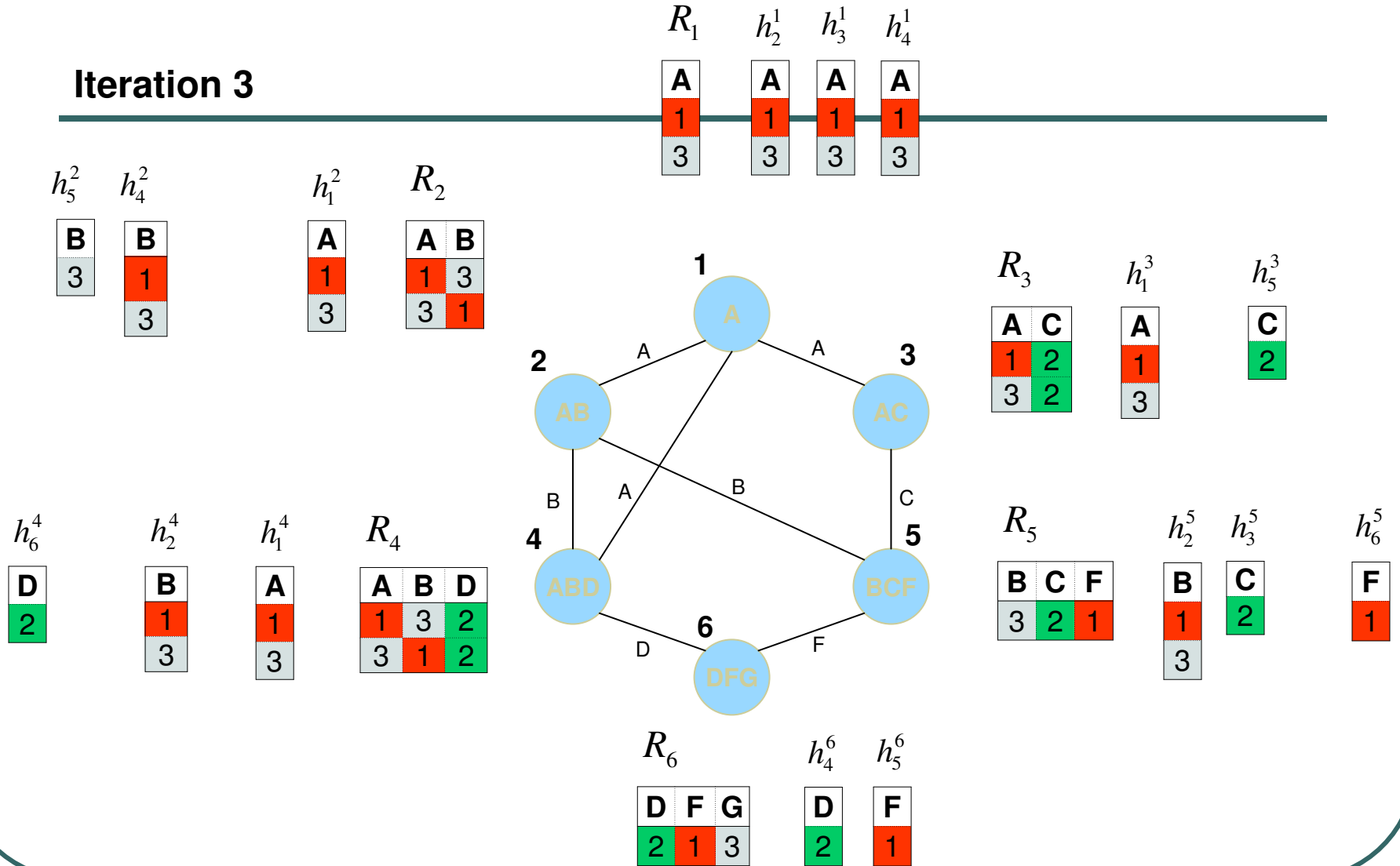
$$R_i \leftarrow R_i \cap \left(\bowtie_{k \in ne(i)} h_k^i \right)$$

Iteration 3

R_1

A
1
3

R_2

A	B
1	3

R_3

A	C
1	2
3	2

R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

$$h_i^j \leftarrow \pi_{l_{ij}}(R_i \bowtie (\bowtie_{k \in ne(i)} h_k^i)) \qquad (1)$$

Iteration 4

R_1

A
1
3

h_2^1

A
1

h_3^1

A
1
3

h_4^1

A
1
3

h_5^2

B
3

h_4^2

B
1
3

h_1^2

A
1
3

R_2

A	B
1	3

R_3

A	C
1	2
3	2

h_1^3

A
1
3

h_5^3

C
2

h_6^4

D
2

h_2^4

B
1
3

h_1^4

A
1
3

R_4

A	B	D
1	3	2
3	1	2

R_5

B	C	F
3	2	1

h_2^5

B
3

h_3^5

C
2

h_6^5

F
1

Graph:
1 A — 2 AB — 3 AC — 4 ABD — 5 BCF — 6 DFG
edges labeled A, A, B, A, B, C, D, F

R_6

D	F	G
2	1	3

h_4^6

D
2

h_5^6

F
1
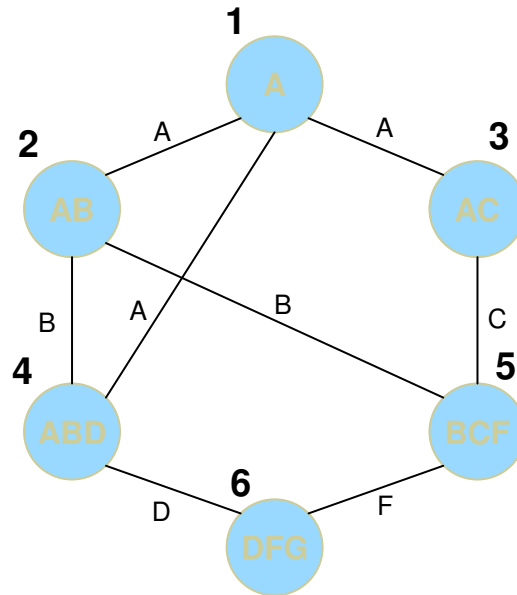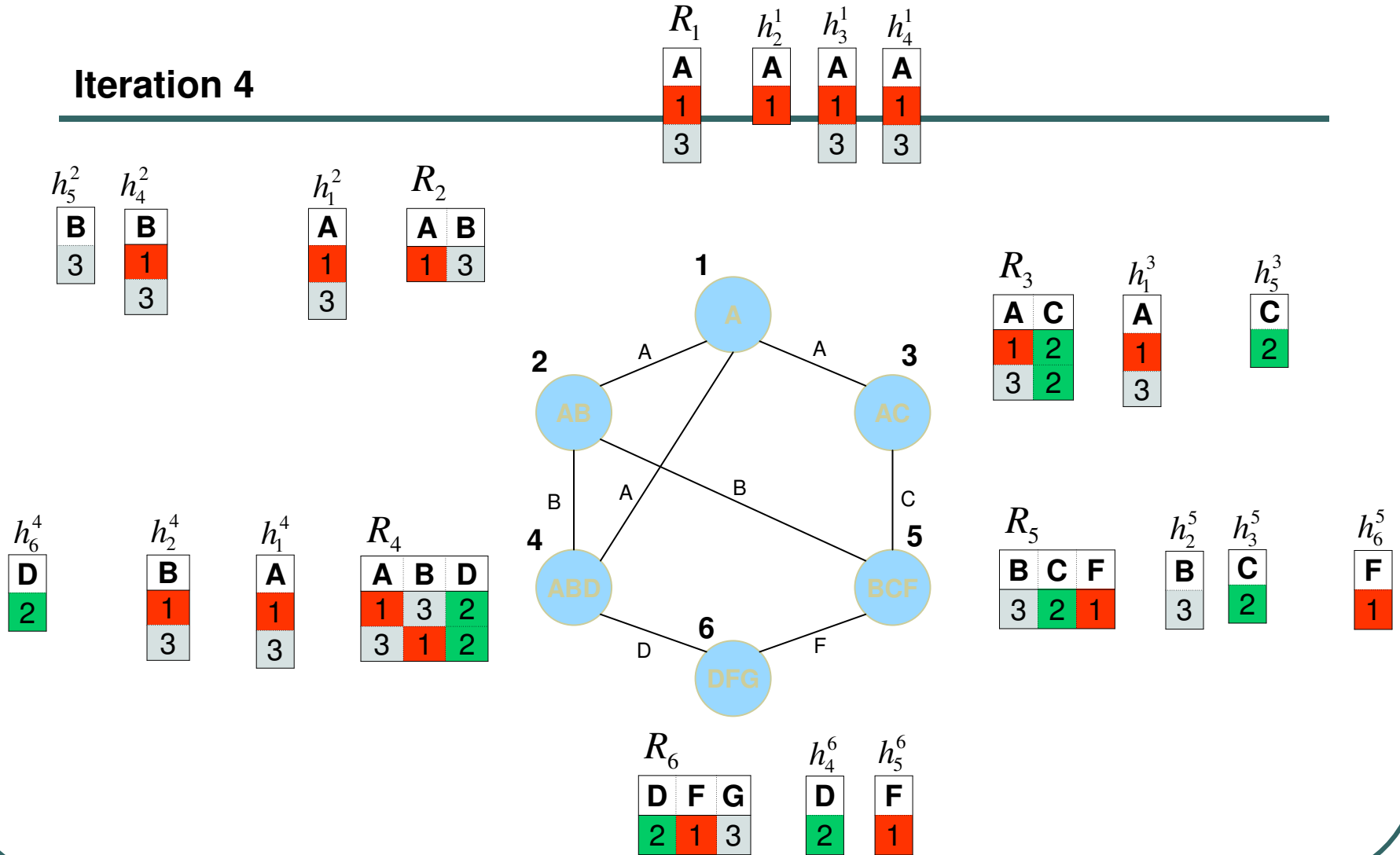
Iteration 4

R_1

A
1

R_2

A	B
1	3

R_3

A	C
1	2
3	2

R_4

A	B	D
1	3	2

R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

Iteration 5

R_1

A
1

h_2^1

A
1

h_3^1

A
1

h_4^1

A
1

h_5^2

B
3

h_4^2

B
3

h_1^2

A
1

R_2

A	B
1	3

R_3

A	C
1	2
3	2

h_1^3

A
1

h_5^3

C
2

h_6^4

D
2

h_2^4

B
3

h_1^4

A
1

R_4

A	B	D
1	3	2

R_5

B	C	F
3	2	1

h_2^5

B
3

h_3^5

C
2

h_6^5

F
1

R_6

D	F	G
2	1	3

h_4^6

D
2

h_5^6

F
1

Graph nodes:
1 A
2 AB
3 AC
4 ABD
5 BCF
6 DFG

Edges labelled: A, A, B, A, B, C, D, F

Iteration 5

R_1

A
1

R_2

A	B
1	3

R_3

A	C
1	2

R_4

A	B	D
1	3	2

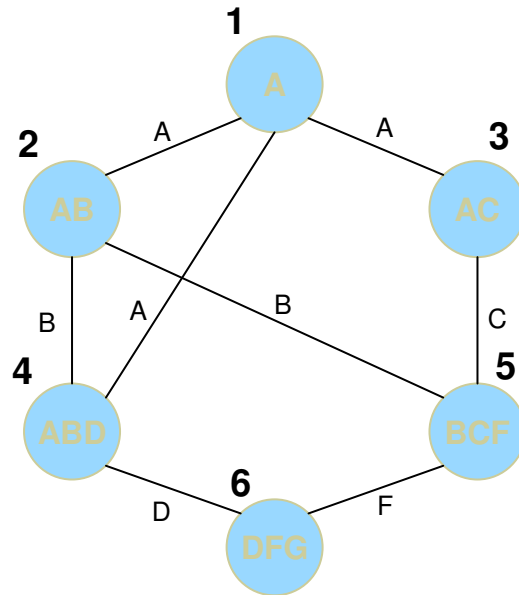R_5

B	C	F
3	2	1

R_6

D	F	G
2	1	3

Cluster Tree Elimination - properties

- Correctness and completeness: Algorithm CTE is correct, i.e. it computes the exact joint probability of every single variable and the evidence.

- Time complexity: $O (deg \times (n+N) \times d^{w^*+1})$

- Space complexity: $O (N \times d^{sep})$

  where deg = the maximum degree of a node

  n = number of variables (= number of CPTs)

  N = number of nodes in the tree decomposition

  d = the maximum domain size of a variable

  w^* = the induced width

  sep = the separator size

Time and space by hyperwidth: $O(Nt^{2hw})$,   time O(N t^tw) space

Join-tree clustering is a restricted tree-decomposition

R_{12}, R_{13}, R_{14}
R_{23}, R_{24}, R_{34}

(a)

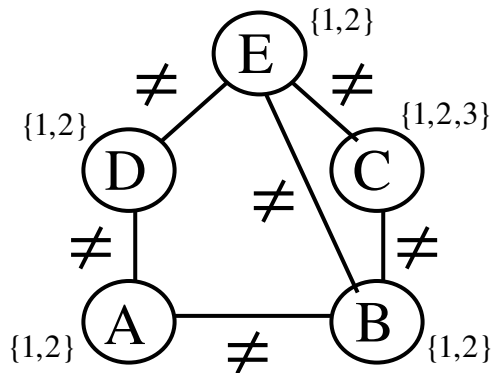R_{12}, R_{34}

R_{13} R_{14} R_{23} R_{24}
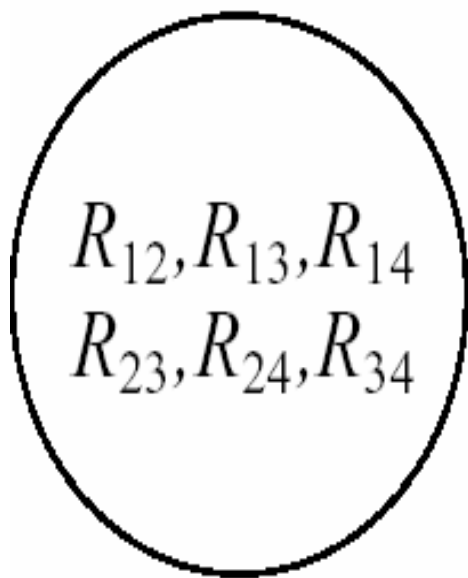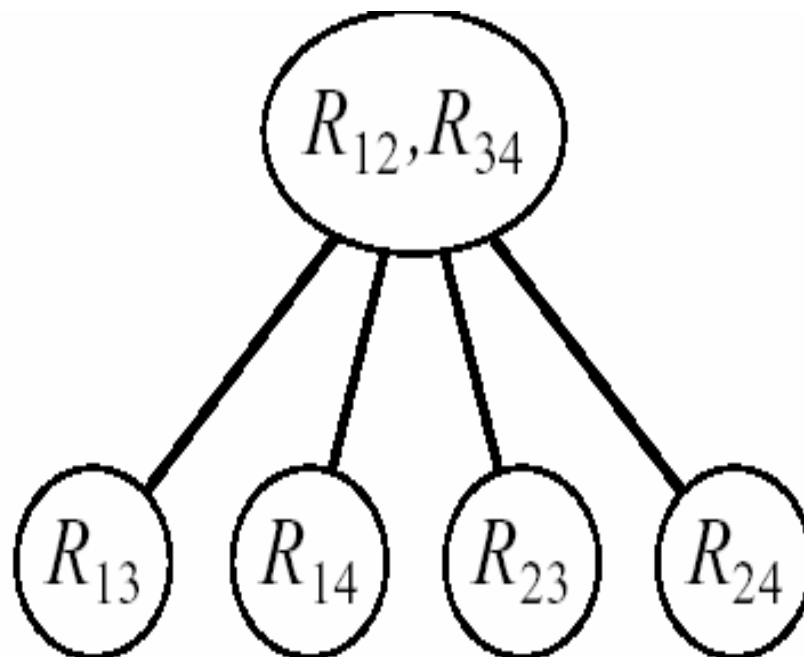
(b)

Adaptive-consistency as tree-decomposition

- Adaptive consistency is a message-passing along a bucket-tree

- **Bucket trees**: each bucket is a node and it is connected to a bucket to which its message is sent.
  - The variables are the clicue of the triangulated graph
  - The funcions are those placed in the initial partition

Bucket Elimination

Adaptive Consistency (Dechter and Pearl, 1987)

$Bucket(E): \ \mathrm{E} \neq \mathrm{D}, \ \mathrm{E} \neq \mathrm{C}, \ \mathrm{E} \neq \mathrm{B}$

$Bucket(D): \ \mathrm{D} \neq \mathrm{A} \ \| \ R_{DCB}$

$Bucket(C): \ \mathrm{C} \neq \mathrm{B} \ \| \ R_{ACB}$

$Bucket(B): \ \mathrm{B} \neq \mathrm{A} \ \| \ R_{AB}$

$Bucket(A): \qquad R_A$

$Bucket(A): \ \mathrm{A} \neq \mathrm{D}, \ \mathrm{A} \neq \mathrm{B}$

$Bucket(D): \ \mathrm{D} \neq \mathrm{E} \ \| \ R_{DB}$

$Bucket(C): \ \mathrm{C} \neq \mathrm{B}, \ \mathrm{C} \neq \mathrm{E}$

$Bucket(B): \ \mathrm{B} \neq \mathrm{E} \ \| \ R^{D}_{BE}, \ R^{C}_{BE}$
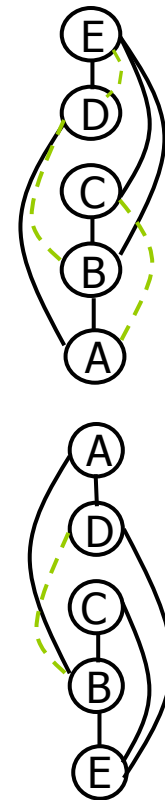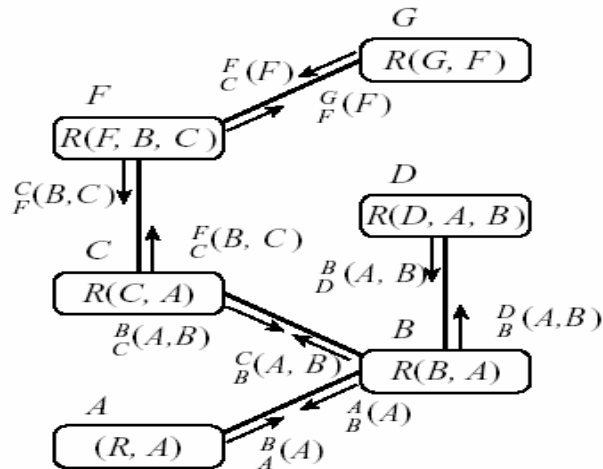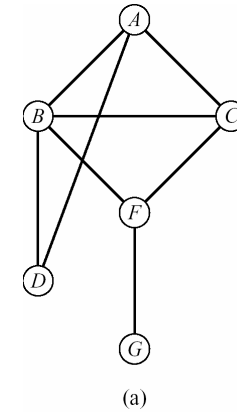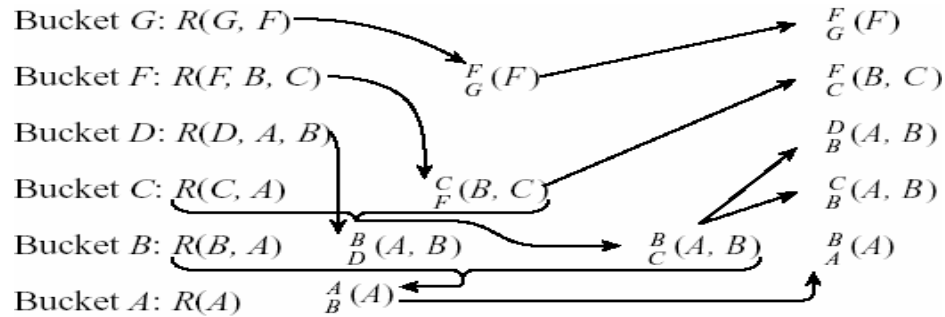
$Bucket(E): \ \| \ R_E$

Complexity : $O(n \exp(w^{*}(d)))$,
$w^{*}(d)$ - *induced width along ordering d*

From bucket-elimination to bucket-tree propagation

(a)

Bucket G: $R(G,F)$ $_G^F(F)$

Bucket F: $R(F,B,C)$

Bucket D: $R(D,A,B)$

Bucket C: $R(C,A)$ $_F^C(B,C)$

Bucket B: $R(B,A)$ $_D^B(A,B)$ $_C^B(A,B)$

Bucket A: $R(A)$ $_B^A(A)$

(a)

G $R(G,F)$

$_G^F(F)$

F $R(F,B,C)$

$_F^C(B,C)$

D $R(D,A,B)$

C $R(C,A)$

$_D^B(A,B)$

$_C^B(A,B)$ B $R(B,A)$

A (R,A)

$_B^A(A)$

(b)

The bottom up messages

Adaptive-consistency as tree-decomposition

- Adaptive consistency is a message-passing along a bucket-tree

- **Bucket trees**: each bucket is a node and it is connected to a bucket to which its message is sent.

- **Theorem**: A bucket-tree is a tree-decomposition
- Therefore, CTE adds a bottom-up message passing to bucket-elimination.