Problem Solving and Search in Artificial Intelligence

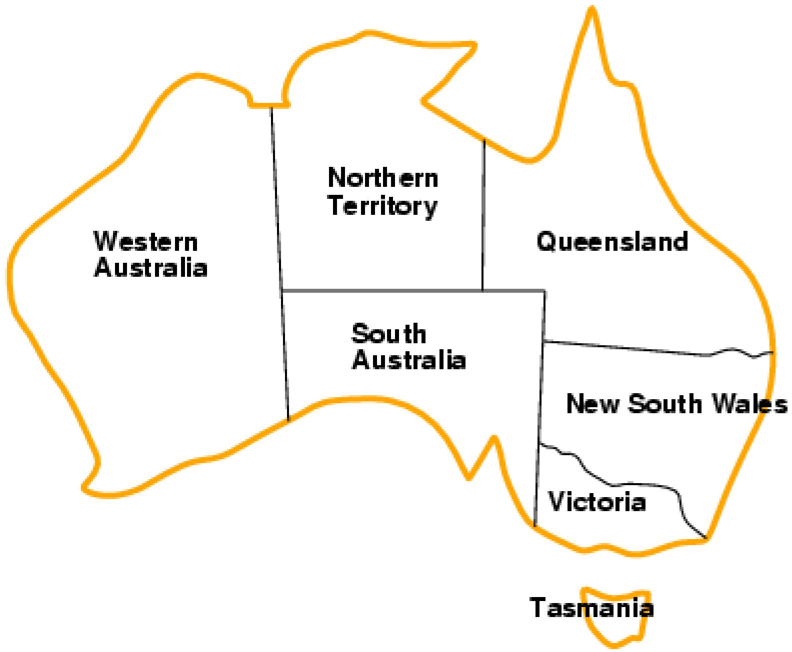
Tree and Hypertree Decompositions

Nysret Musliu
Database and Artificial Intelligence Group
Institut für Informationssysteme, TU-Wien

Introduction

- Many instances of constraint satisfaction problems can be solved in polynomial time if their treewidth (or hypertree width) is small
- Solving of problems with bounded width includes two phases:
  - Generate a (hyper)tree decomposition with small width
  - Solve a problem (based on generated decomposition) with a particular algorithm such as for example dynamic programming
- The efficiency of solving of problem based on its (hyper)tree decomposition depends from the width of (hyper)tree decomposition
- It is of high importance to generate (hyper)tree decompositions with small width

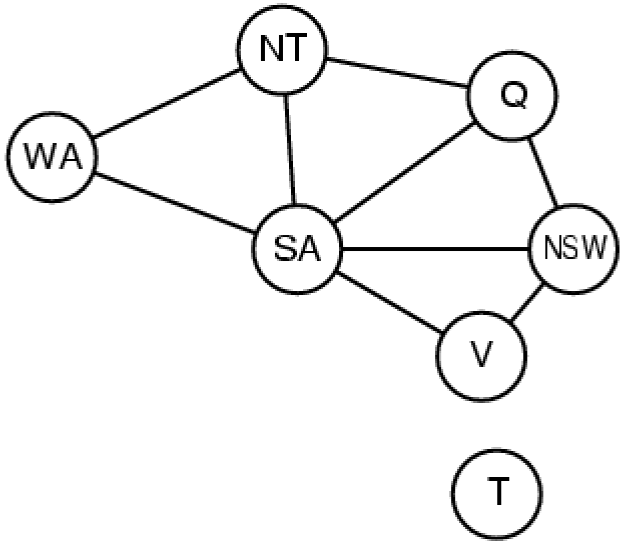
Constraint Satisfaction Problems: Map-Coloring



- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors
- e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}□

Constraint graph

- Constraint graph: vertices of nodes are variables, edges are constraints
- Binary constraints



Constraint Satisfaction Problems: SAT Problem

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \overline{x_4} \vee x_5 \vee x_6) \wedge \ldots \wedge (x_3 \vee x_4 \vee x_7 \vee x_8) \ldots$$

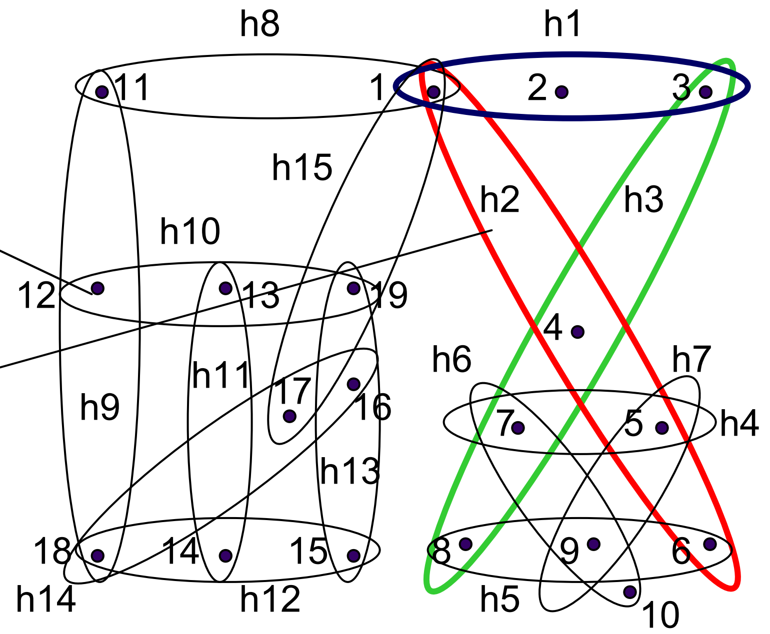
Possible CSP formulation:

- Variables: $x_1, x_2, x_3 \cdots$
- Domain of variables: 0,1
- Constraints:
- C1: $(x_1 \vee x_2 \vee \bar{x}_3)$ → true
- C2: $(x_1 \vee \overline{x_4} \vee x_5 \vee x_6)$ → true
- …

CSP and Hypergraph

Vertices of hypergraph represents variables

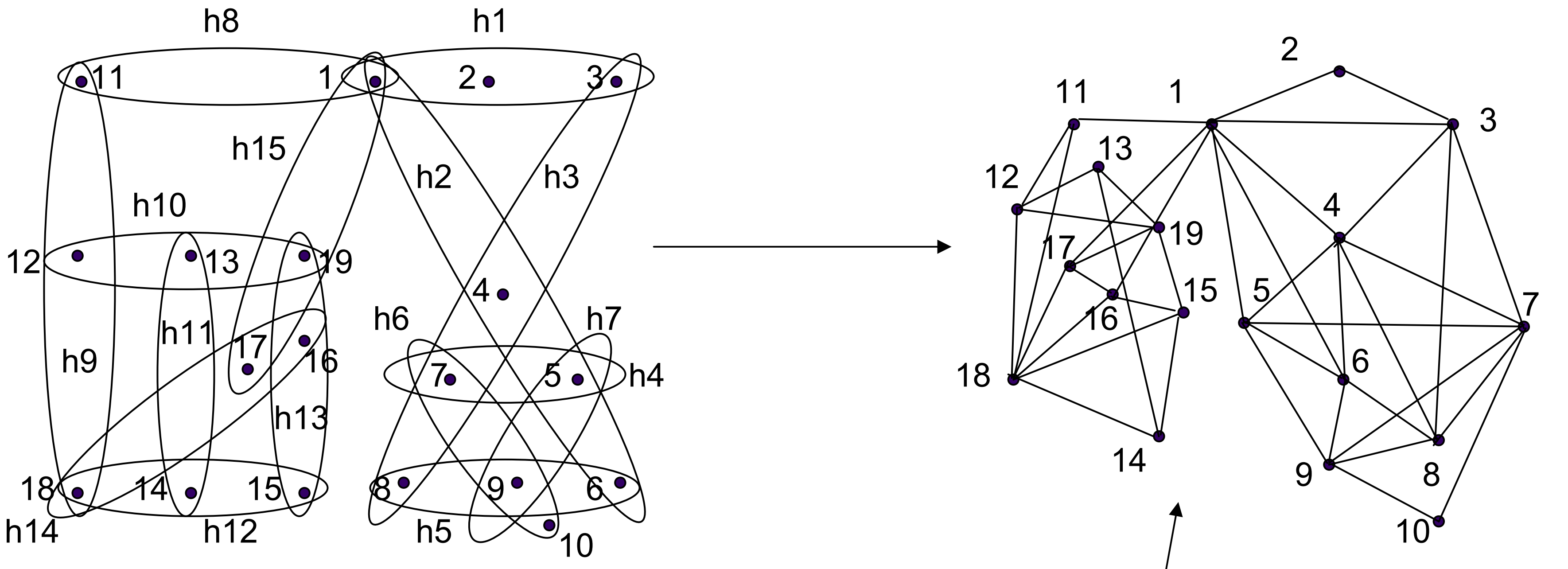
Hyperedges represent the scope of constraints



(X1 or X2 or NOT X3) AND (X1 or NOT X4 or X5 or X6) AND (X3 or X4 or X7 or X8) …

In general worst case complexity: **$2^{NumberOfVariables} = 2^{19}$**

Hypergraph and its primal graph



Primal Graph

CSP and (hyper)tree width

In general exponential worst case complexity

Can we solve this instance more efficiently (or in polynomial time)?

Yes, if it has a small hyper (tree) width!!!

Tree Decomposition

Definition 1. *(see [16], [12]) Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair (T, χ), where $T = (I, F)$ is a tree with node set I and edge set F, and $\chi = \{\chi_i : i \in I\}$ is a family of subsets of V, one for each node of T, such that*

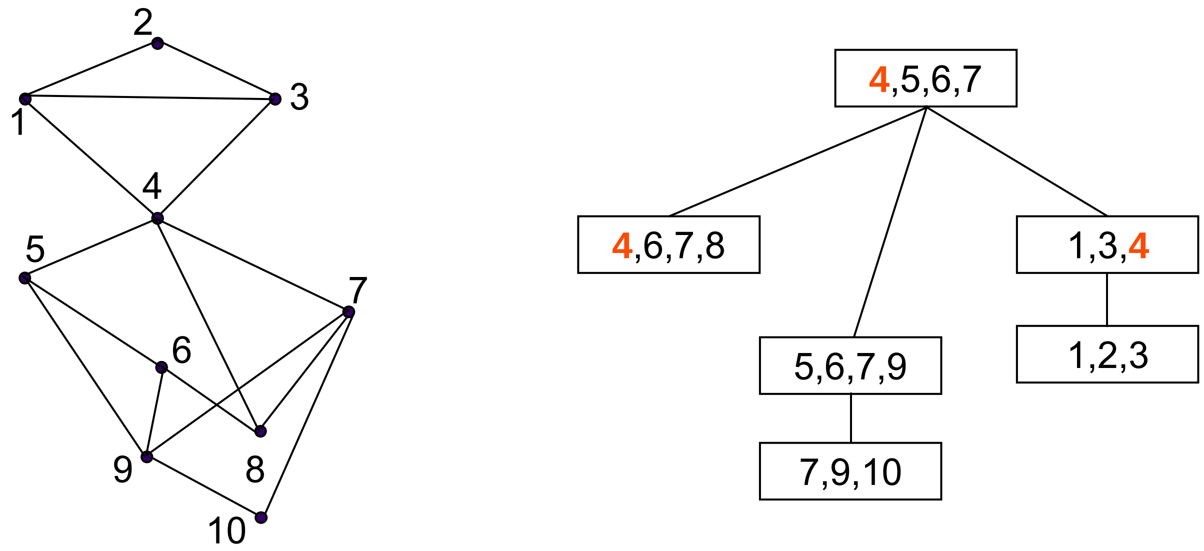
1. $\bigcup_{i \in I} \chi_i = V$,
2. *for every edge $(v, w) \in E$, there is an $i \in I$ with $v \in \chi_i$ and $w \in \chi_i$, and*
3. *for all $i, j, k \in I$, if j is on the path from i to k in T, then $\chi_i \cap \chi_k \subseteq \chi_j$.*

The width of a tree decomposition is $max_{i \in I} |\chi_i| - 1$. The treewidth of a graph G, denoted by $tw(G)$, is the minimum width over all possible tree decompositions of G.

16. N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal Algorithms*, 7:309–322, 1986.

12. A. Koster, H. Bodlaender, and S. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics 8, Elsevier Science Publishers*, 2001.

Tree decomposition of a graph



All pairs of vertices that are connected appear in some node of the tree

Connectedness condition for *vertices*

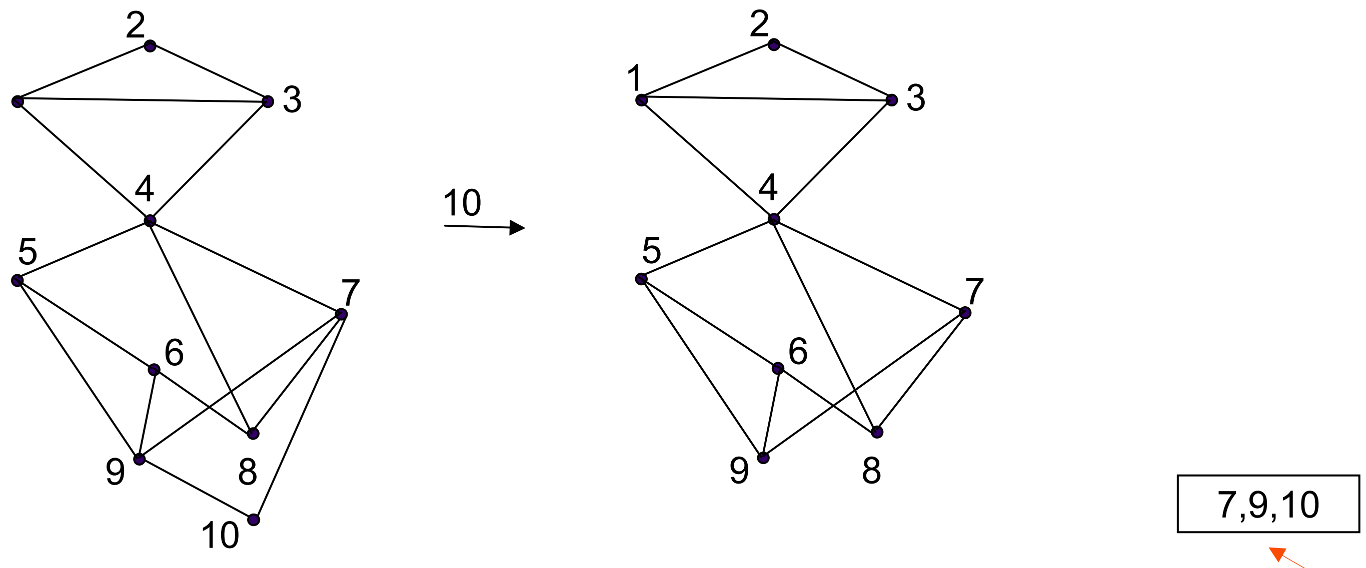
Problem

- For the given problem find the tree decomposition with minimal width -> NP hard
- There exist perfect elimination ordering which produces tree decomposition with treewidth (smallest width)
- Tree decomposition problem → search for the best elimination ordering of vertices!
- Permutation Problem -> similar to TSP

Possible elimination ordering for the graph in the previous slide:

10, 9, 8, 7, 2, 3, 6, 1, 5, 4

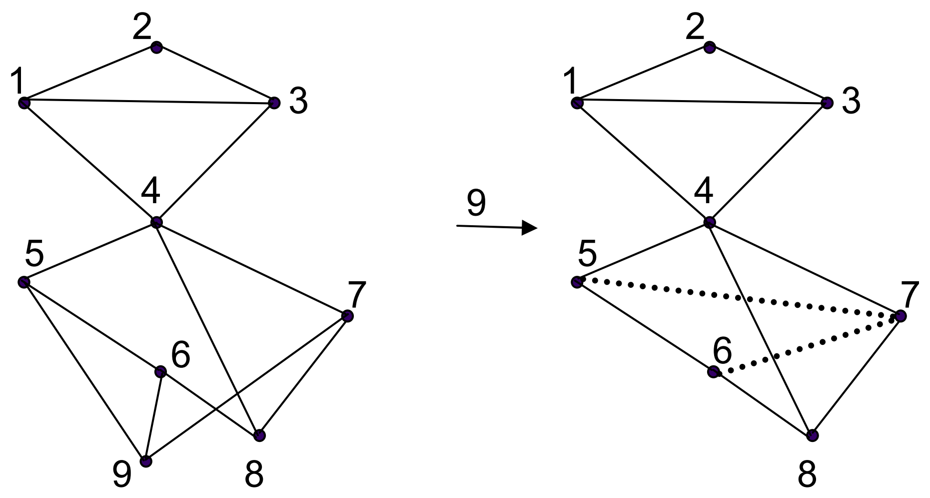
Perfect Elimination Ordering



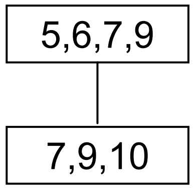
Vertex 10 is eliminated from the graph. All neighbors of 10 are connected and a tree node is created that contains vertex 10 and its neighbors

Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

Perfect Elimination Ordering



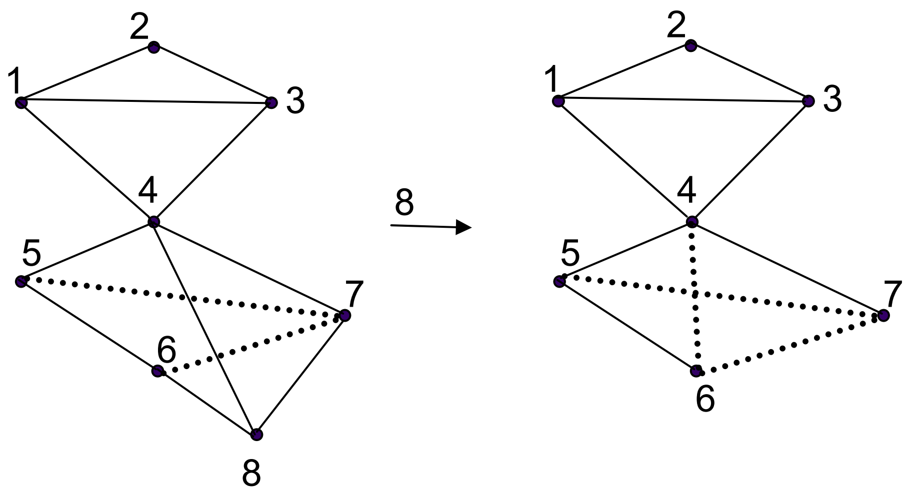
The tree decomposition node with vertices [7,9,10] is connected with the tree decomposition node which is created when the next vertex which appears in [7,9,10] is eliminated (in this case vertex 9)

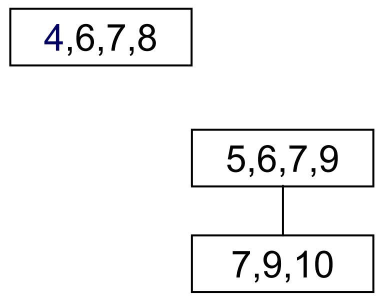


Vertex 9 is eliminated from the graph. All neighbors of vertex 9 are connected and a new tree node is created

Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

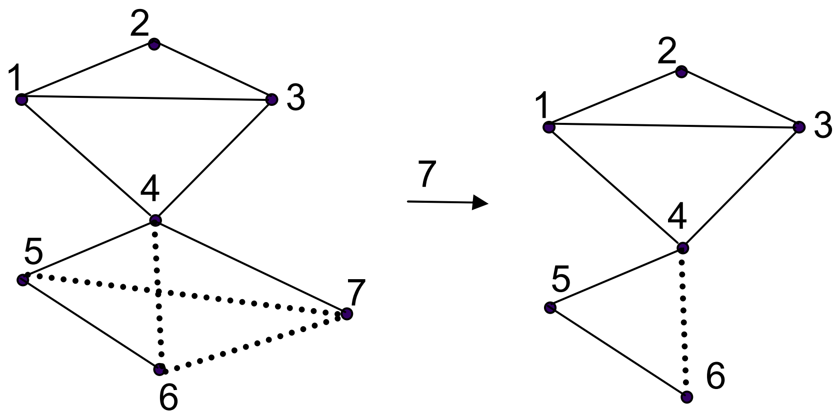
Perfect Elimination Ordering

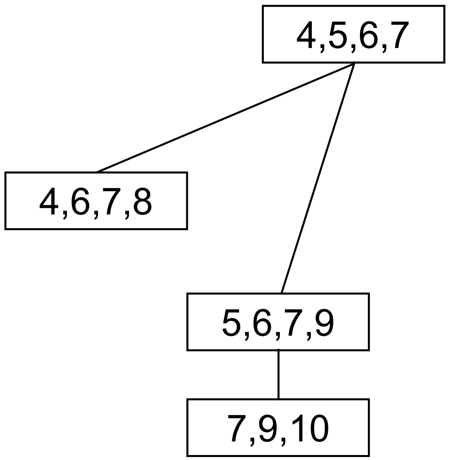




Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

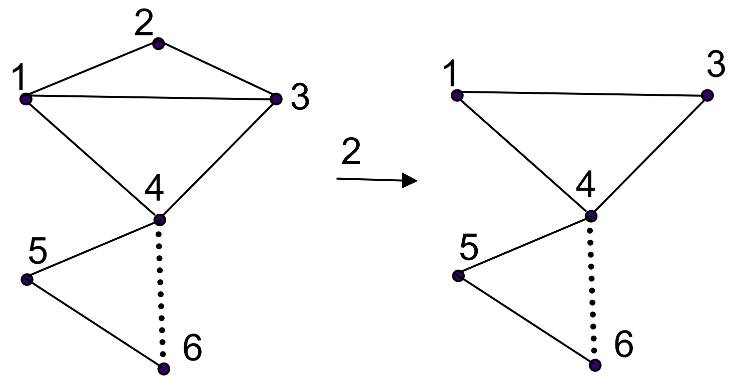
Perfect Elimination Ordering

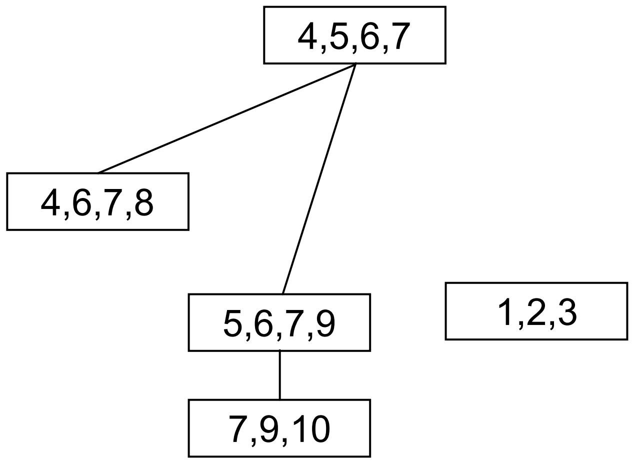




Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

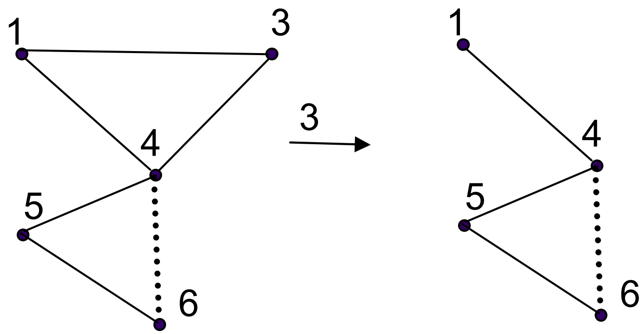
Perfect Elimination Ordering

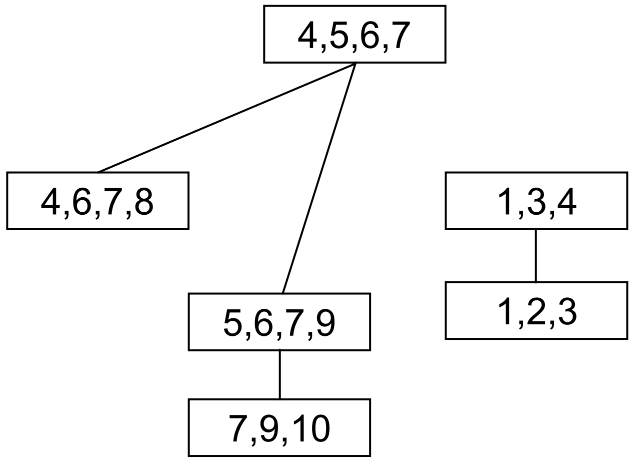




Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

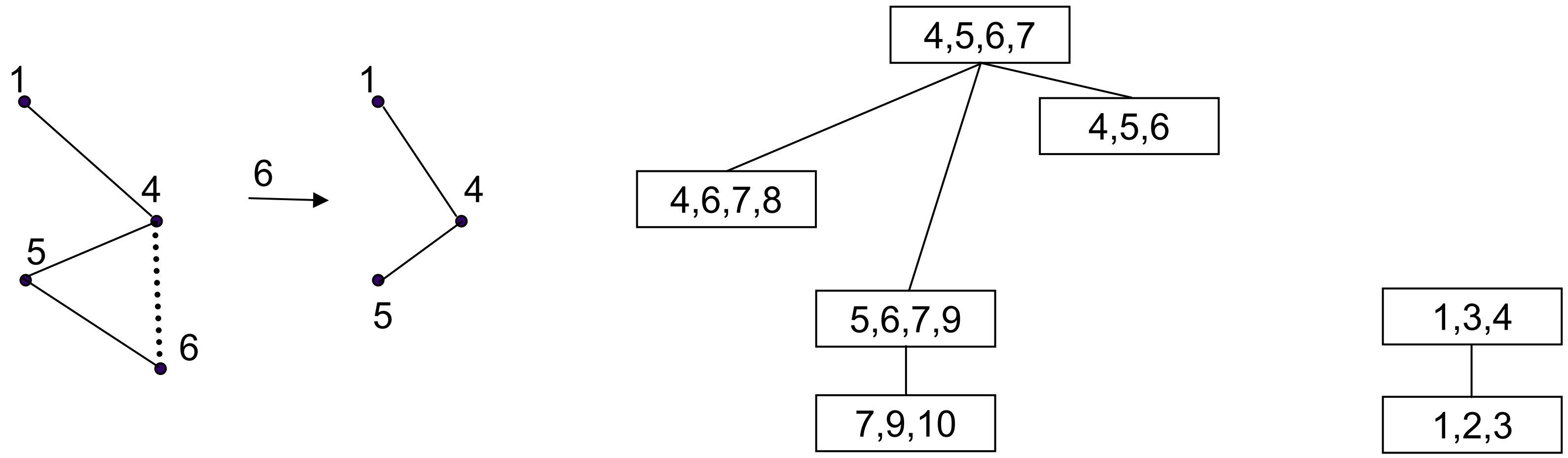
Perfect Elimination Ordering





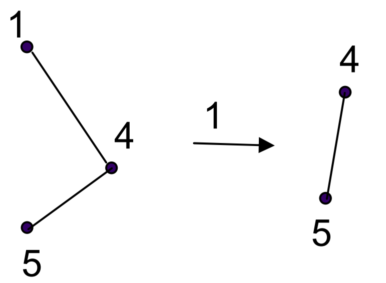
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

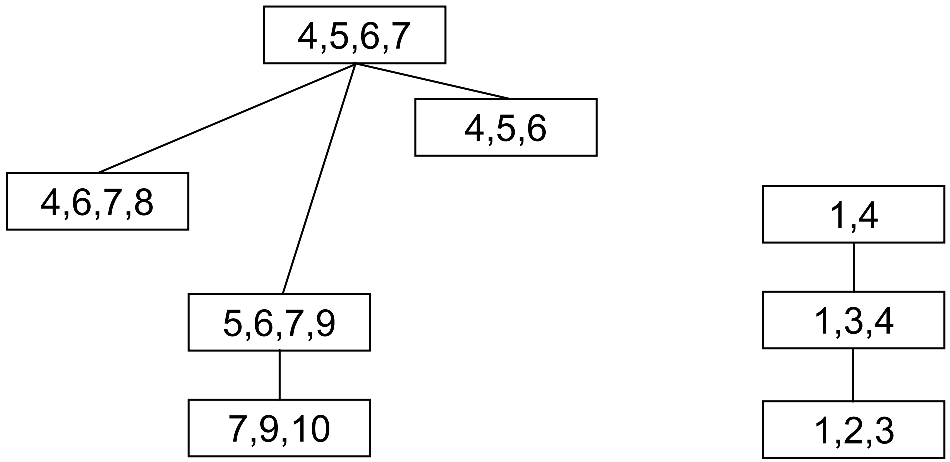
Perfect Elimination Ordering



Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

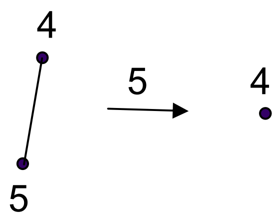
Perfect Elimination Ordering

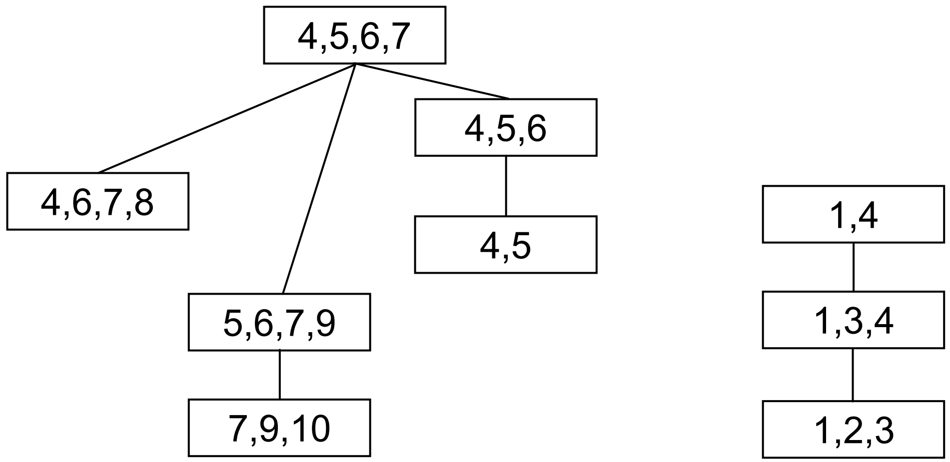




Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

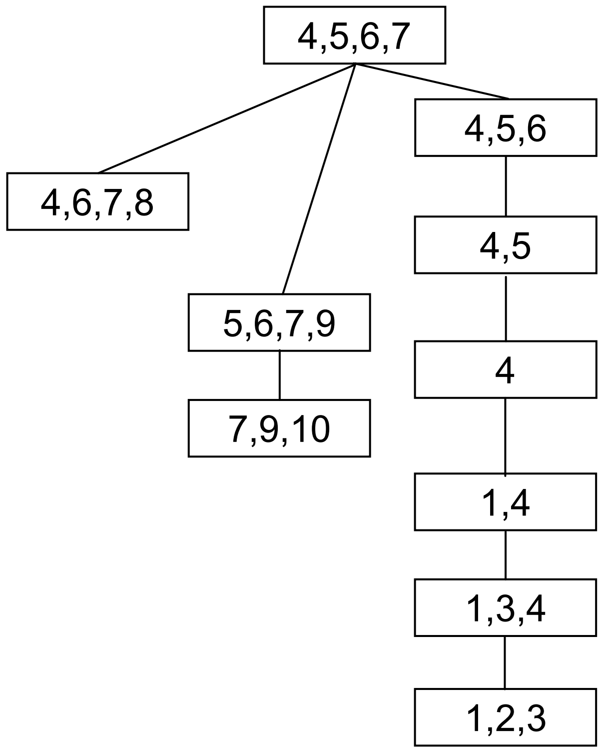
Perfect Elimination Ordering





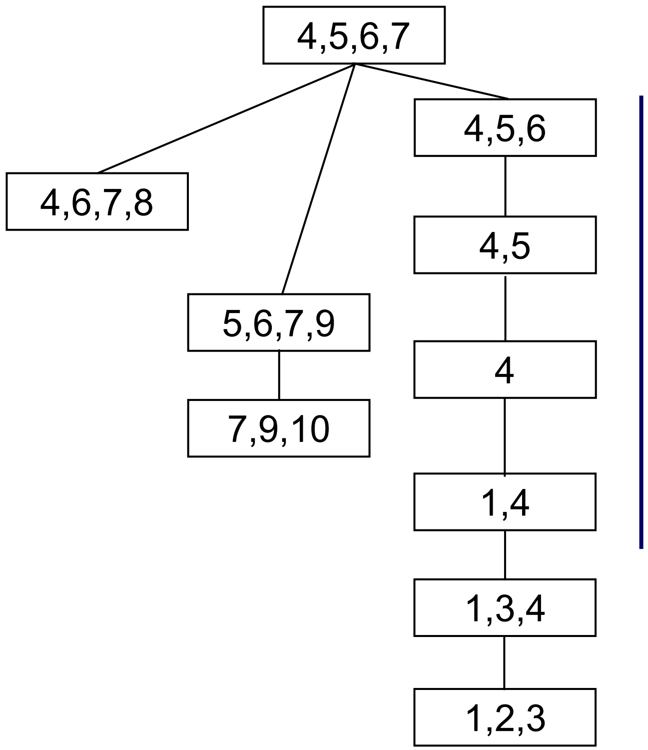
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

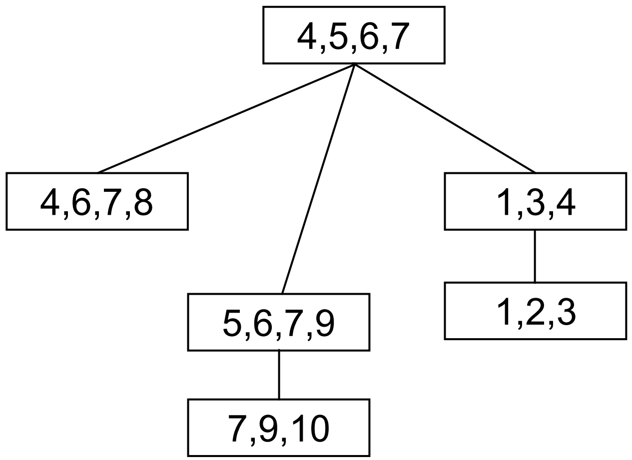
Perfect Elimination Ordering



Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

Perfect Elimination Ordering

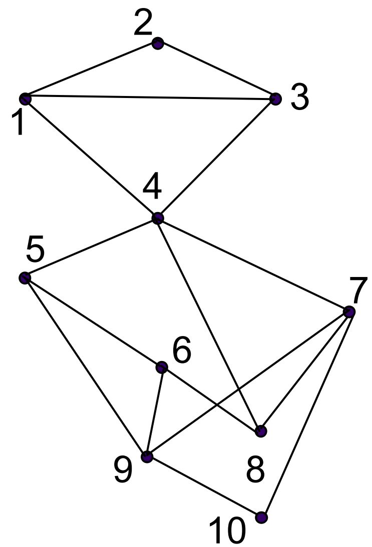


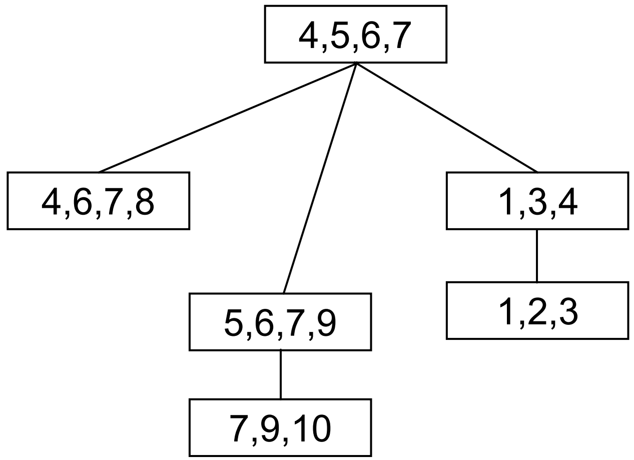


These tree nodes can be removed, because they are contained in other tree nodes

Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

Tree decomposition of a graph





Width: *Max(vertices in tree node) -1 = 3*

Treewidth: minimal width over all possible tree decompositions

Solving of problem based on tree decomposition

If CSP instance has tree decomposition with treewidth ***k***

The problem can be solved in *$O(n*d^{k+1})$ time*

n – number of variables
d – maximum domain size of any variable in CSP

See *Artificial Intelligence: A Modern Approach* (Russell and Norvig), 2003
Chapter 5, Section 5.4

Algorithms for finding good elimination ordering of vertices

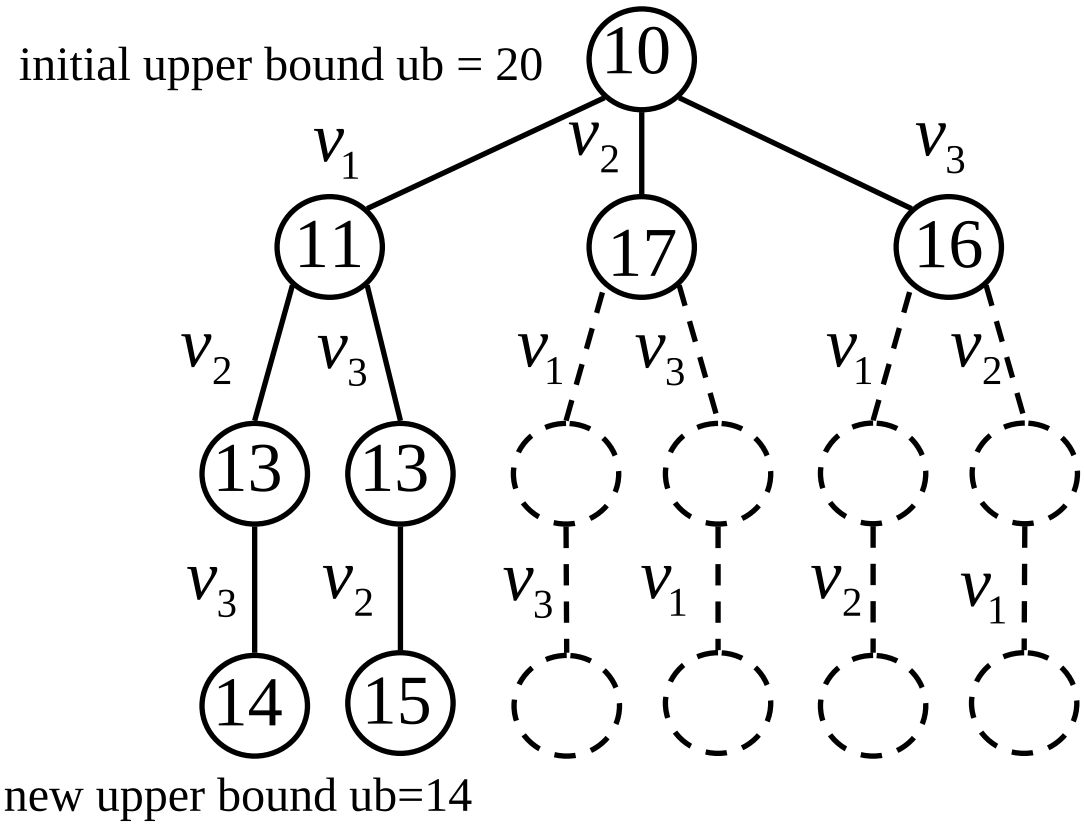
- Exact Methods
  - Branch and bound algorithms
  - A* algorithm

- (Meta) Heuristic methods
  - Maximum Cardinality Search (MCS)
  - Min-Fill heuristics
  - Genetic Algorithms
  - Tabu Search
  - Iterated Local Search

For a detailed description of these algorithms see:
Hans L. Bodlaender, Arie M. C. A. Koster: Treewidth computations I. Upper bounds. Inf. Comput. 208(3): 259-275 (2010) -
http://www.sciencedirect.com/science/article/pii/S0890540109000947

T. Hammerl, N. Musliu, W. Schafhauser. Metaheuristic Algorithms and Tree Decomposition. Handbook of Computational Intelligence, to appear.
http://www.dbai.tuwien.ac.at/staff/musliu/TreeDecompChap.pdf

Branch and Bound algorithm

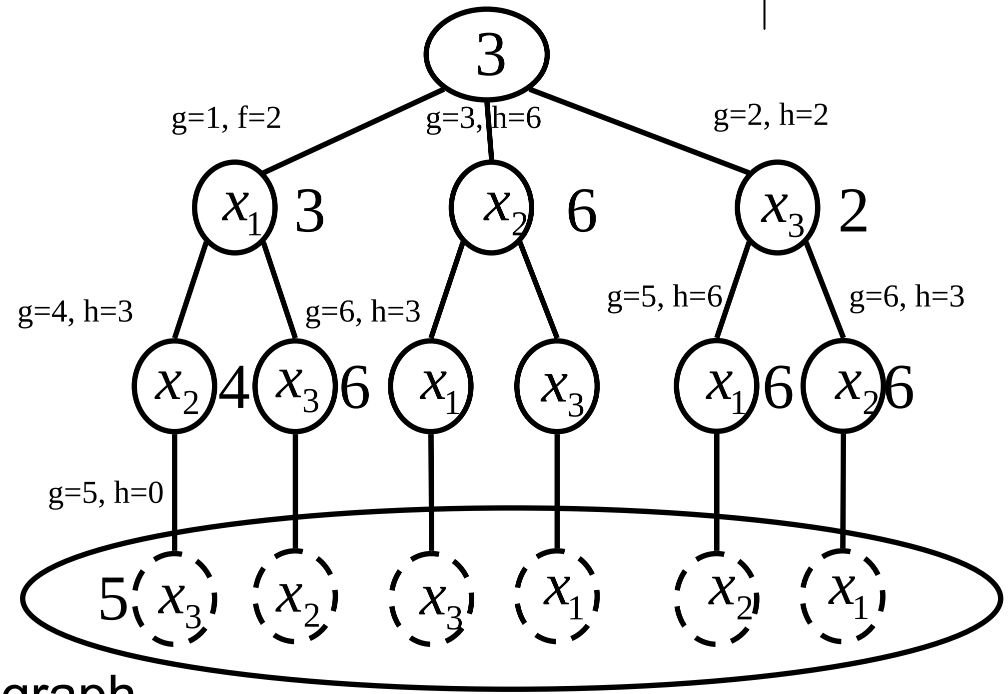


- Different lower bounds for tree width

Werner Schafhauser. *New Heuristic Methods for Tree Decompositions and Generalized Hypertree Decompositions*, Master Thesis, TU Wien, 2006.

A* Algorithm for Treewidth

- f = g + h
- tree = all elimination orderings
- f = max(g, h)
- g = width of partial solution
- h = lower bound for remaining graph
- state with smallest f-value is visited next



Polynomial greedy algorithms

- Maximum Cardinality Search (Tarjan and Yanakakis)
  - Select a random vertex of the graph to be the first vertex in the elimination ordering
  - Pick the next vertex that has the highest connectivity with the vertices previously selected in the elimination ordering (the ties are broken randomly)
  - Repeat step 2 until the whole elimination ordering is constructed

Polynomial greedy algorithms

- Min-Fill heuristics
  - Select the vertex which adds the smallest number of edges when eliminated (the ties are broken randomly) to be the first vertex in the elimination ordering
  - Pick the next vertex that adds the minimum number of edges when eliminated from the graph (the ties are broken randomly)
  - Repeat step 2 until the whole elimination ordering is constructed

  When the vertex is eliminated from the graph all its neighbors are connected (new edges are inserted in the graph)
- Literature and Benchmark Instances for tree decomposition:
  - **TreewidthLIB**

    http://www.cs.uu.nl/~hansb/treewidthlib/index.php

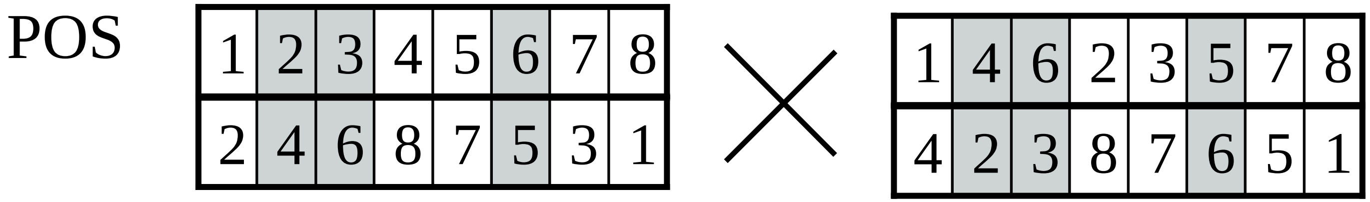
Tabu Search Algorithm

- Moves:
  - Swap to nodes in the elimination ordering

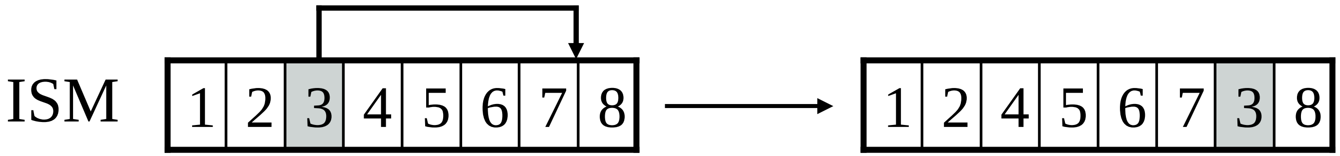
    10, 9, 8, 1, 2, 6,7,…
  - Neighborhood
    - All possible solutions that can be obtained with swap of two vertices
  - Tabu list: moved nodes are made tabu for several iterations
    - Diversification of search
  - Aspiration criteria
  - Use of frequency based memory
    - Intensification or diversification of search

Operators and Control Parameters

- Crossover position-based crossover (POS)



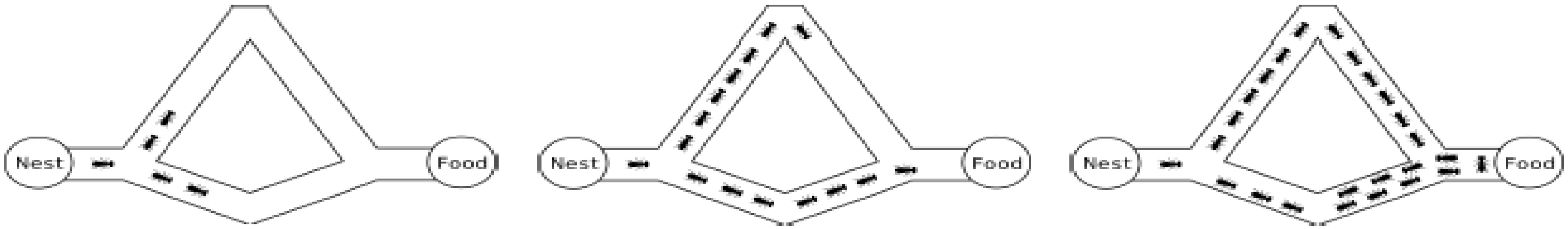
- mutationinsertion mutation (ISM)



- population size , mutation rate, crossover rate, t. s. group size

Ant Colony Optimization

The “Double Bridge Experiment”:

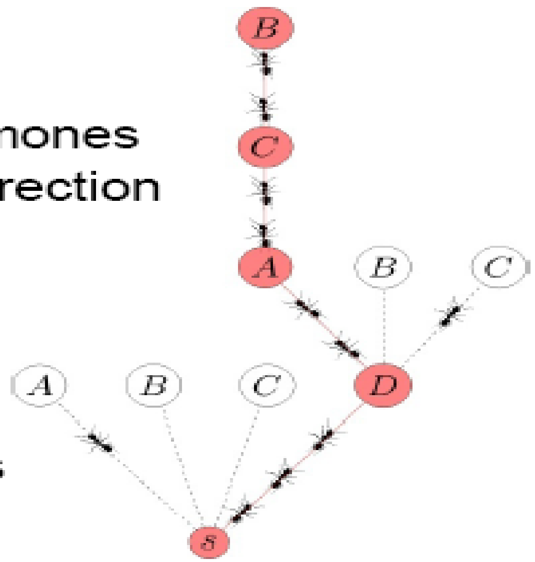


a) Ants start exploration.

b) Ants on shorter branch reach food source.

c) Ants more likely to return on shorter branch.

- Nature-inspired algorithm
- Ants find shortest path by depositing pheromones
- Pheromone ⟶ ants more likely to choose direction



- Adoption of principle for (hyper)tree decomposition
- Artificial ants construct elimination orderings

Thomas Hammerl. *Ant Colony Optimization for Tree and Hypertree Decompositions*. Master Thesis, Vienna University of Technology, 2009.

Iterated Heuristic Algorithm

Algorithm 1 Iterative heuristic algorithm - IHA

Generate initial solution $S1$

$BestSolution = S1$

while Termination Criteria is not fulfilled **do**

  $S2 = ConstructionPhase(S1)$

  if Solution $S2$ fulfils the acceptance criteria **then**

    $S1 = S2$

  else

    $S1 = BestSolution$

  end if

  Apply perturbation in solution $S1$

  Update $BestSolution$ if solution $S2$ has better (or equal) width than the current best solution

end while

RETURN $BestSolution$

Nysret Musliu. An Iterative Heuristic Algorithm for Tree Decomposition. *Studies in Computational Intelligence Springer,Volume 153,pages 133-150, 2008. Carlos Cotta, Jano van Hemert (Eds.).*

Construction Phase

- Simple local search:

while NrNotImprovments < MAXNotImprovments **do**

Select a vertex in the elimination ordering of solution $S2$ which causes the largest clique in the elimination ordering (the ties are broken randomly)

Insert this vertex in the random position in the ordering OR swap the vertex with another vertex in a random chosen position in the ordering

end while

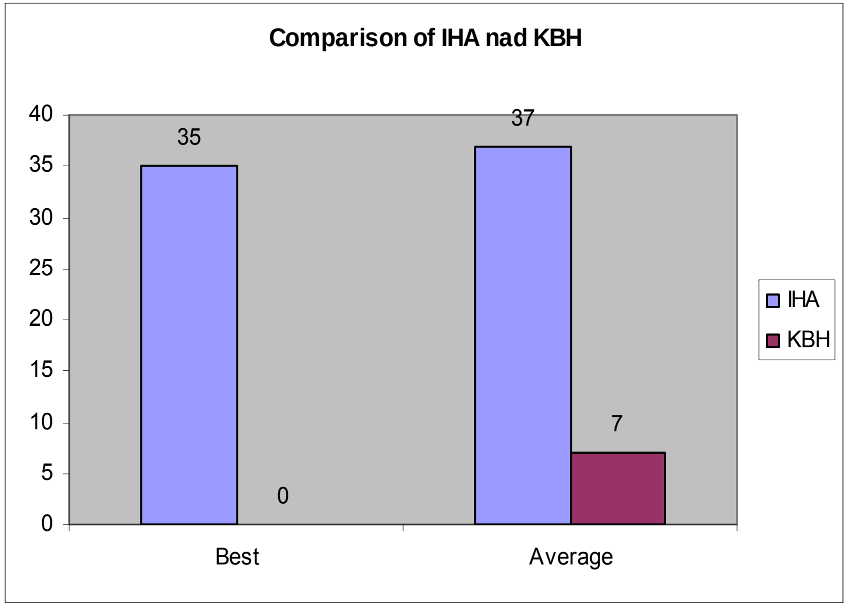
- Other local search techniques
  - Acceptance of solution is based on some probability that depends on adaptive temperature (like simulated annealing)
  - Solution is accepted after the move, if its quality is not so worse (adaptive threshold accepting)

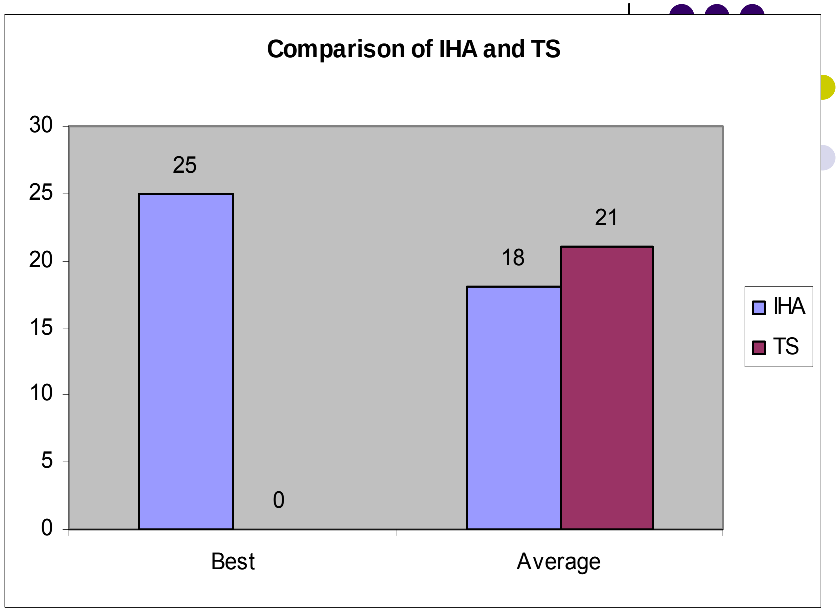
Perturbation Mechanisms

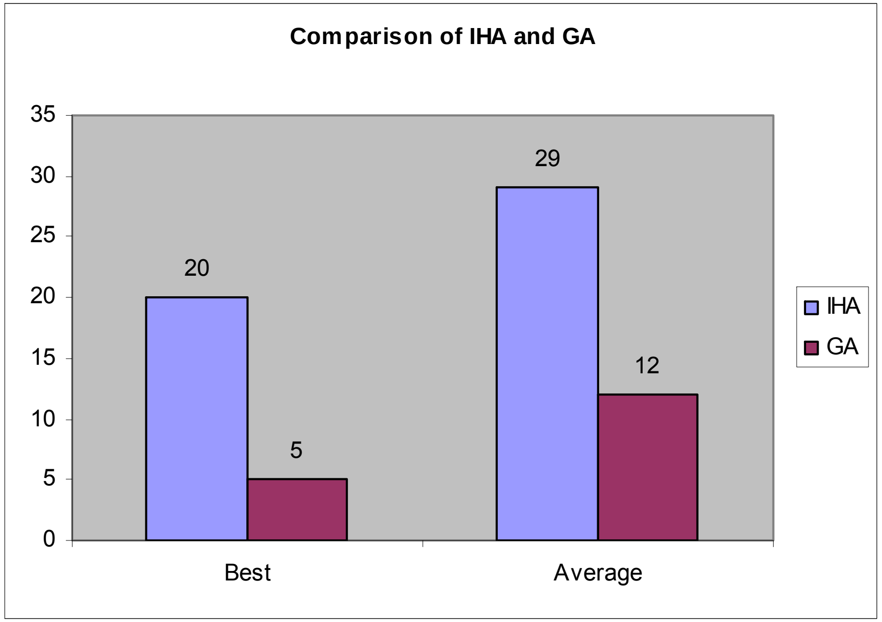
  - RandPert: N vertices are chosen randomly and they are moved in new random position in the ordering
  - MaxCliquePer: All nodes which produce the maximal clique in the elimination ordering are inserted in an new randomly chosen position in the ordering
- Different sizes
- Self-Adaptive perturbation size
  - The number of nodes N varies from 2 to 10
  - Begin with size 2
  - The size of perturbation is increased or decreased based on the feedback during the search

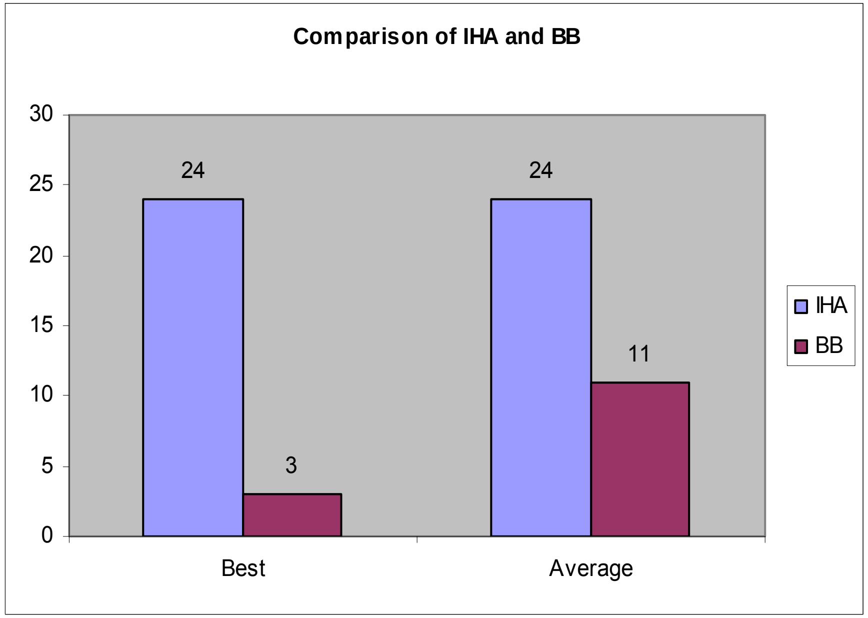
Results

- Dimacs instances for graph coloring
  - Koster, Bodlaender, et al:
    - Treewidth: Computational experiments -> different ordering heuristics
  - Clautiaux et al:
    - Heuristic and Meta-Heurisistic Methods for Computing Graph Treewidth – Tabu Search
  - Gogate and Dechter, Bachore and Bodlaender (branch and bound alg.):
    - A complete anytime algorithm for treewidth – Branch and bound algorithm
  - GA
  - Ant Colony optimization
  - IHA


Comparison of IHA nad KBH
40
35
30
25
20
15
10
5
0
35
0
37
7
Best
Average
IHA
KBH

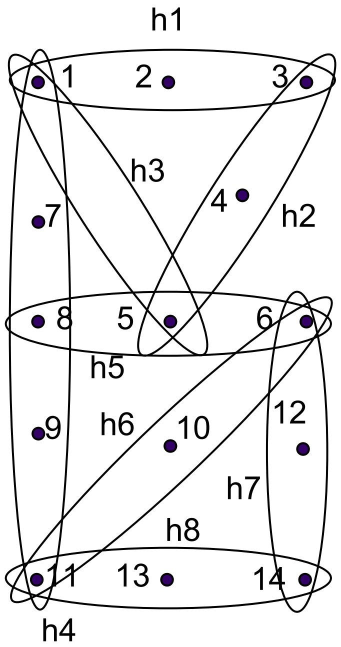

Comparison of IHA and TS
30
25
20
15
10
5
0
25
0
18
21
Best
Average
IHA
TS


Comparison of IHA and GA
35
30
25
20
15
10
5
0
20
5
29
12
Best
Average
IHA
GA

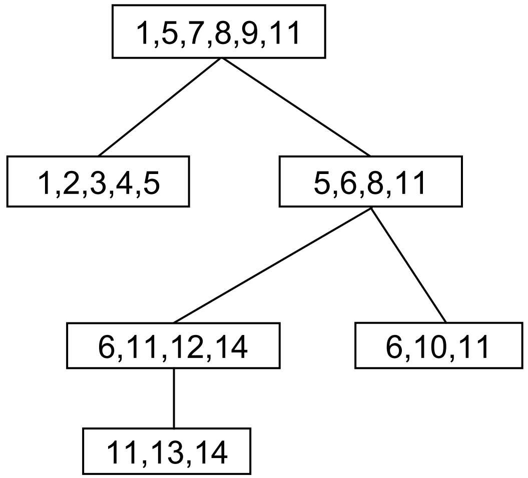

Comparison of IHA and BB
30
25
20
15
10
5
0
24
3
24
11
Best
Average
IHA
BB

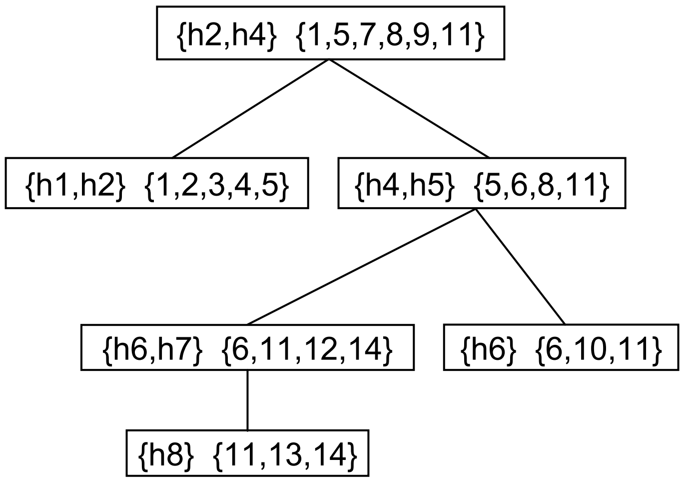
Generalized Hypertree Decomposition

- A generalized hypertree decomposition (GHD) of H is a tree decomposition of H with the following extension.
  - GHD associates additionally to each node of the decomposition tree the set of hyperedges of H
    - The set of vertices associated to each node of the tree must be covered by the set of hyperedges associated to that node
  - The width of a generalized hypertree decomposition is the maximum number of hyperedges associated to a same node of the decomposition



Tree decomposition





Generalized hypertree decomposition

Hypertree decomposition

Definition 1 (Gottlob, Leone, and Scarcello [17]) Let $H = (V(H), E(H))$ be a hypergraph, consisting of a nonempty set $V(H)$ of *vertices*, and a set $E(H)$ of subsets of $V(H)$, the *hyperedges* of H. A *hypertree decomposition* of a hyperg[illegible] H is a hypertree $< T, \chi, \lambda >$ for H which satisfies all the following conditions:

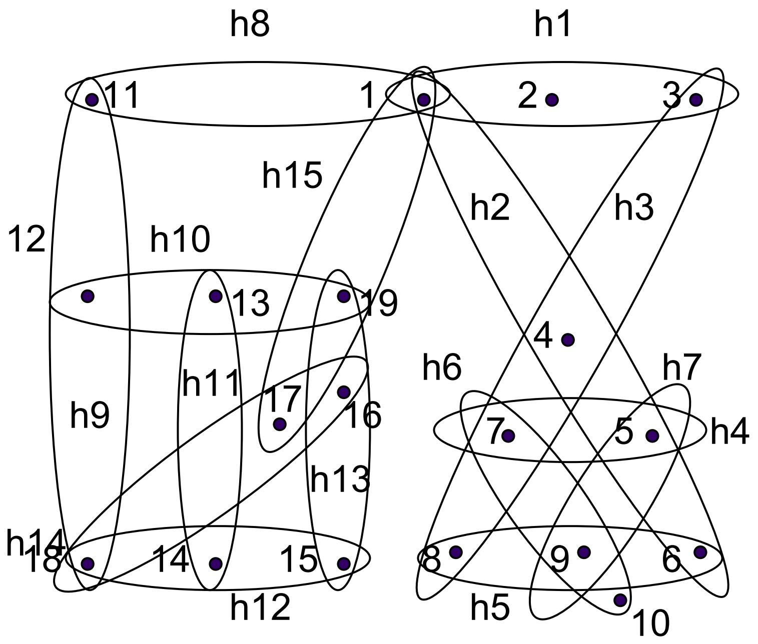
1. for each hyperedge $h \in E(H)$, there exists $p \in vertices(T)$ such that $vertices(h) \subseteq \chi_p$;
2. for each vertex $Y \in V(H)$, the set $\{p \in vertices(T) \mid Y \in \chi_p\}$ induces a (connected) subtree of T;
3. for each vertex $p \in vertices(T)$, $\chi_p \subseteq var(\lambda_p)$;
4. for each vertex $p \in vertices(T)$, $var(\lambda_p) \cap \chi_{T_p} \subseteq \chi_p$

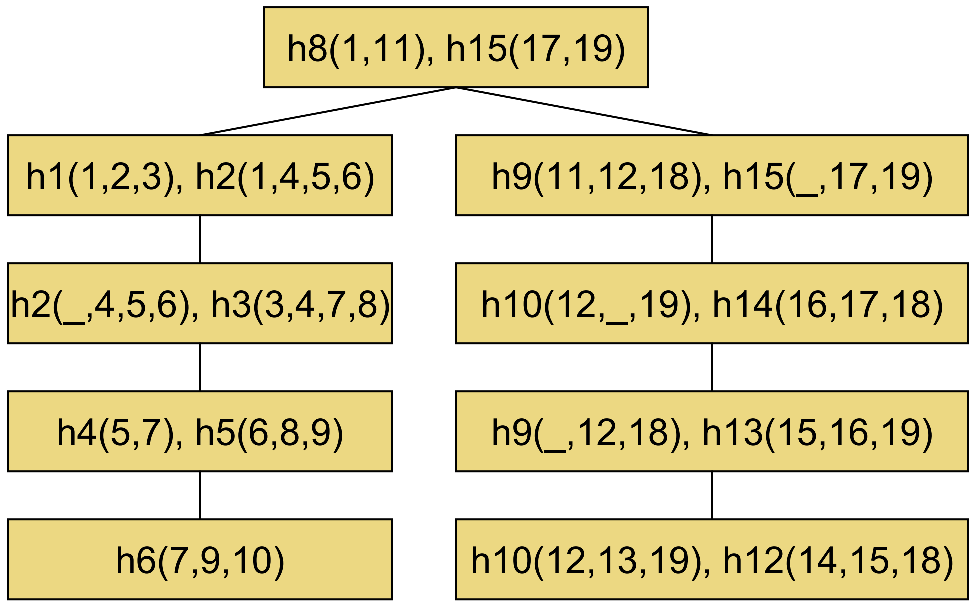
The *width* of the hypertree decomposition $< T, \chi, \lambda >$ is $max_{p \in vertices(T)} |\lambda_p|$. The *hypertree width*, $hw(H)$, of H is the minimum width over all its hypertree decompositions.

17. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decomposition and tractable queries. Journal of Computer and System Sciences 64(3), 579–627 (2002)

Generalized hypertree decomposition

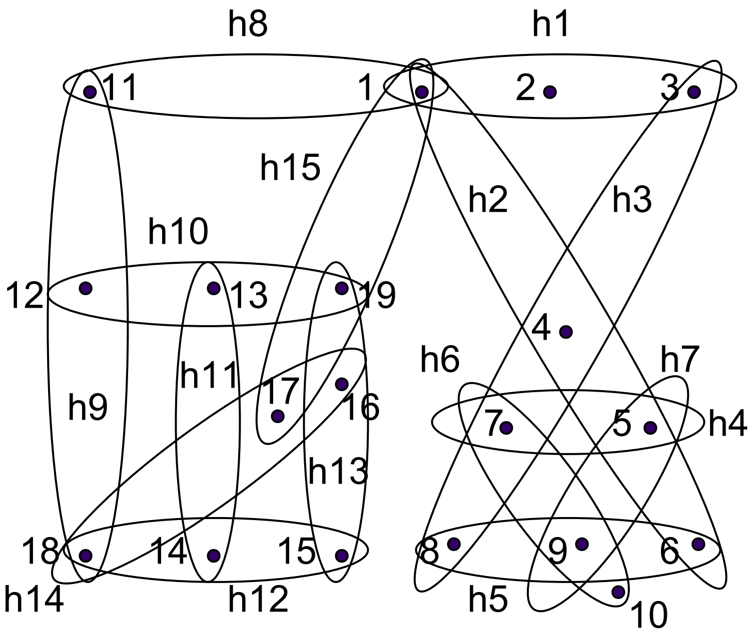
Generalized hypertree decomposition does not include the fourth condition of hypertree decomposition

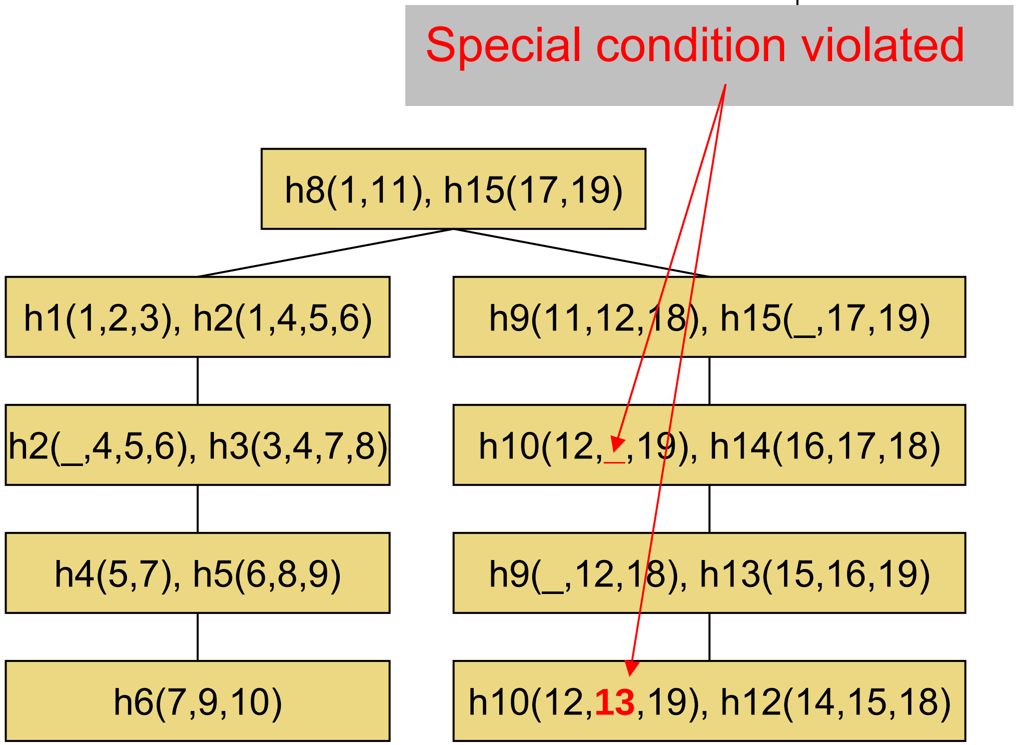




Generalized hypetree decomposition of width 2

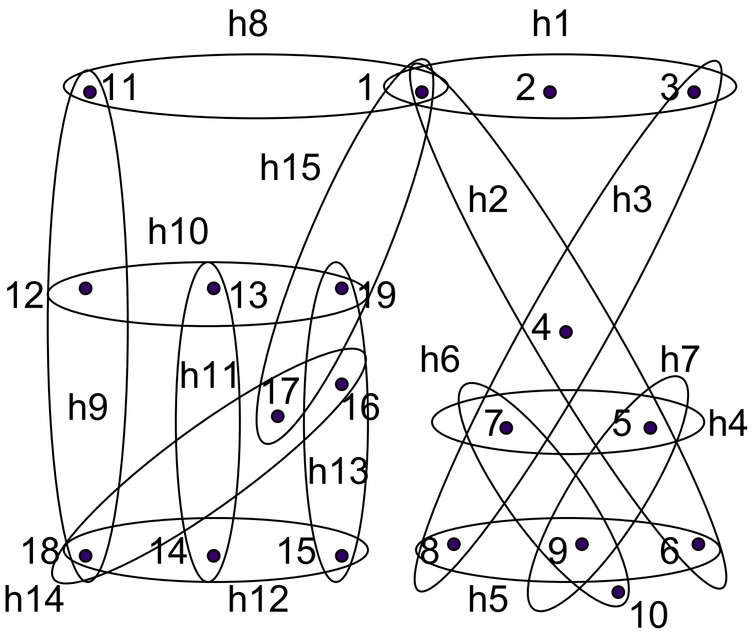
Generalized hypertree decomposition

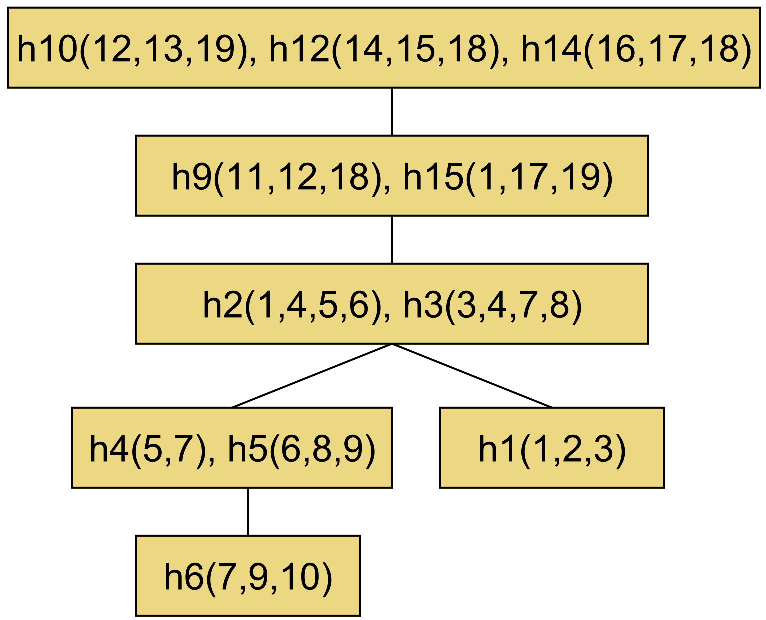




Generalized hypetree decomposition of width 2

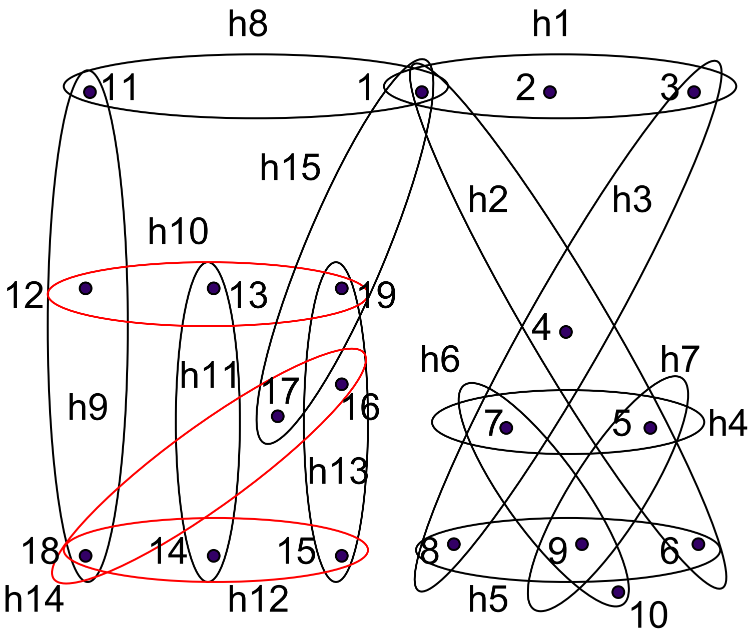
Hypertree decomposition

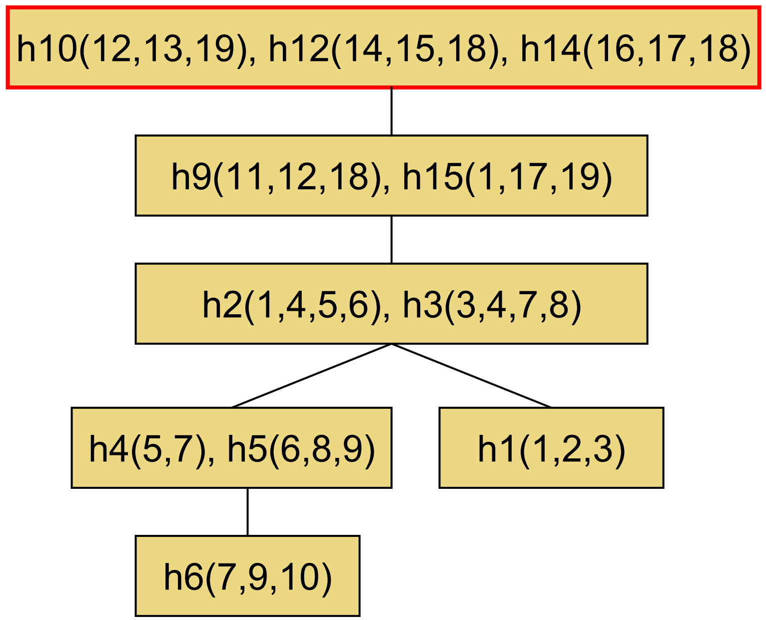




Hypertee decomposition of width 3

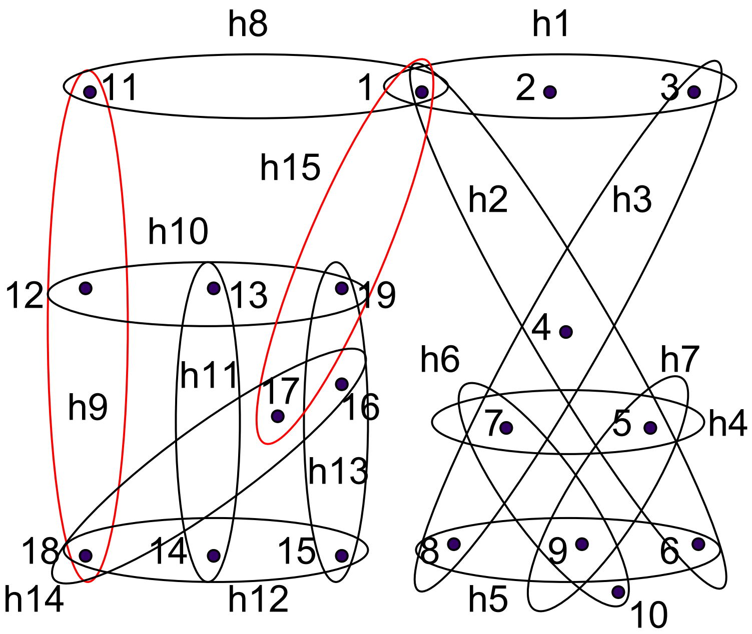
Hypertree decomposition

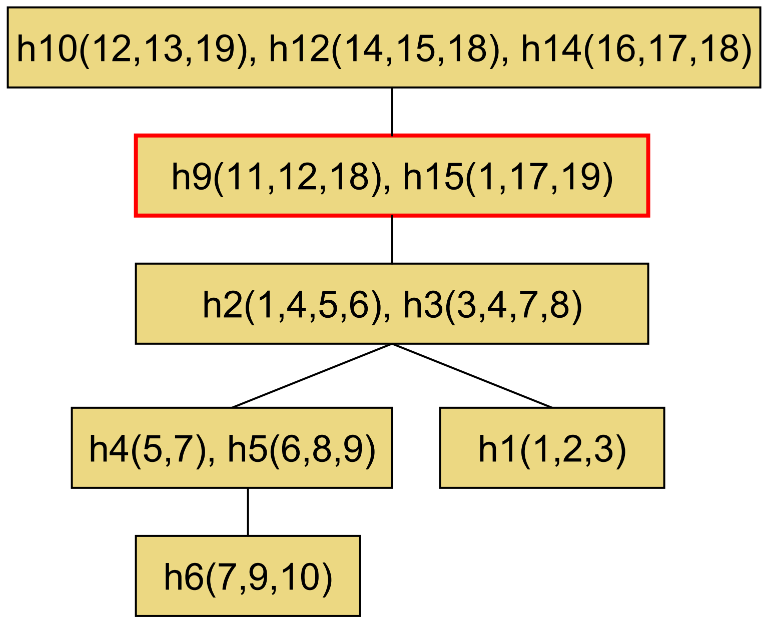




Hypertee decomposition of width 3

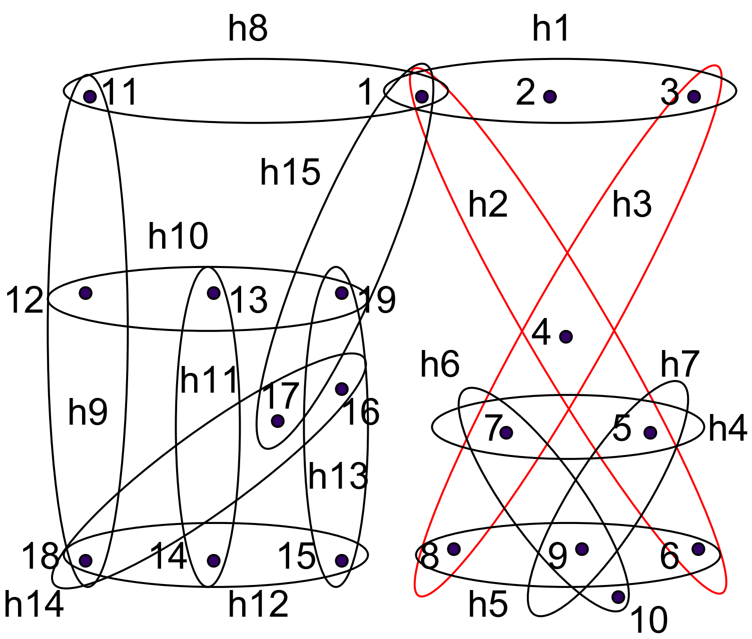
Hypertree decomposition

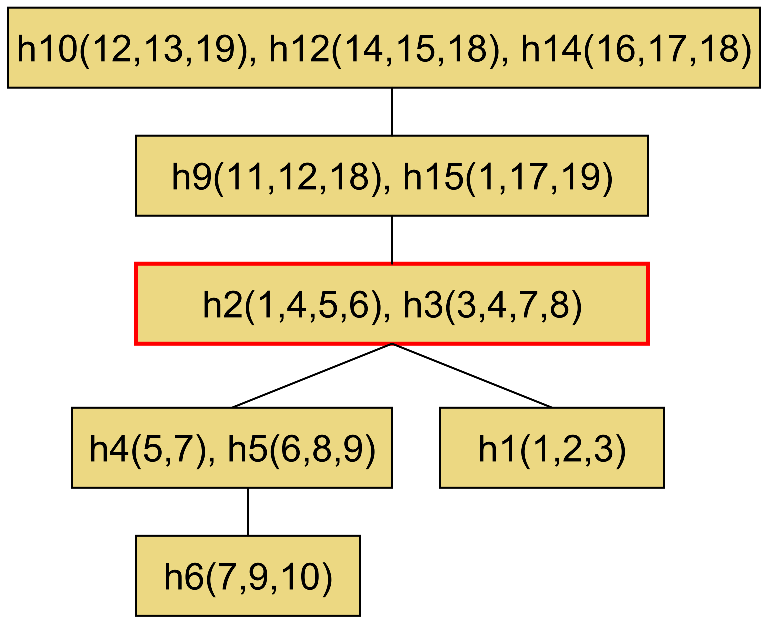




Hypertee decomposition of width 3

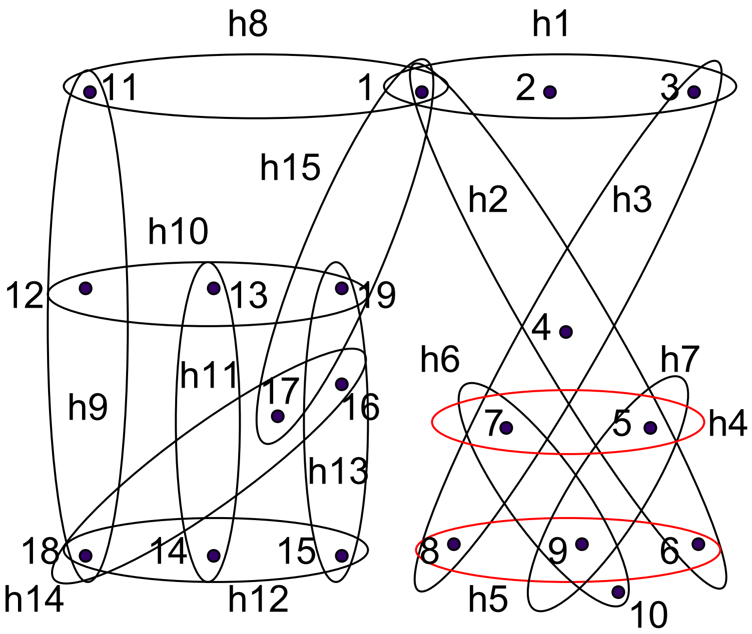
Hypertree decomposition

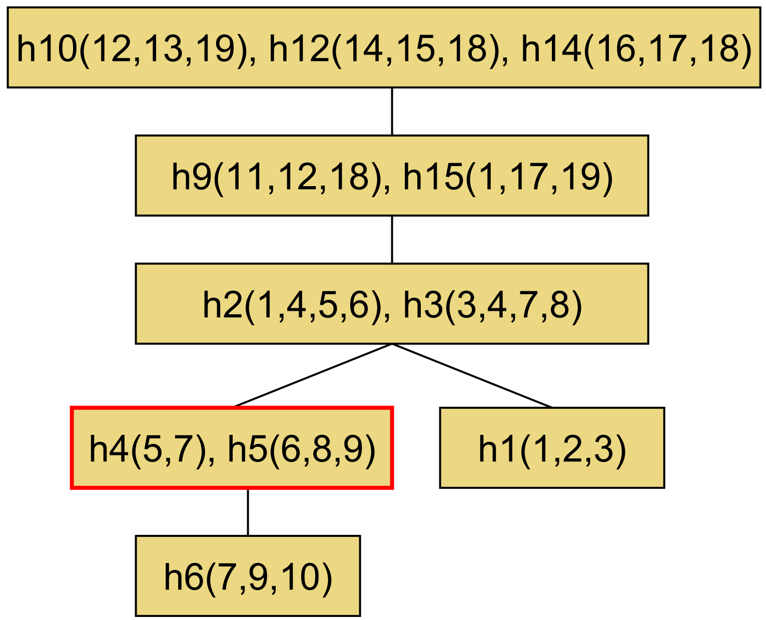




Hypertee decomposition of width 3

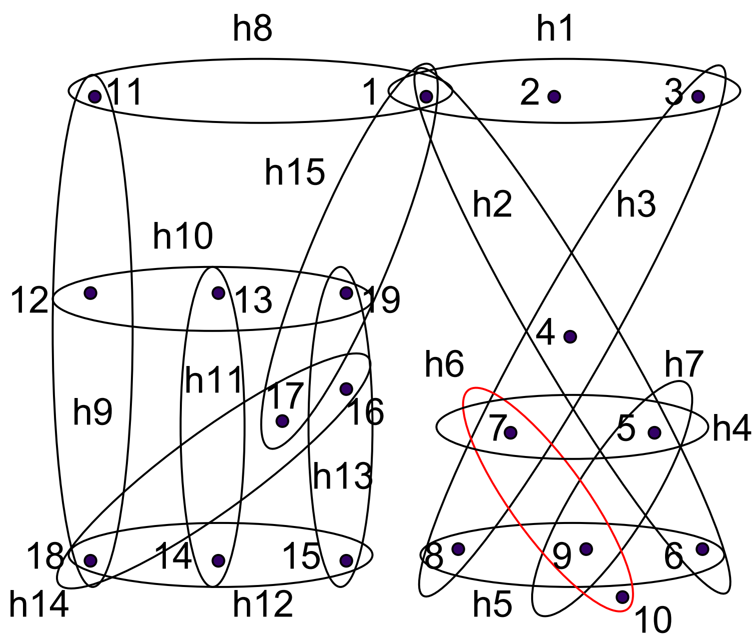
Hypertree decomposition

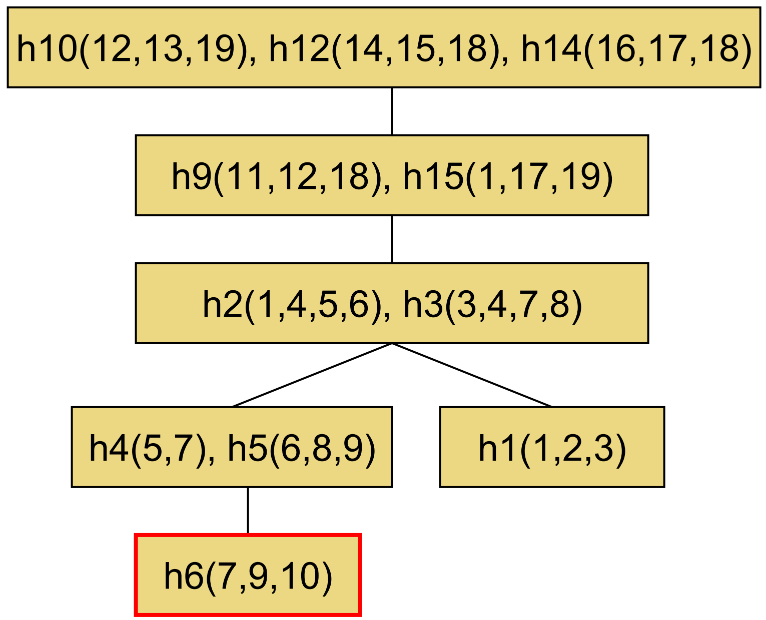




Hypertee decomposition of width 3

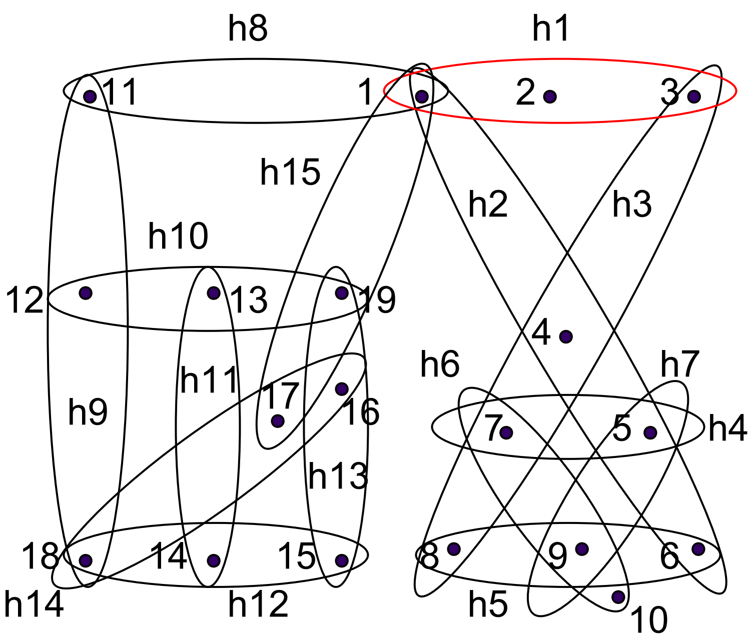
Hypertree decomposition

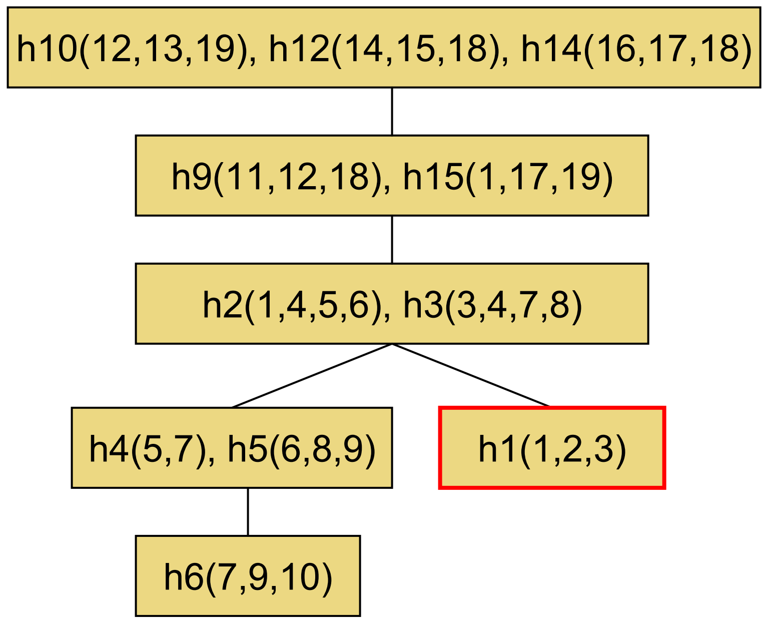




Hypertee decomposition of width 3

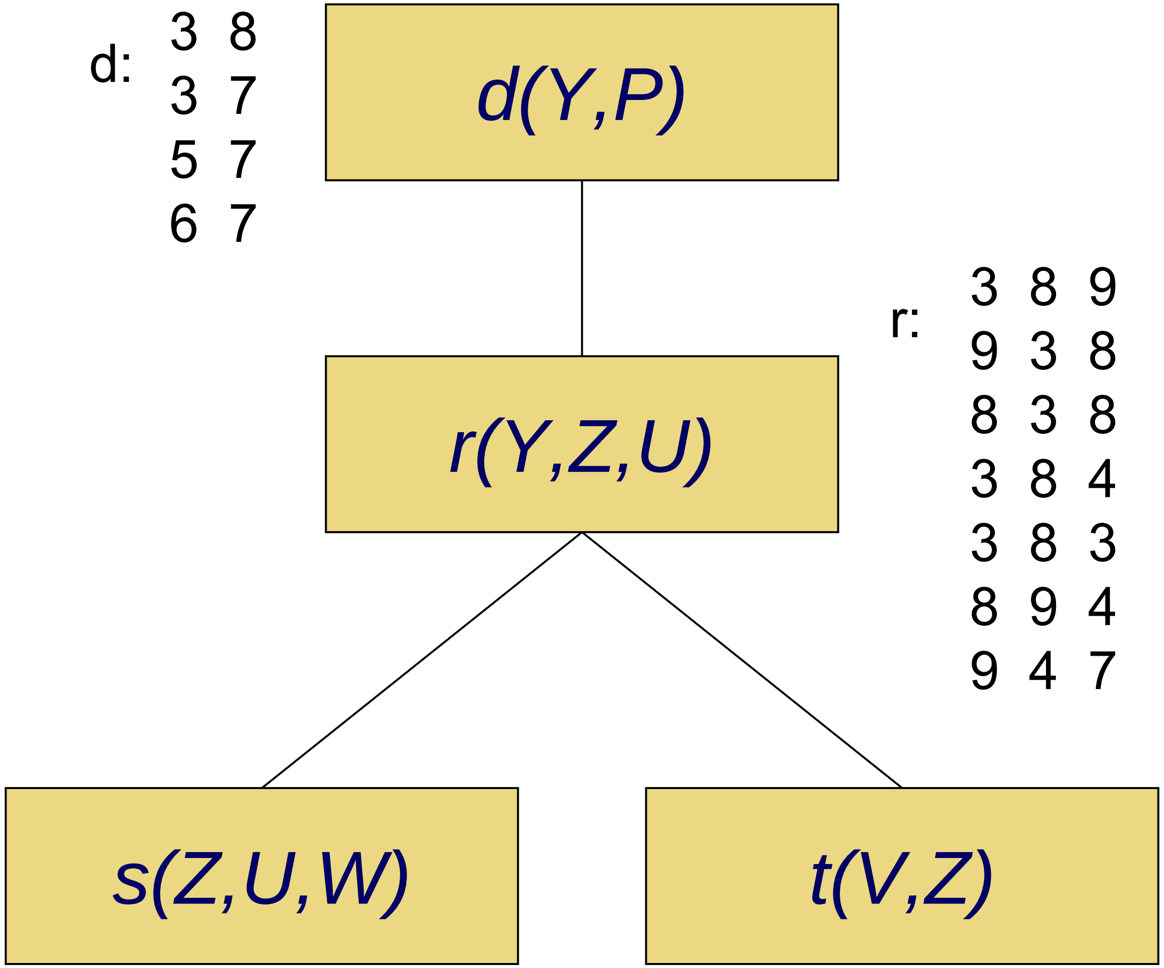
Hypertree decomposition





Hypertee decomposition of width 3

Solving problems based on hypertree decomposition



d:

3	8
3	7
5	7
6	7

r:

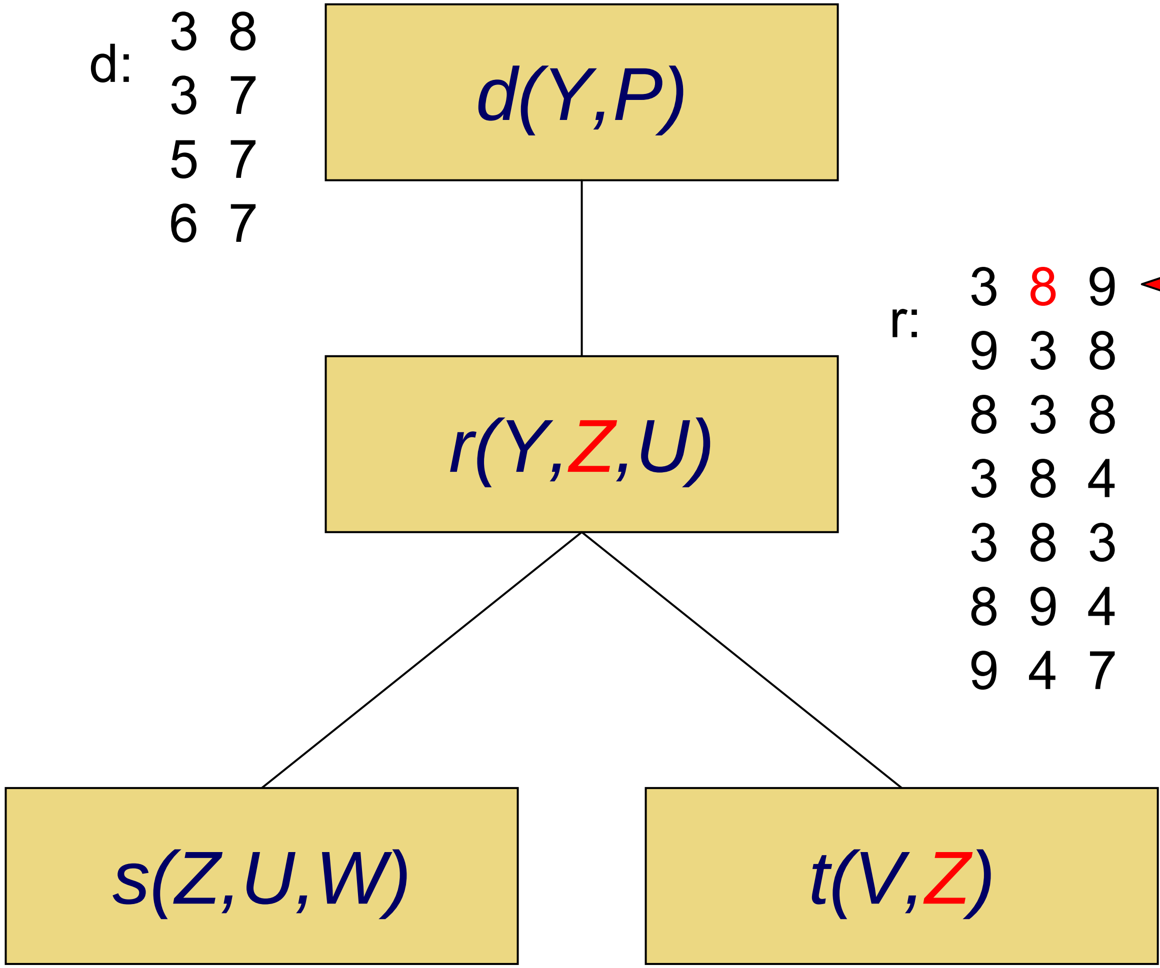
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5



d:

3	8
3	7
5	7
6	7

r:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

s:

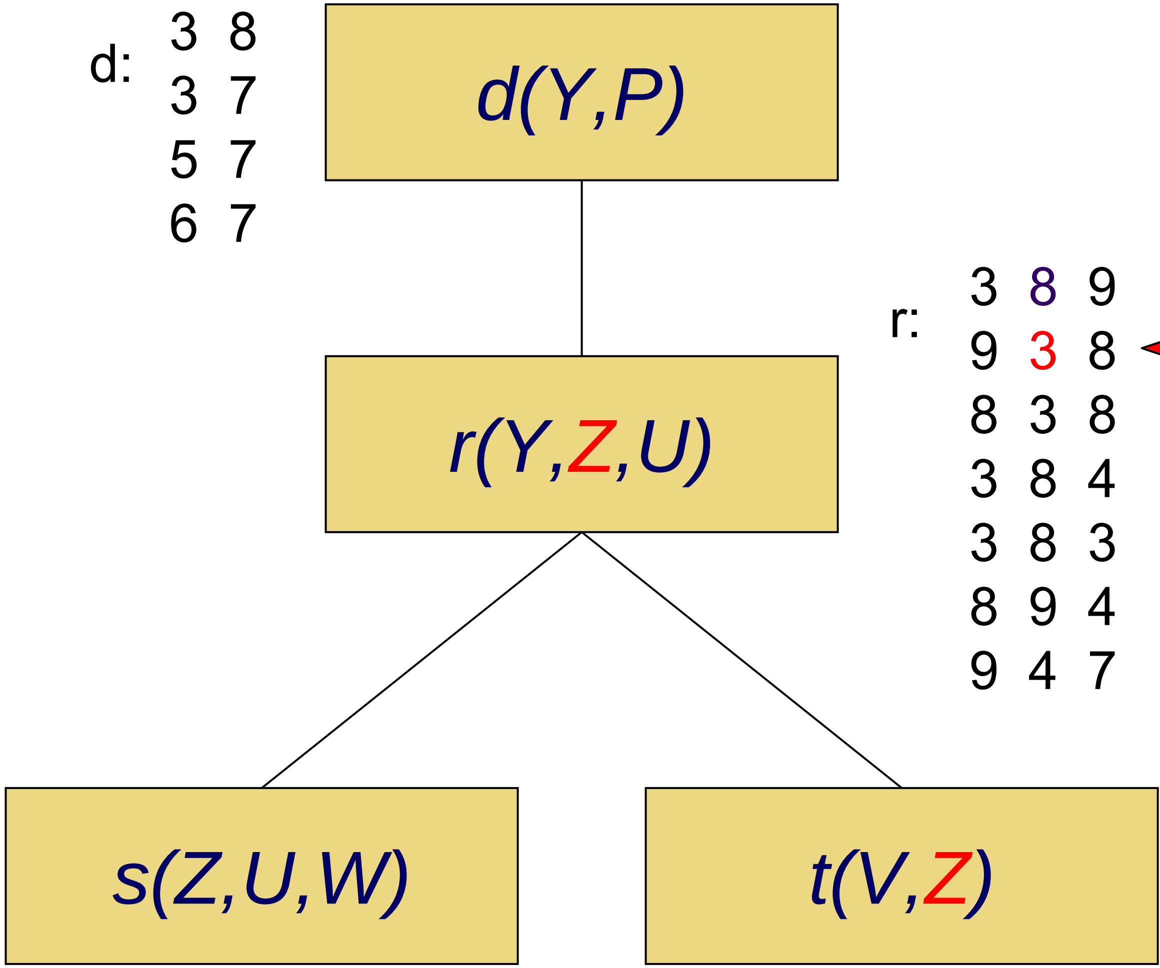
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5

d:

3	8
3	7
5	7
6	7



r:

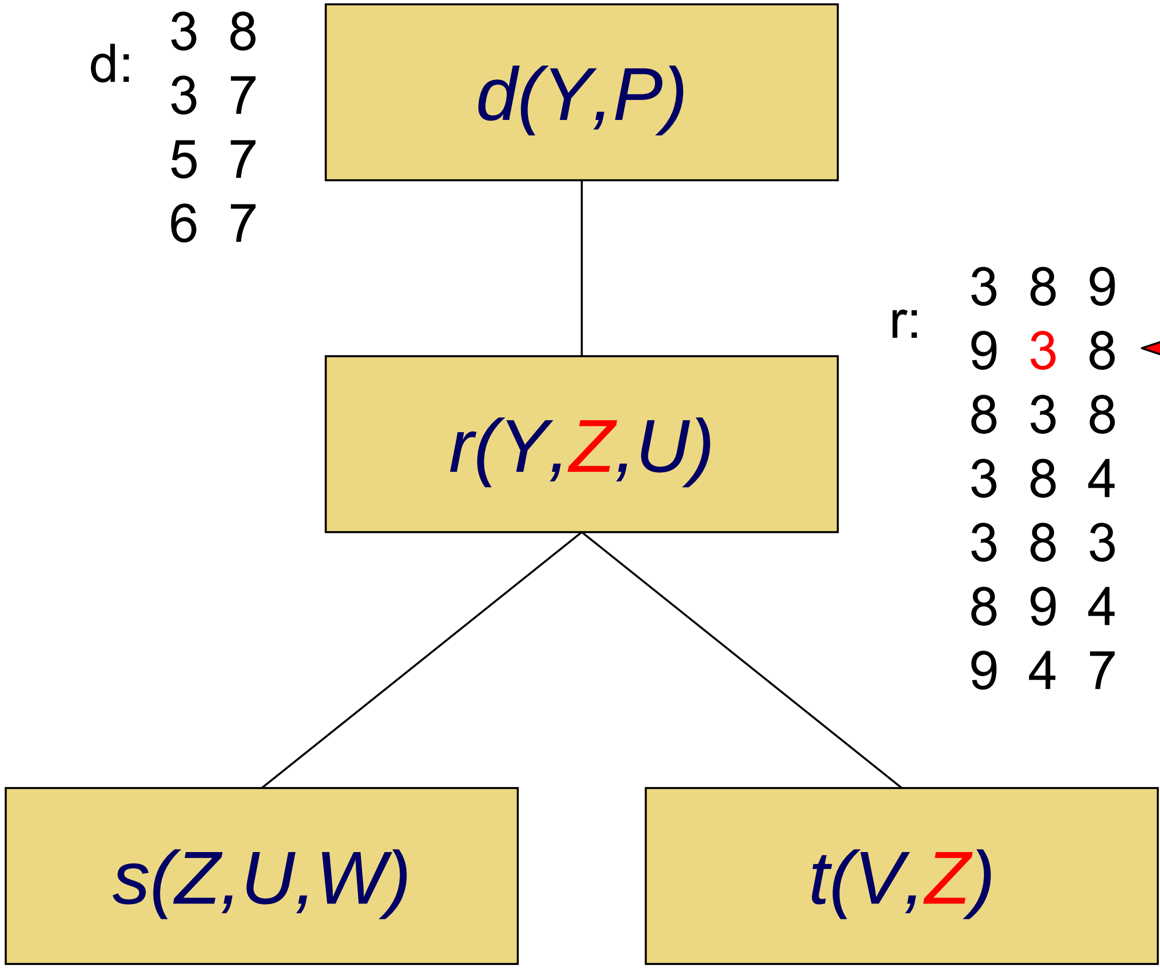
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5



d:

3	8
3	7
5	7
6	7

r:

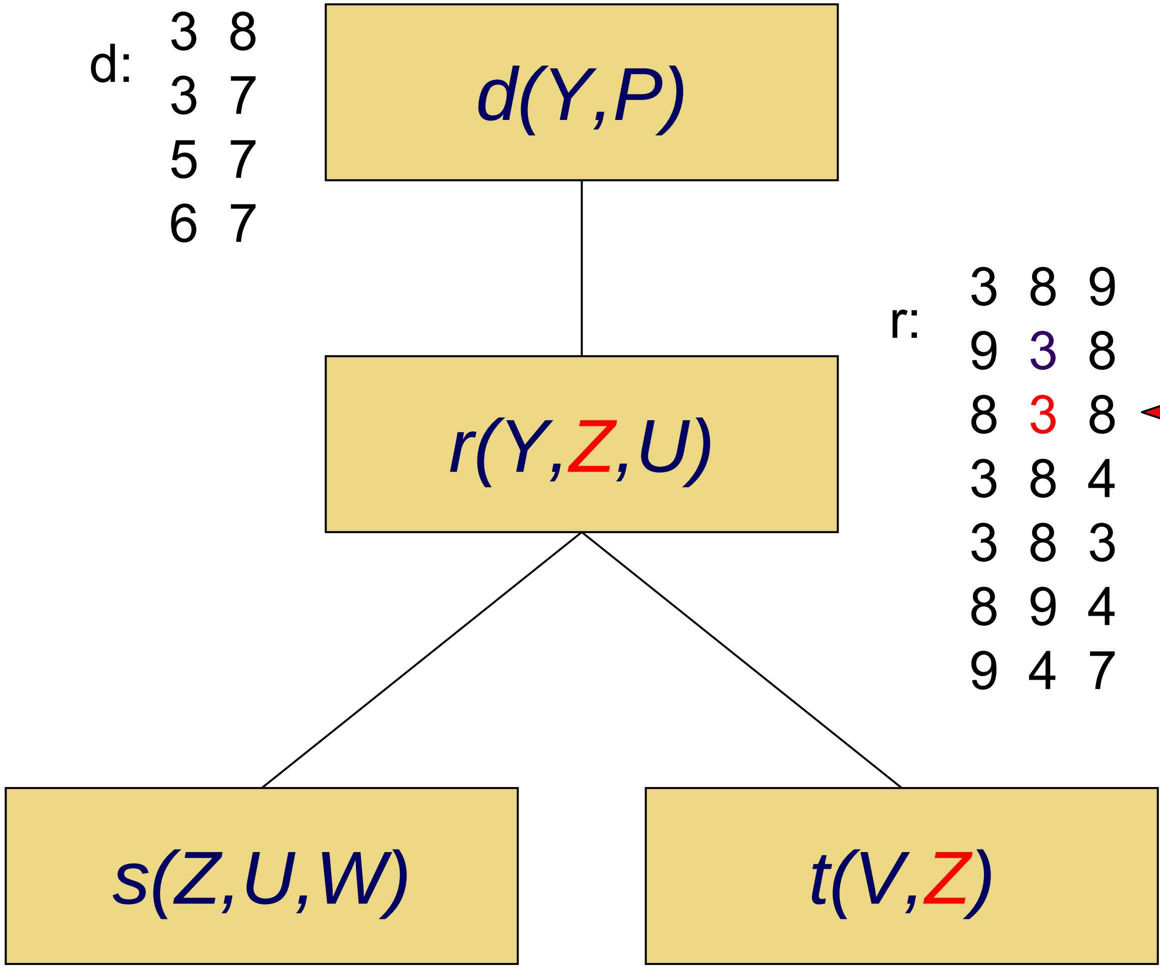
3	8	9
9	3	8 ←
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3 ←
9	5



d:

3	8
3	7
5	7
6	7

r:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

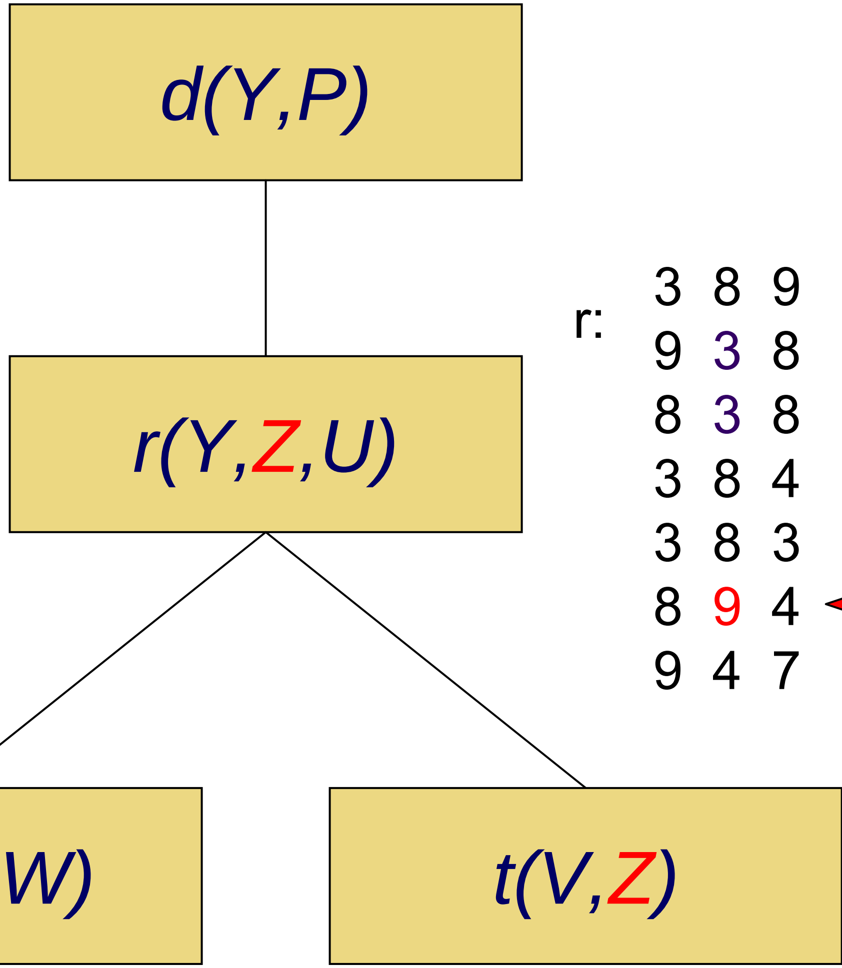
…

s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5



d:

3	8
3	7
5	7
6	7

r:

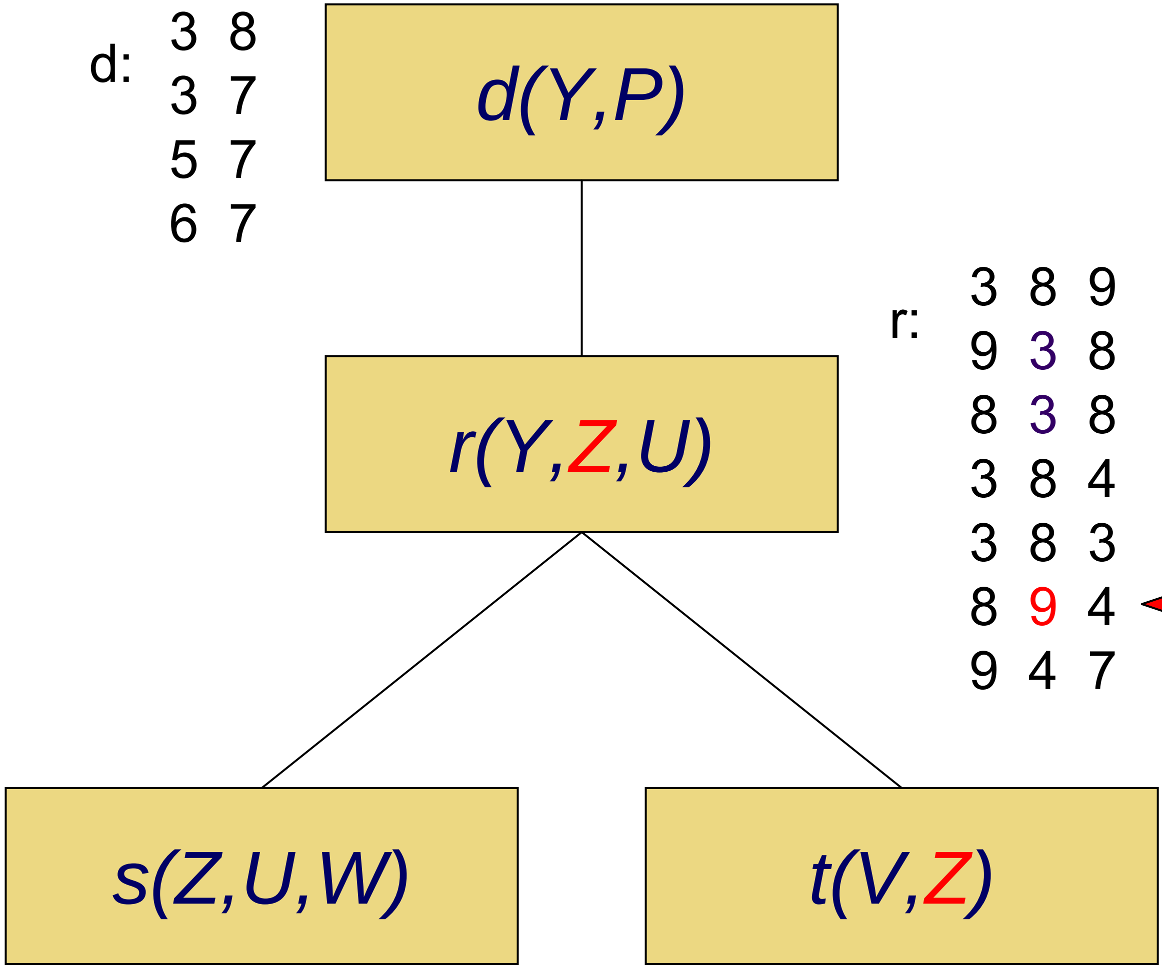
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4 ←
9	4	7

s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8 ←
9	3
9	5



d:

3	8
3	7
5	7
6	7

r:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4 ←
9	4	7

s:

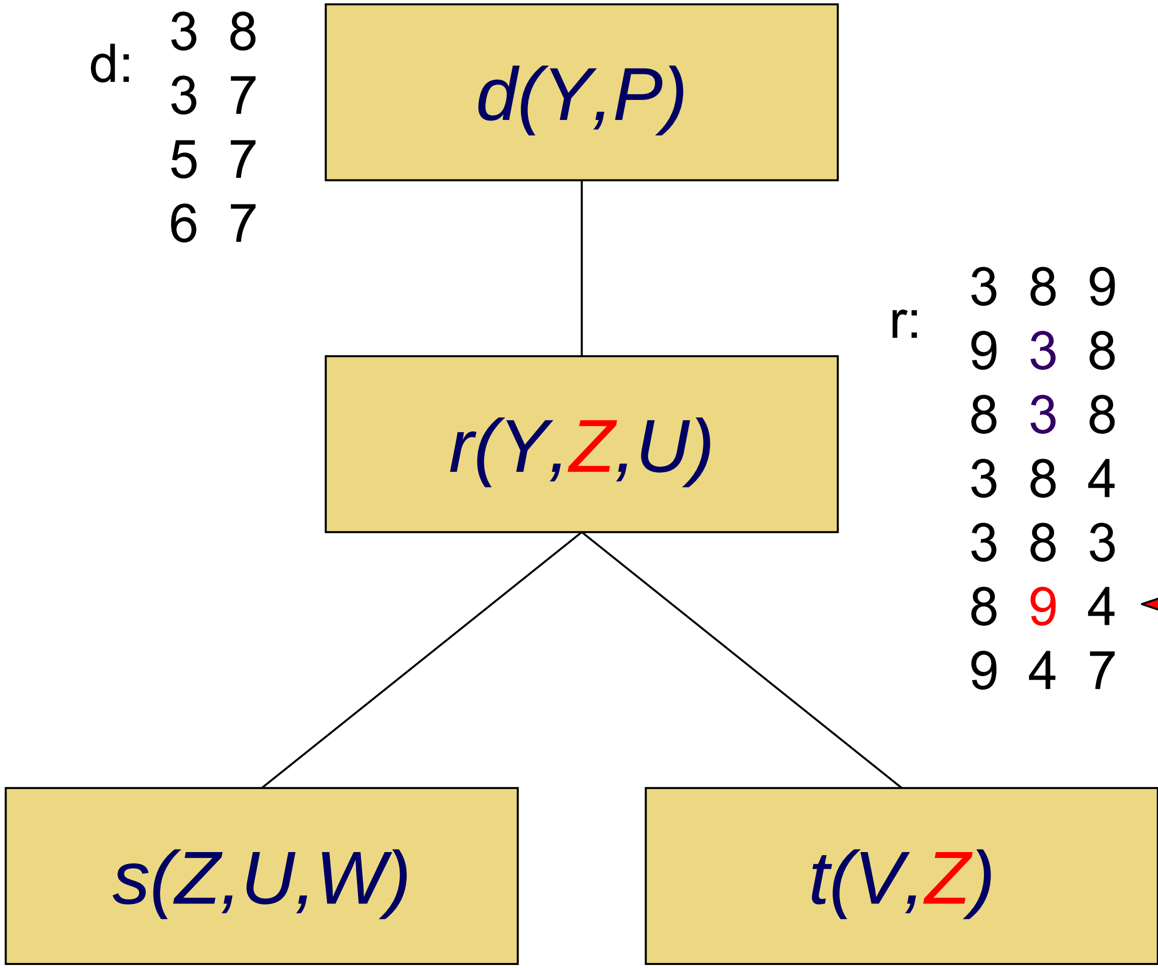
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3 ←
9	5

d:

3	8
3	7
5	7
6	7



r:

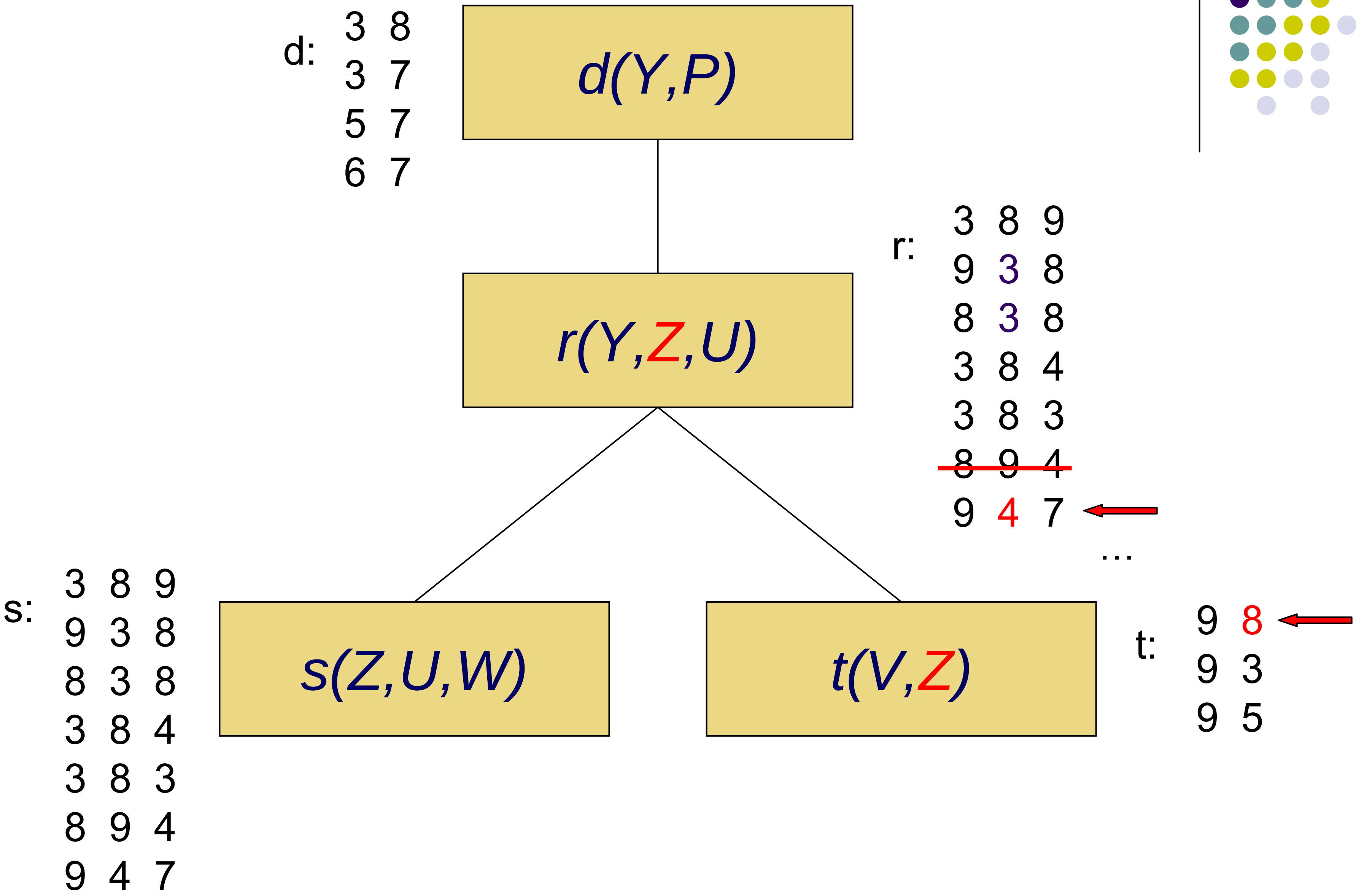
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4 ←
9	4	7

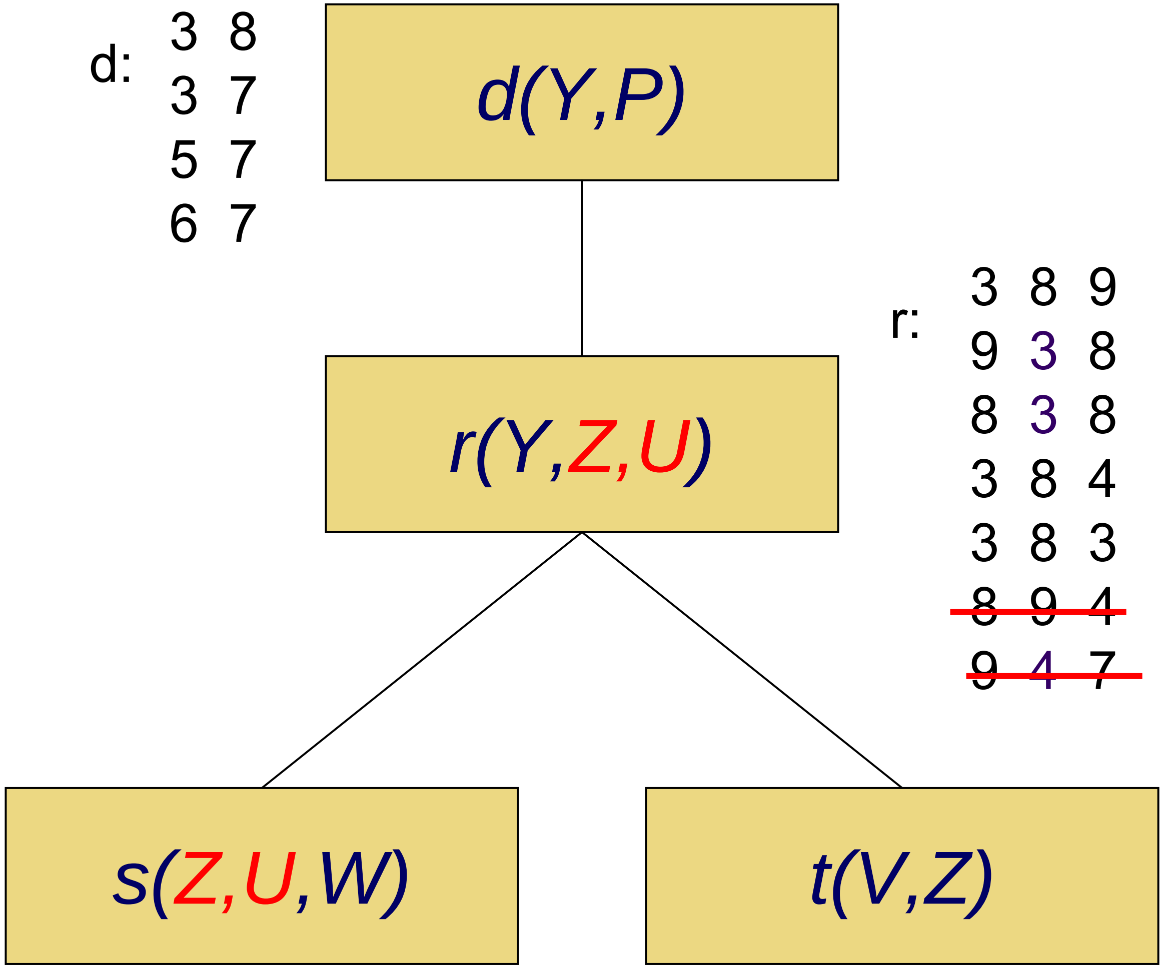
s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5 ←


d(Y,P)
d:
3 8
3 7
5 7
6 7
r(Y,Z,U)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
…
s(Z,U,W)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
t(V,Z)
t:
9 8
9 3
9 5



d:

3	8
3	7
5	7
6	7

r:

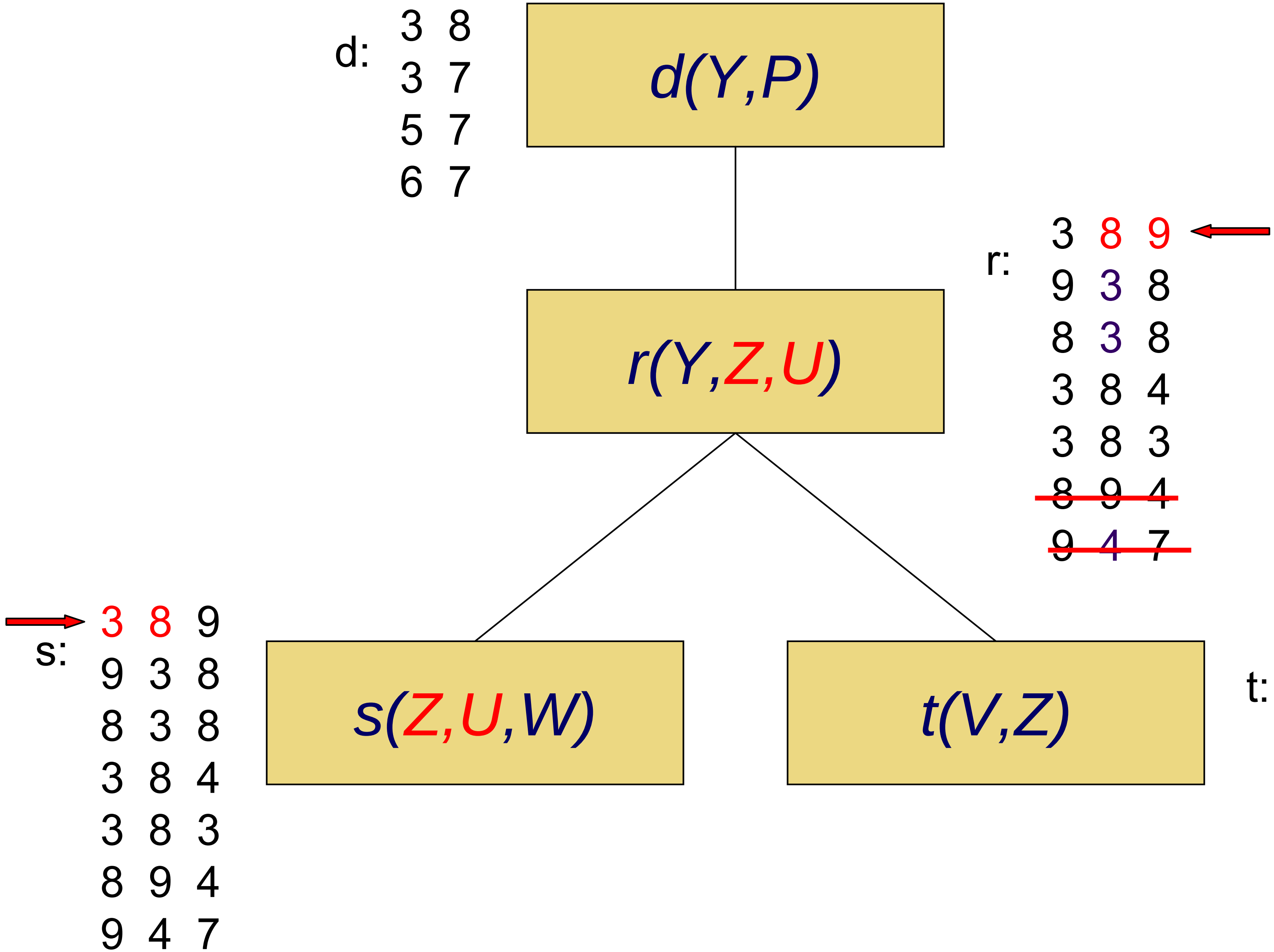
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
~~8~~	~~9~~	~~4~~
~~9~~	~~4~~	~~7~~

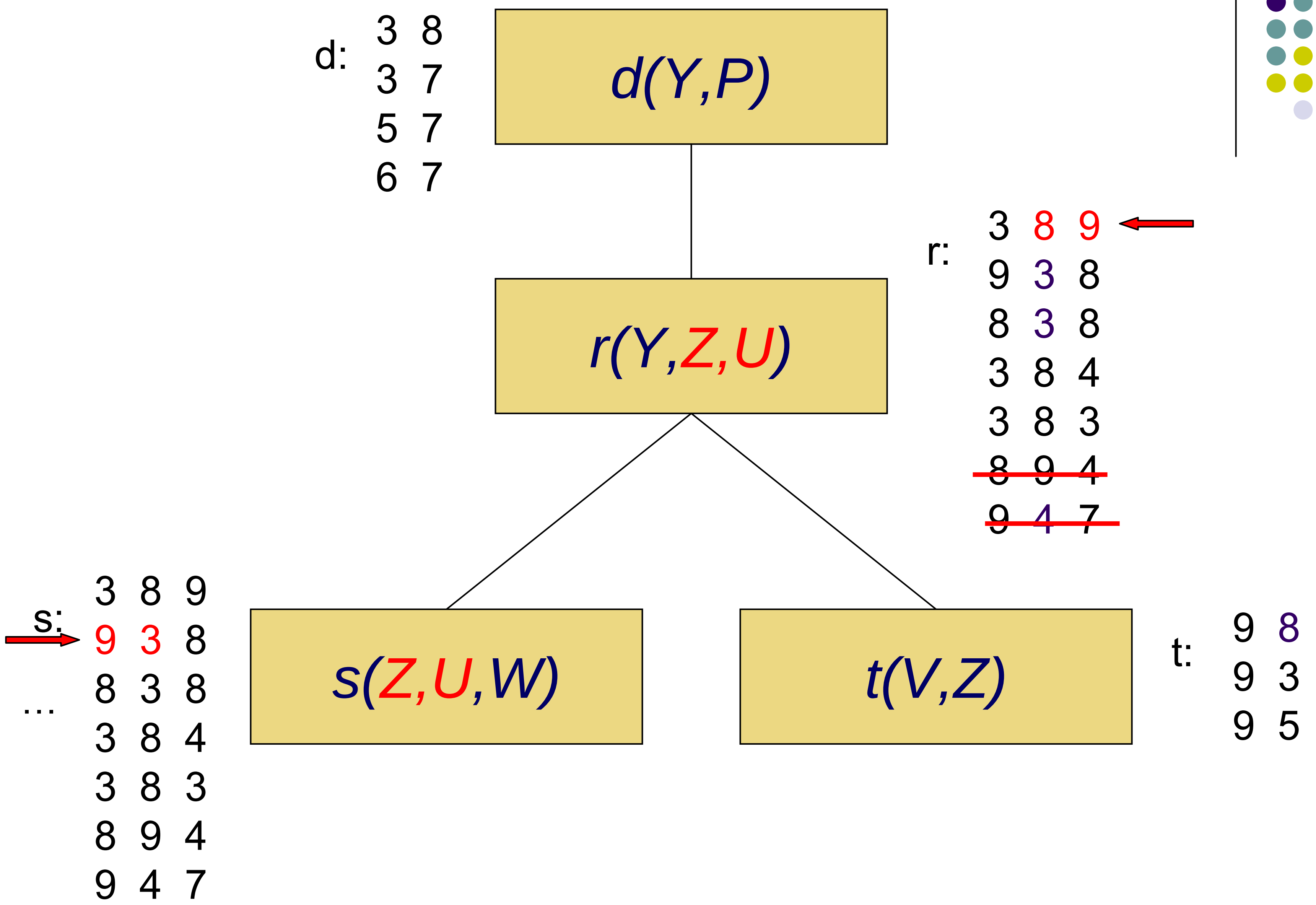
s:

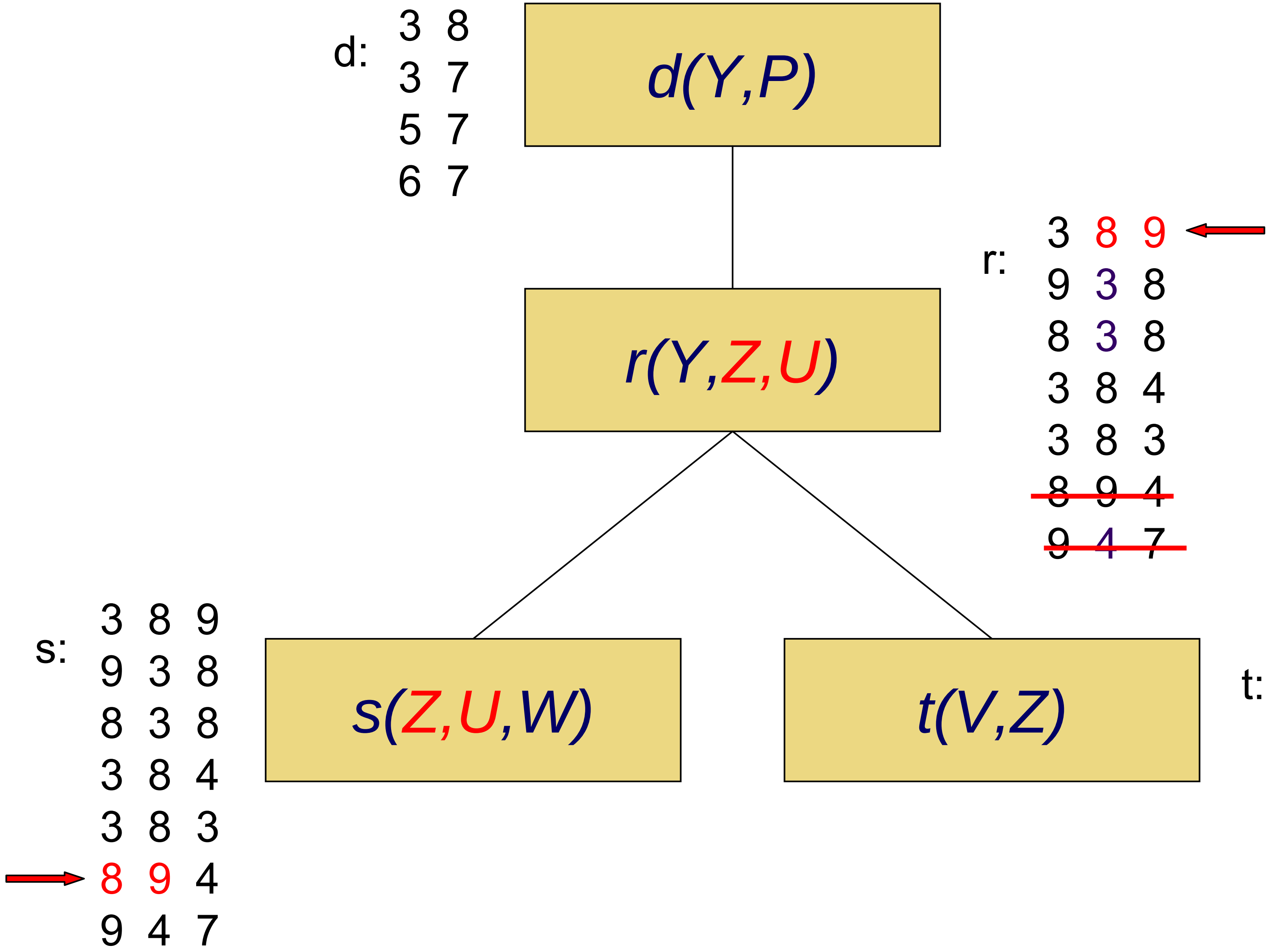
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

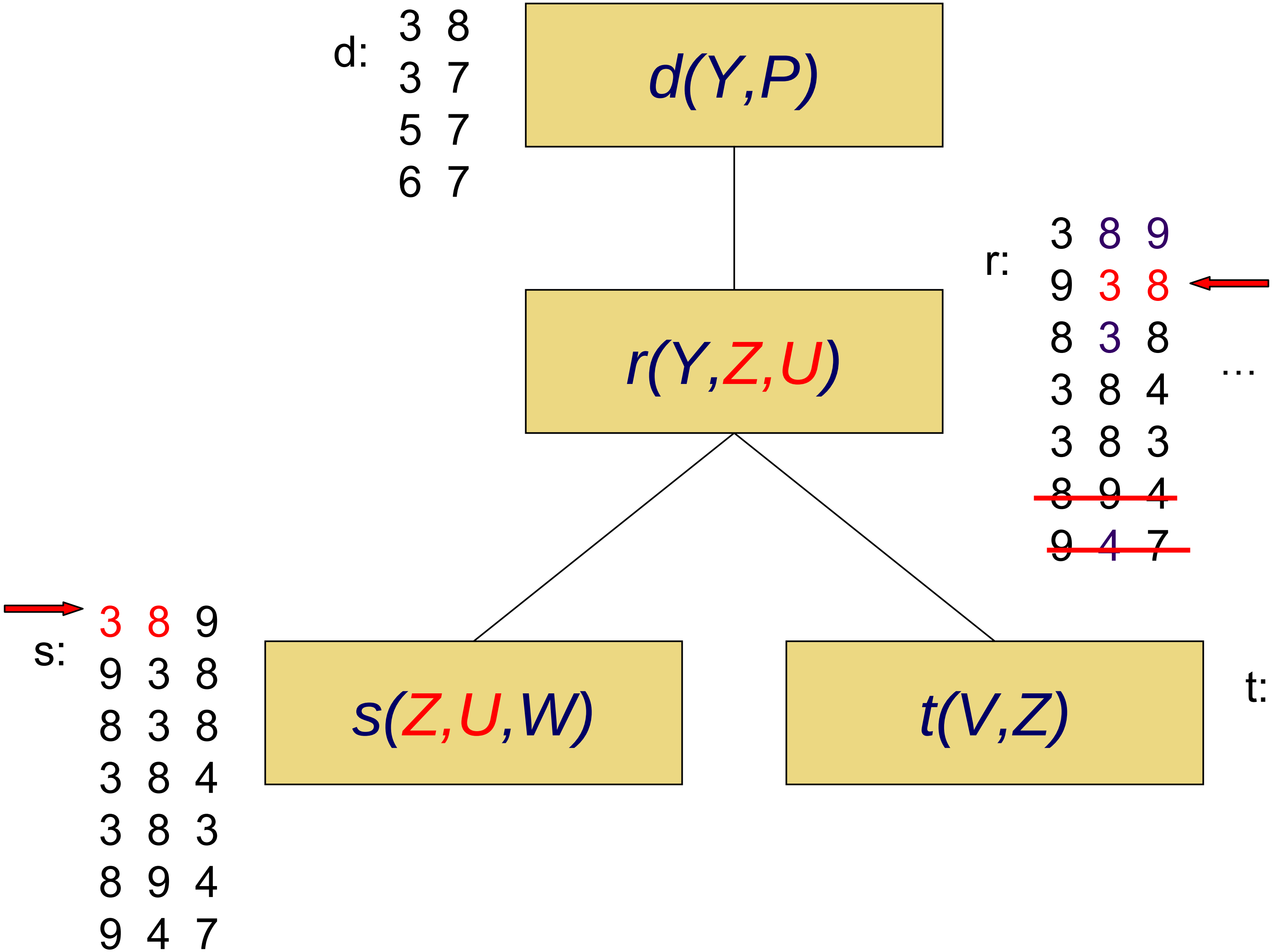
t:

9	8
9	3
9	5


d:
3 8
3 7
5 7
6 7
d(Y,P)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
r(Y,Z,U)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s(Z,U,W)
t(V,Z)
t:
9 8
9 3
9 5

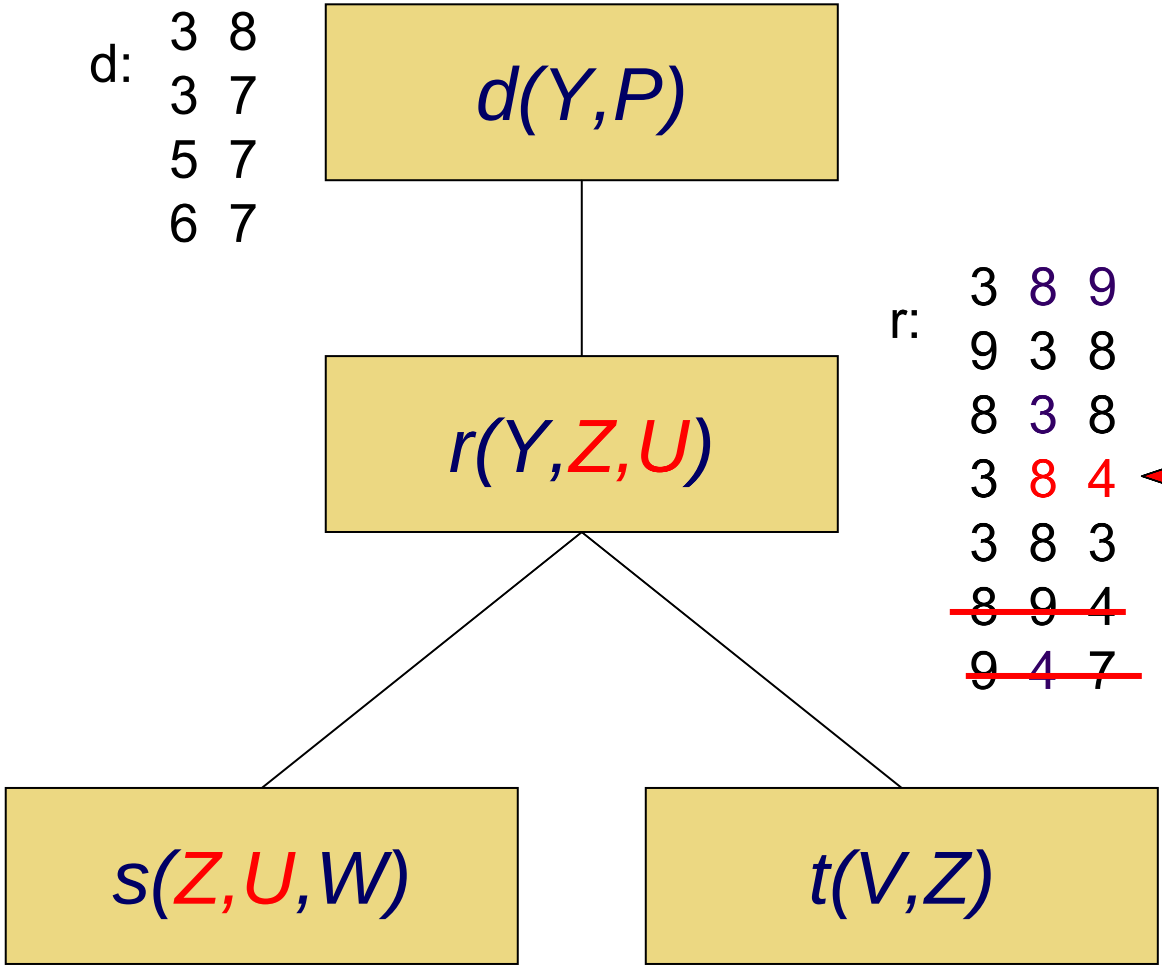

d:
3 8
3 7
5 7
6 7
d(Y,P)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
r(Y,Z,U)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
…
s(Z,U,W)
t(V,Z)
t:
9 8
9 3
9 5


d(Y,P)
d:
3 8
3 7
5 7
6 7
r(Y,Z,U)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s(Z,U,W)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
t(V,Z)
t:
9 8
9 3
9 5


d:
3 8
3 7
5 7
6 7
d(Y,P)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
...
r(Y,Z,U)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s(Z,U,W)
t(V,Z)
t:
9 8
9 3
9 5

d:

3	8
3	7
5	7
6	7



r:

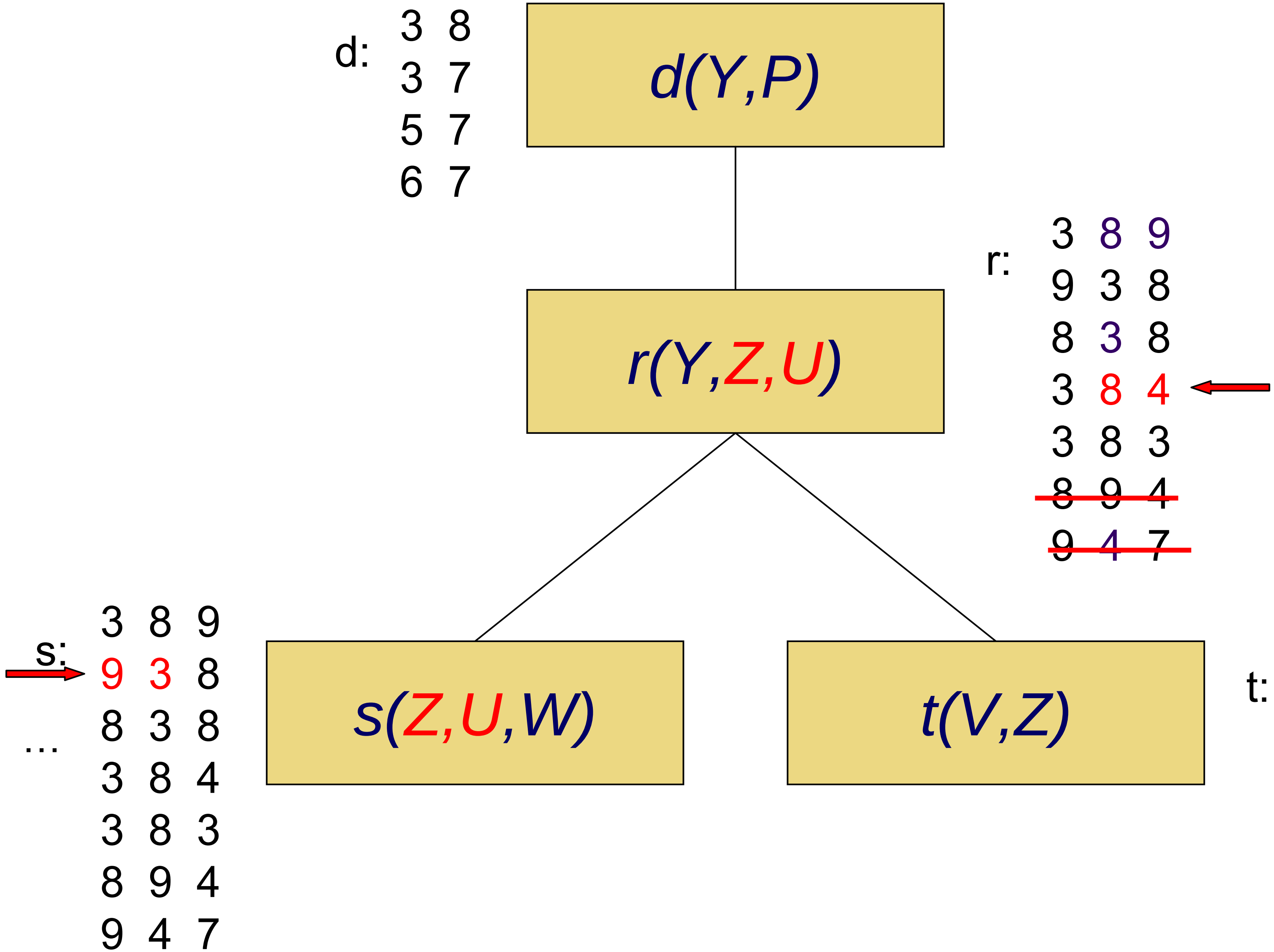
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
~~8~~	~~9~~	~~4~~
~~9~~	~~4~~	~~7~~

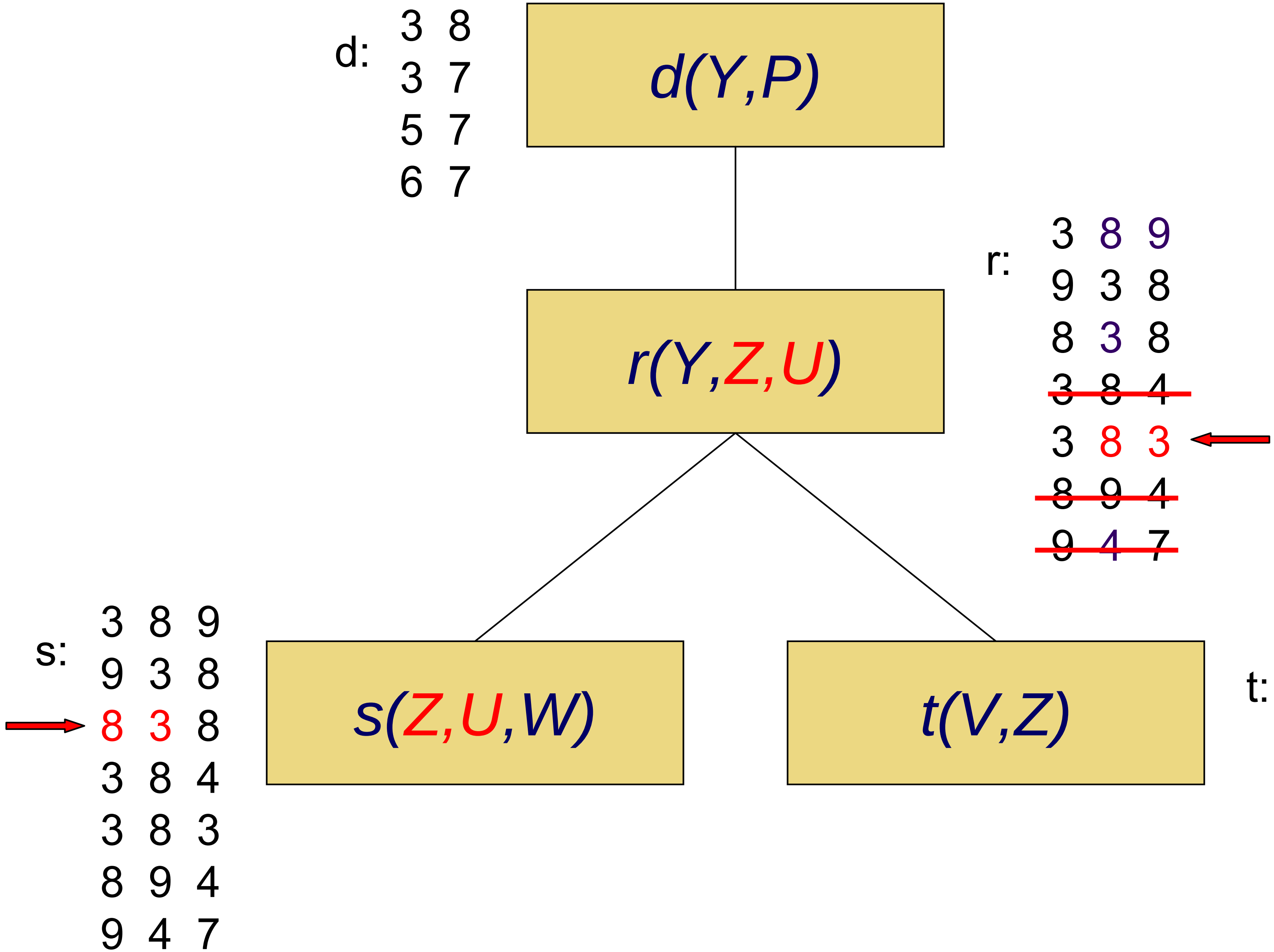
s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

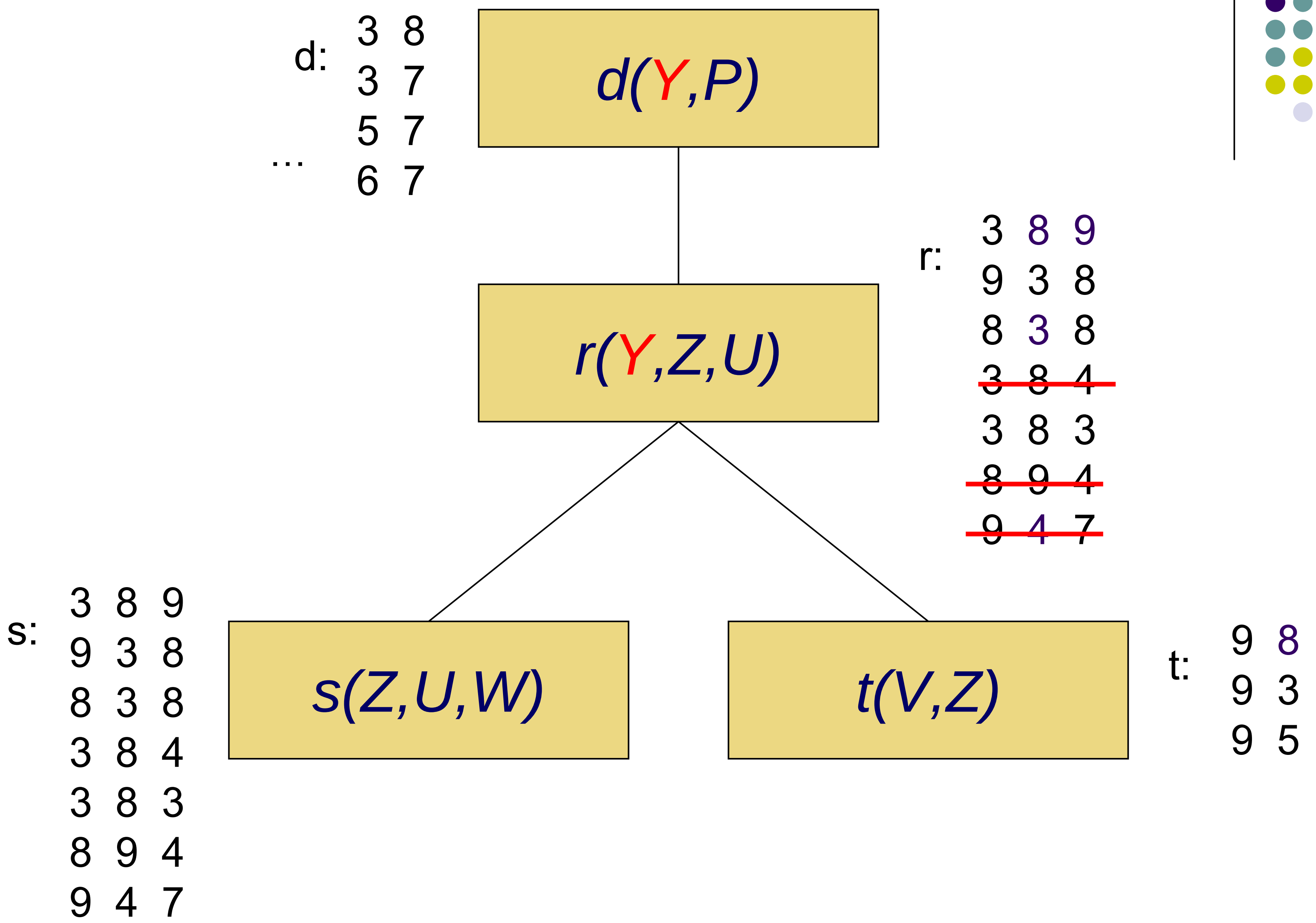
t:

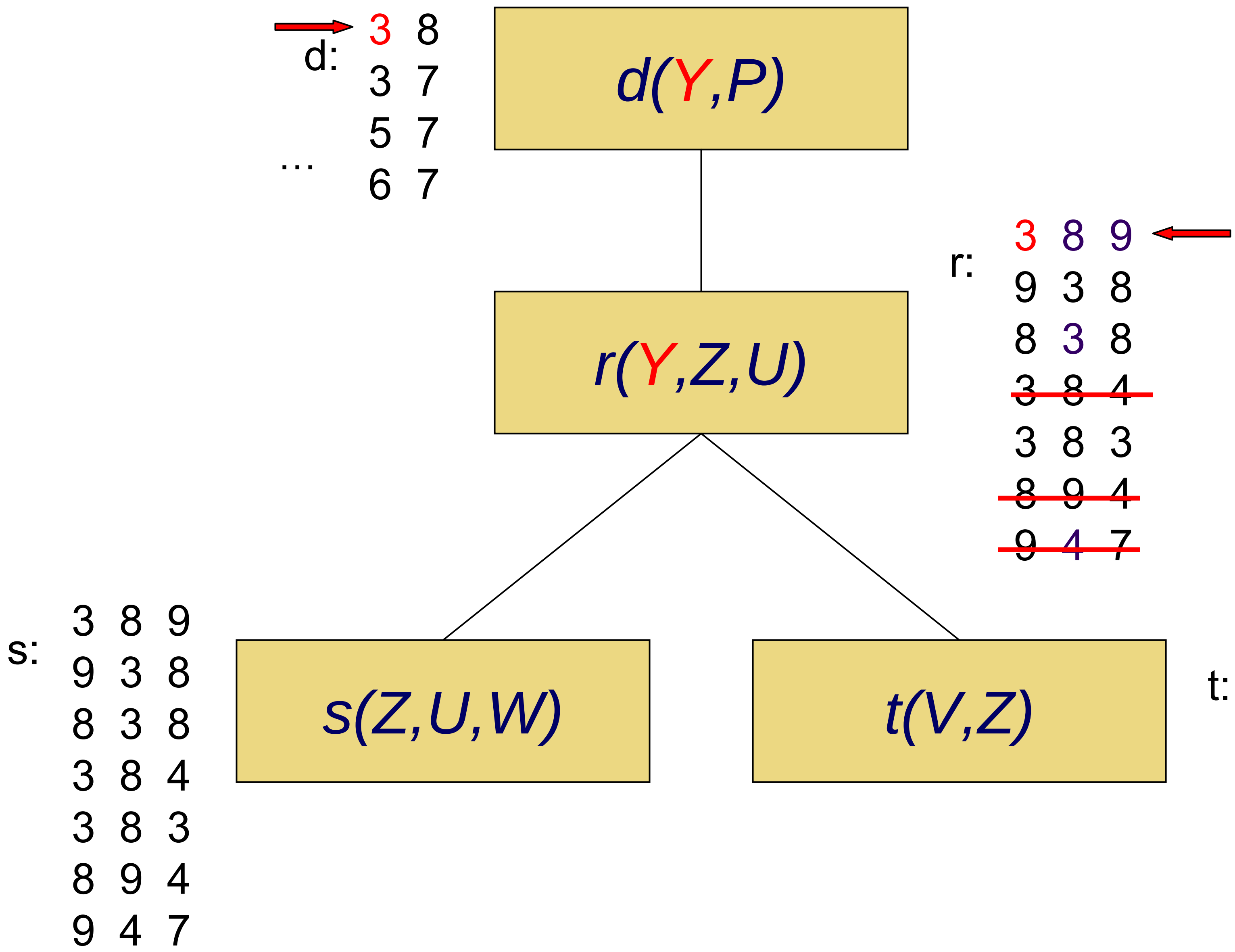
9	8
9	3
9	5


d(Y,P)
d:
3 8
3 7
5 7
6 7
r(Y,Z,U)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s(Z,U,W)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
...
t(V,Z)
t:
9 8
9 3
9 5

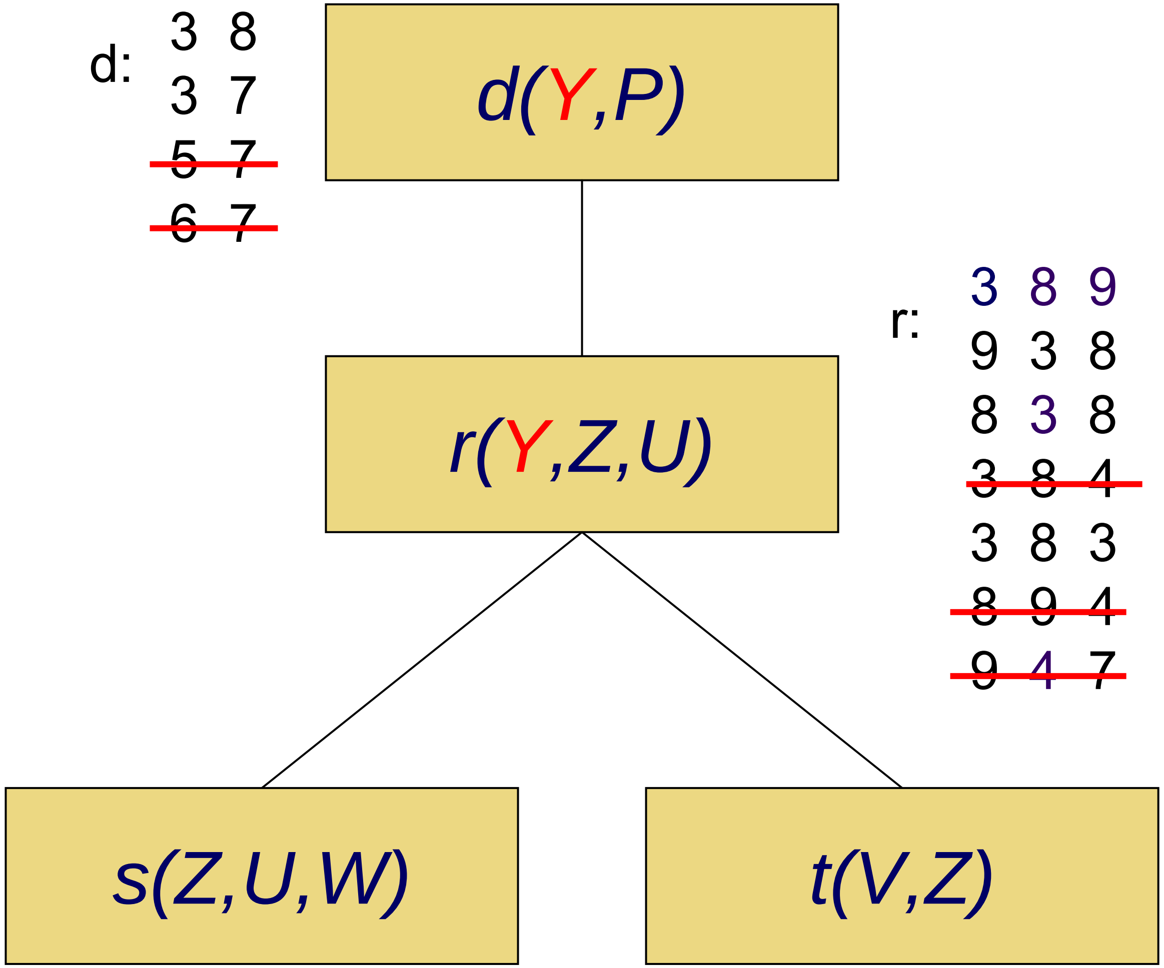


d:
3 8
3 7
5 7
6 7
d(Y,P)
r(Y,Z,U)
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s(Z,U,W)
t(V,Z)
t:
9 8
9 3
9 5


d(Y,P)
r(Y,Z,U)
s(Z,U,W)
t(V,Z)
d:
3 8
3 7
5 7
…
6 7
r:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
t:
9 8
9 3
9 5



d:
3 8
3 7
5 7
...
6 7
d(Y,P)
r(Y,Z,U)
r:
3 8 9
9 3 8
8 3 8
~~3 8 4~~
3 8 3
~~8 9 4~~
~~9 4 7~~
s(Z,U,W)
t(V,Z)
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
t:
9 8
9 3
9 5



d:

3	8
3	7
~~5~~	~~7~~
~~6~~	~~7~~

r:

3	8	9
9	3	8
8	3	8
~~3~~	~~8~~	~~4~~
3	8	3
~~8~~	~~9~~	~~4~~
~~9~~	~~4~~	~~7~~

s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5

Algorithms for Generalized Hypertree Decomposition

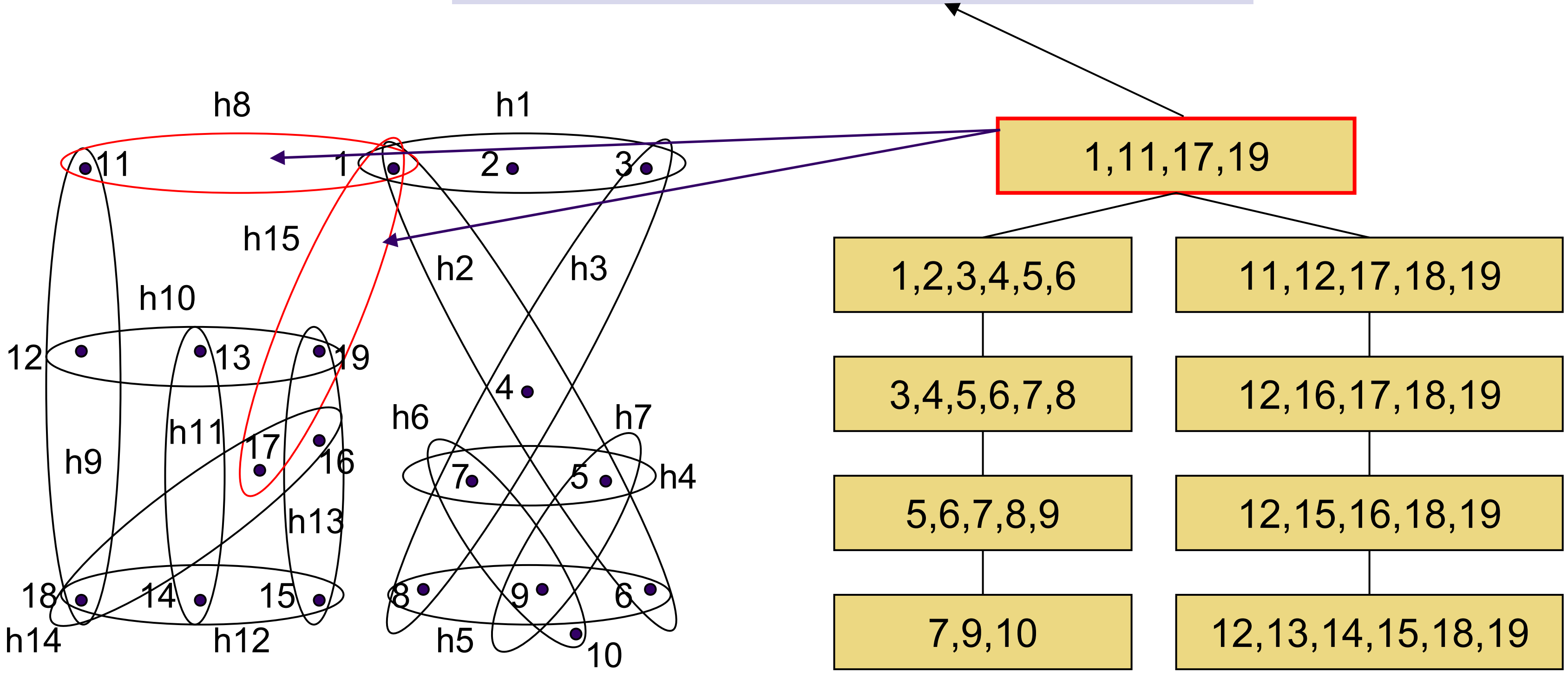
- Methods based on tree decomposition
  - Generalized hypertree decomposition can be generated by algorithms for tree decomposition + Set Covering
- Hypertree decomposition based on hypergraph partitioning
- Exact methods
- Literature and benchmark instances for hypertree decomposition:

  http://www.dbai.tuwien.ac.at/proj/hypertree/

  http://wwwinfo.deis.unical.it/~frank/Hypertrees/

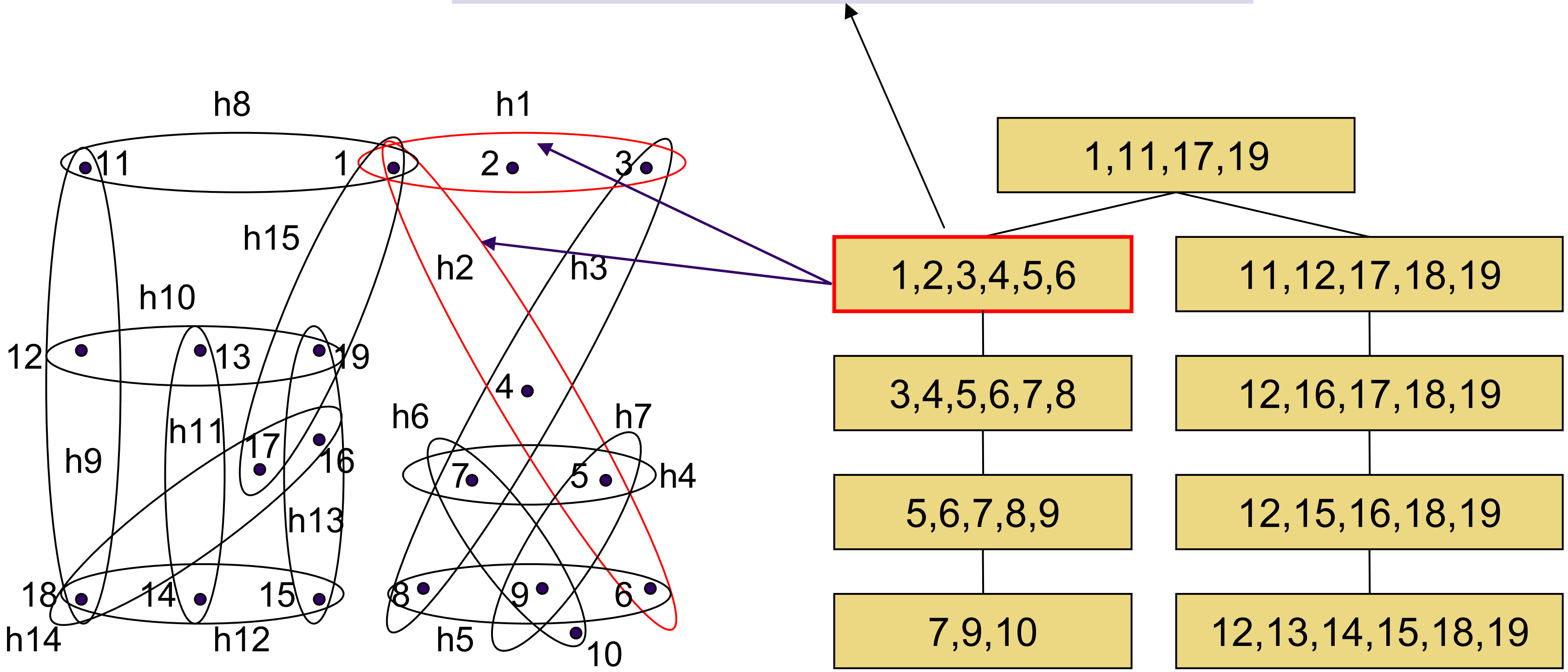
Constructing generalized hypertree decomposition from tree decomposition

Apply for each node of tree decomposition set covering



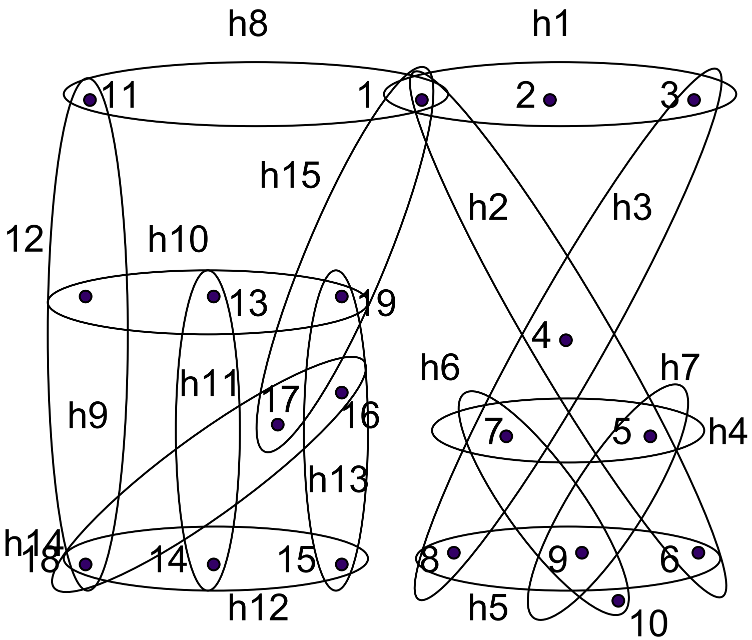
Constructing generalized hypertree decomposition from tree decomposition

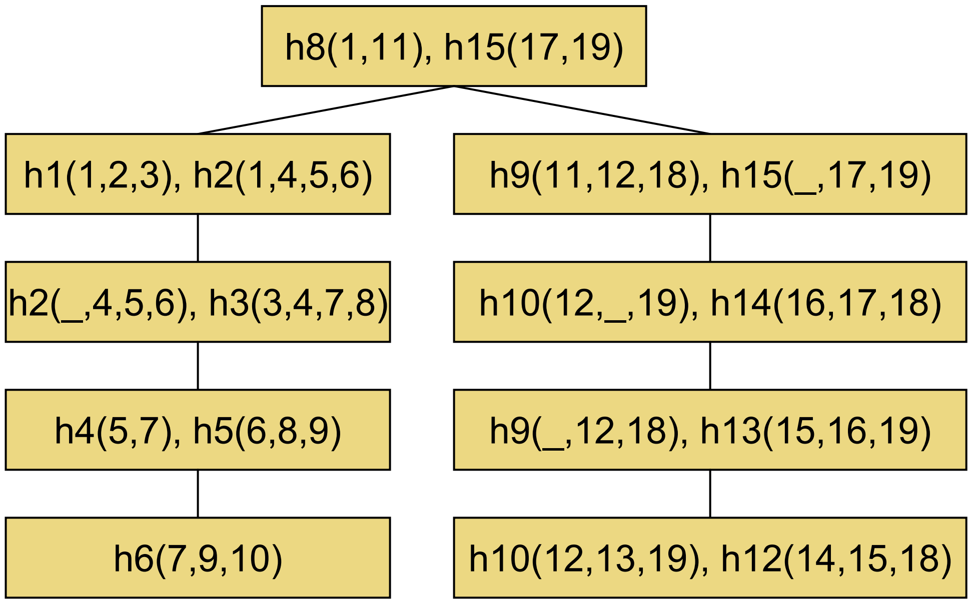
Apply for each node of tree decomposition set covering



…

Generalized hypertree decomposition





Generalized hypetree decomposition of width 2

Hypertree decomposition based on hypergraph partitioning

- A method for generation of generalized hypertree decompositions based on recursive partitioning of the hypergraph is described in:

  A. Dermaku, T. Ganzow, G. Gottlob, B. McMahan, N. Musliu, M. Samer. *Heuristic Methods for Hypertree Decompositions. MICAI 2008: Lecture Notes in Artificial Intelligence, Volume 5317, pages 1-11, 2008, Springer.*

Hypergraph partitioning

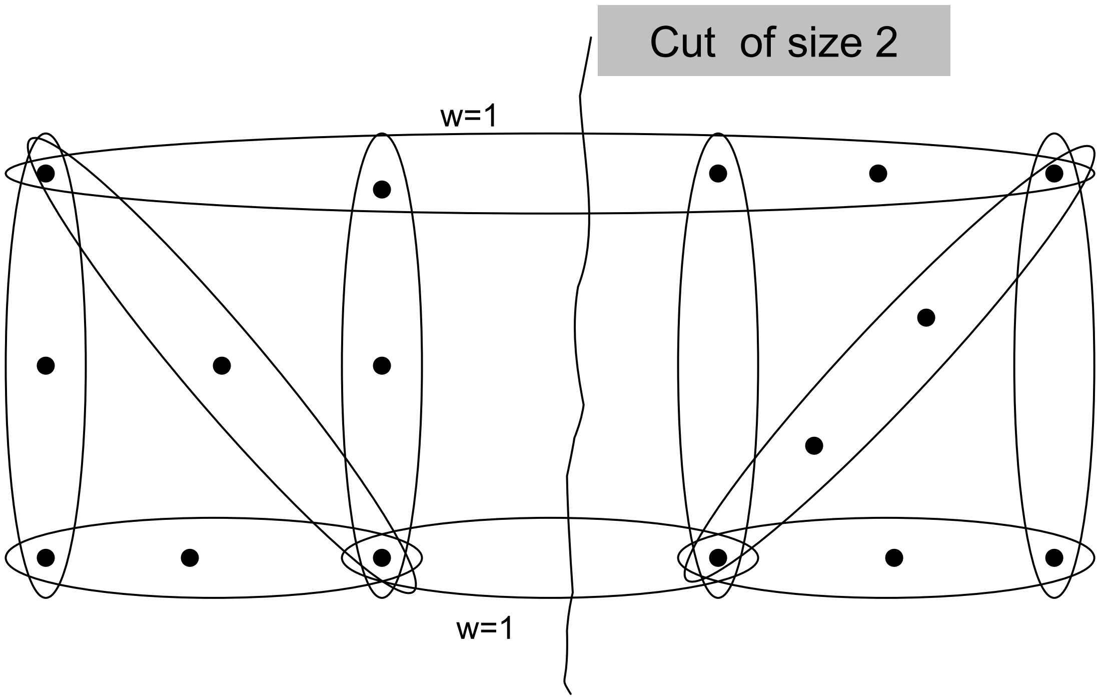
- Given a Hypergraph H(V,E)

  V -> set of vertices

  E -> set of hyperedges, where each hyperedge is a subset of the vertex set V

  Vertices and hyperedges are weighted
- Objective:
  - Find a partitions of set V in two (or k) disjoint subsets such that the number of vertices in each set V_i is bounded, and the function defined over hyperedges is optimized
  - Most commonly used objective is to minimize the sum of the weights of hyperedges connecting two ore more subsets

Hypergraph partitioning: min cut



- Hypergraph Partitioning with constraint about the number of vertices in each partition is NP-Complete problem

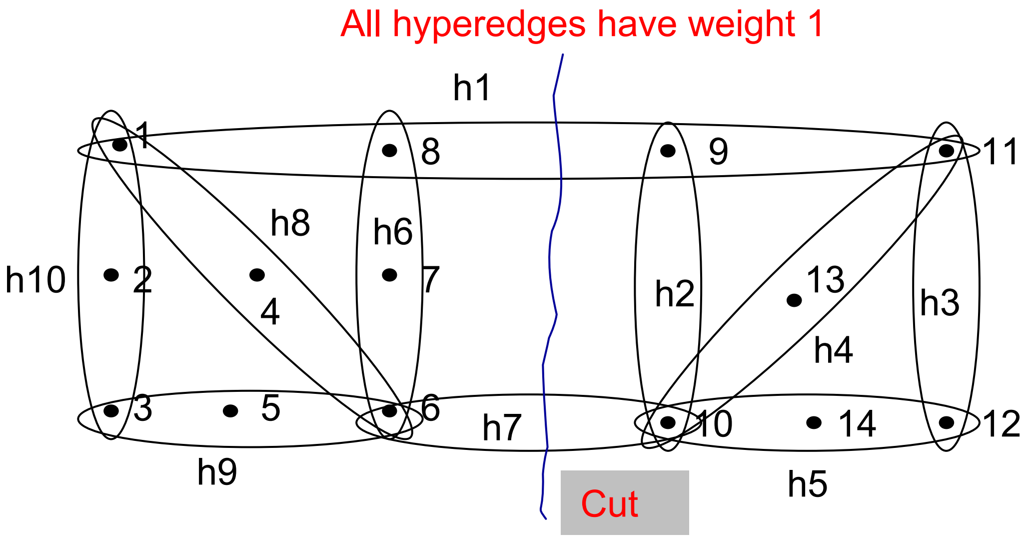
Hypergraph partitioning applications

- VLSI (circuit partitioning, …)
- Data-Mining
- …

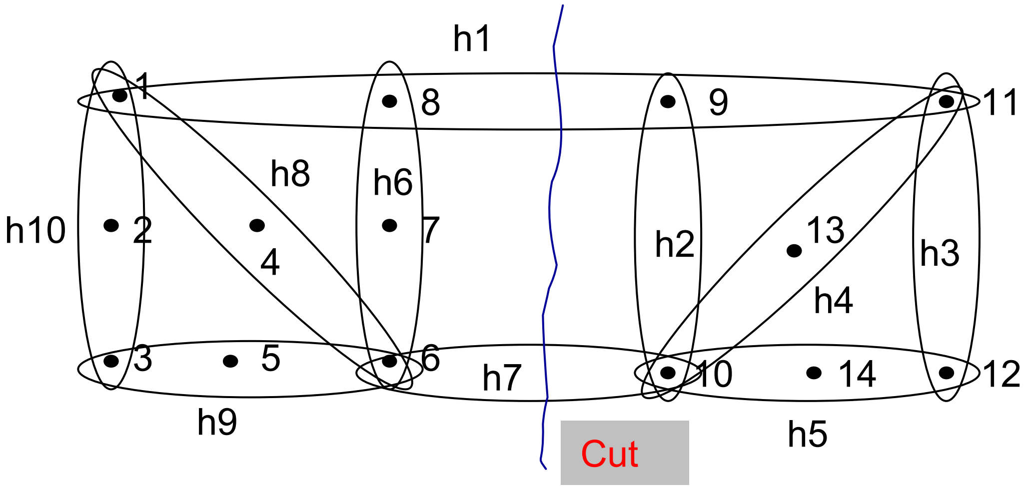
Generation of hypertree decomposition by hypergraph partitioning

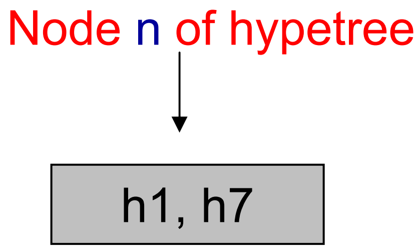
- Does recursive partitioning of hypergraph leads to “good” hypertree decomposition?
  - Every cut in hypergraph partitioning can be considered as a node in a hypertree decomposition
  - Nodes of hypertree are connected to the end of partitioning
    - Connectedness condition for variables should be ensured!

From partitioning to hypertree

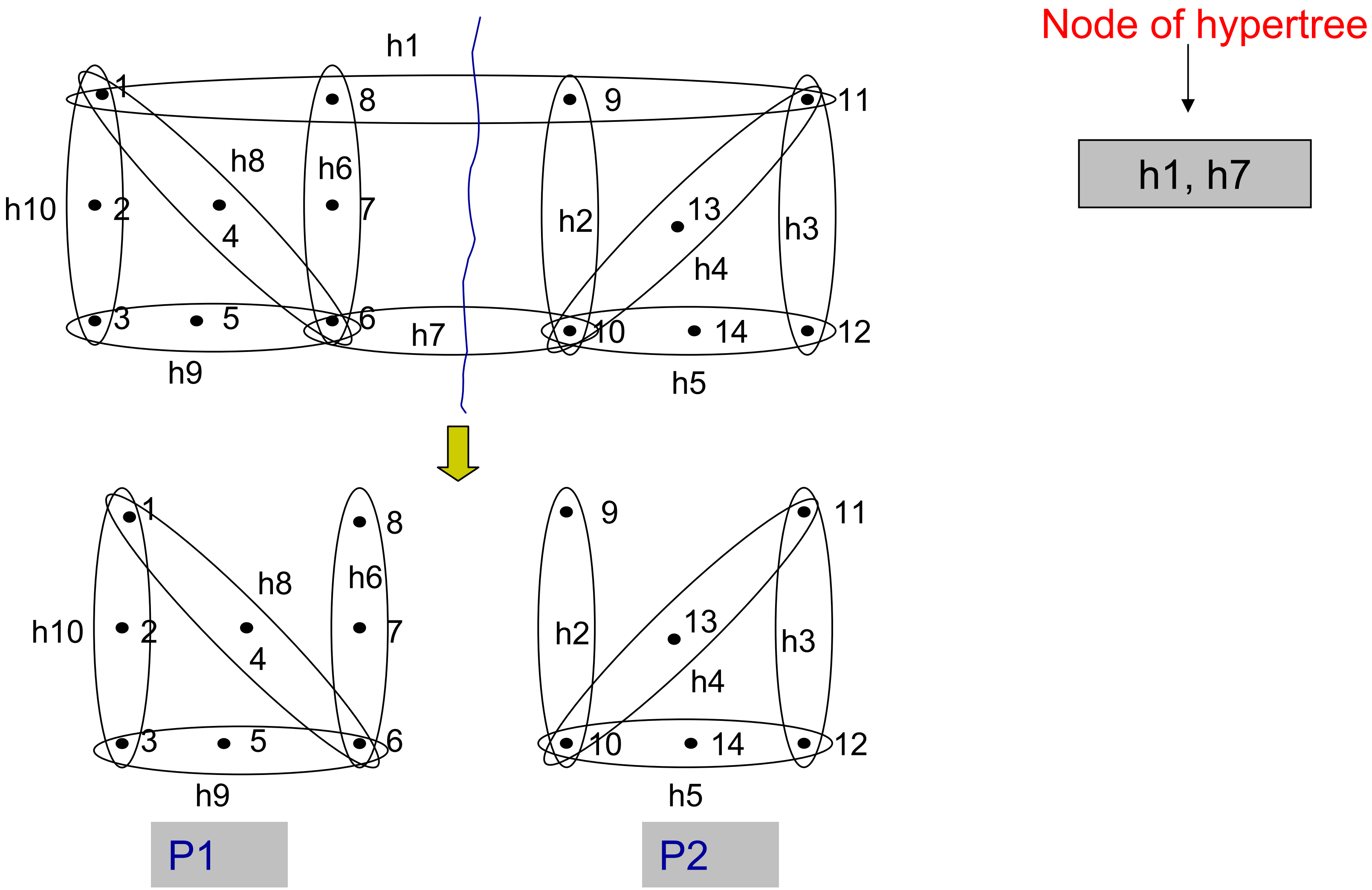


From partitioning to hypertree:

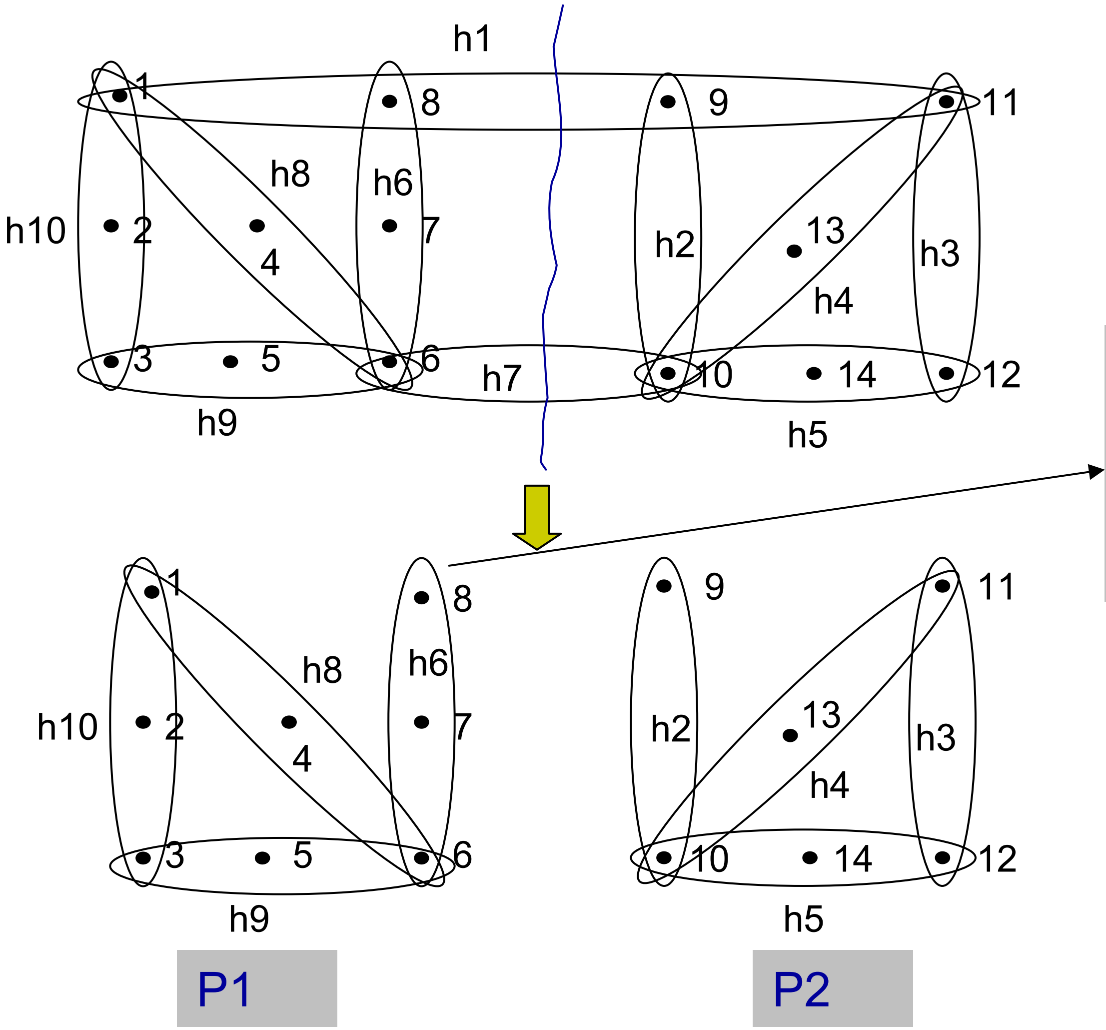




From partitioning to hypertree



From partitioning to hypertree

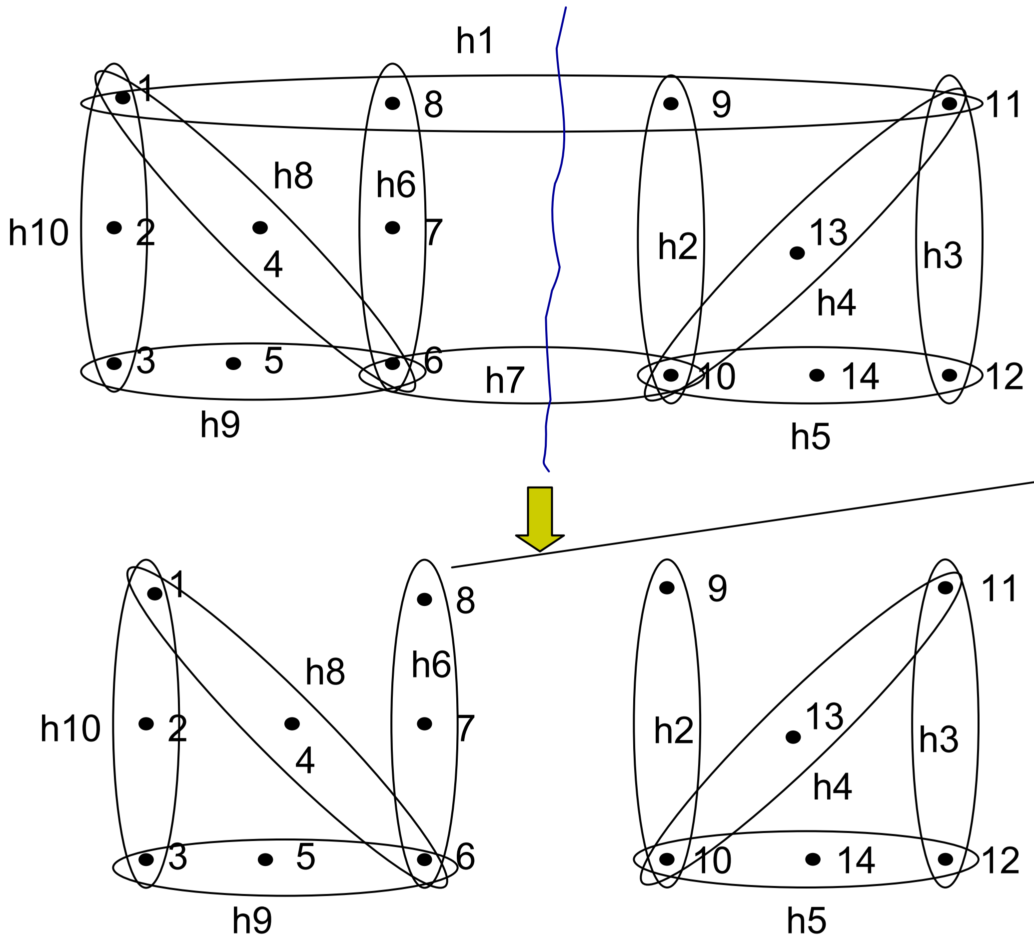


Node n of hypertree

h1, h7

To ensure the connectedness condition nodes 1,8,6 should appear together in some node s. To the end this node will be connected to node n above

From partitioning to hypertree



P1

P2

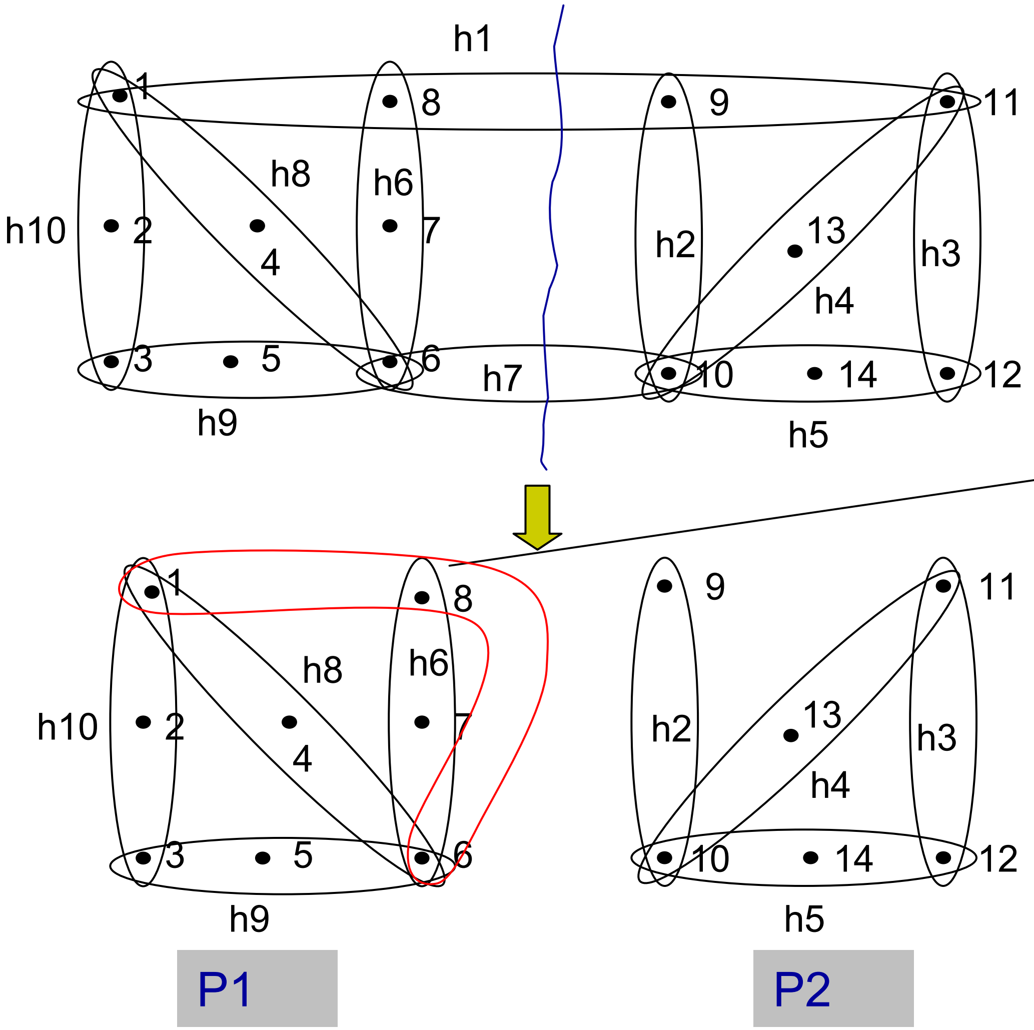
Node n of hypertree

h1, h7

To ensure the connectedness condition nodes 1,8,6 should appear together in some node k. To the end this node will be connected to node n above

Enforce this by introducing new hyperedge which contains all these nodes

From partitioning to hypertree



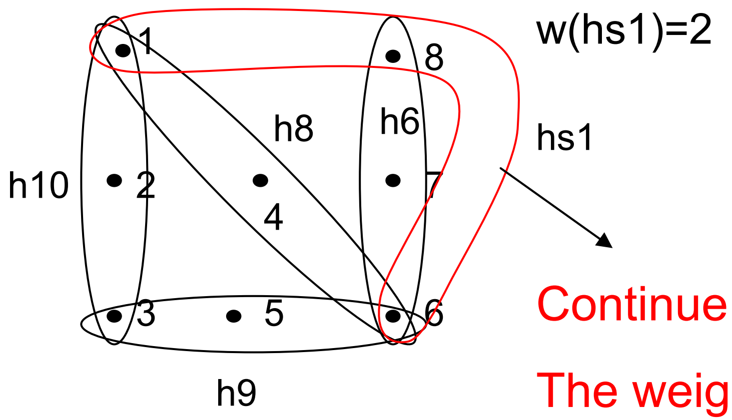
Node n of hypertree

h1, h7

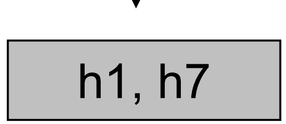
To ensure the connectedness condition nodes 1,8,6 should appear together in some node k. To the end this node will be connected to node n above

Enforce this by introducing new hyperedge which contains all these nodes

From partitioning to hypertree



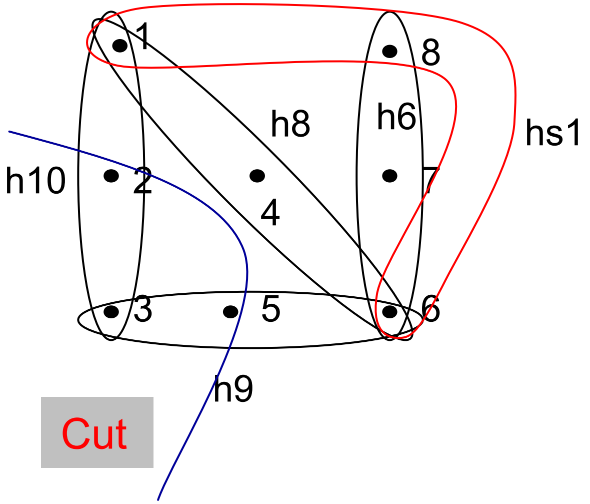
Node n of hypertree

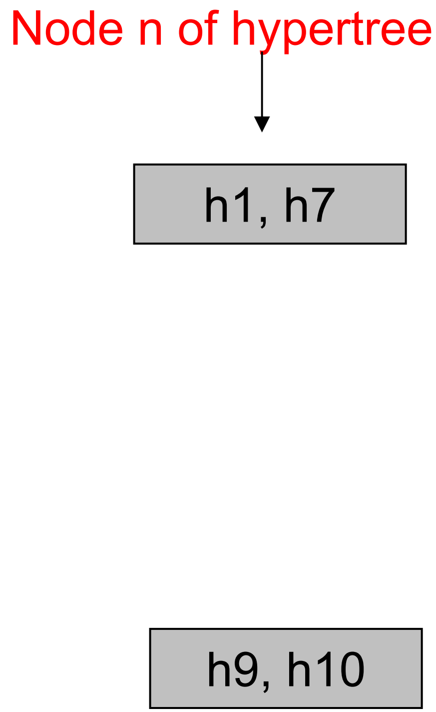


Continue recursively the partitioning

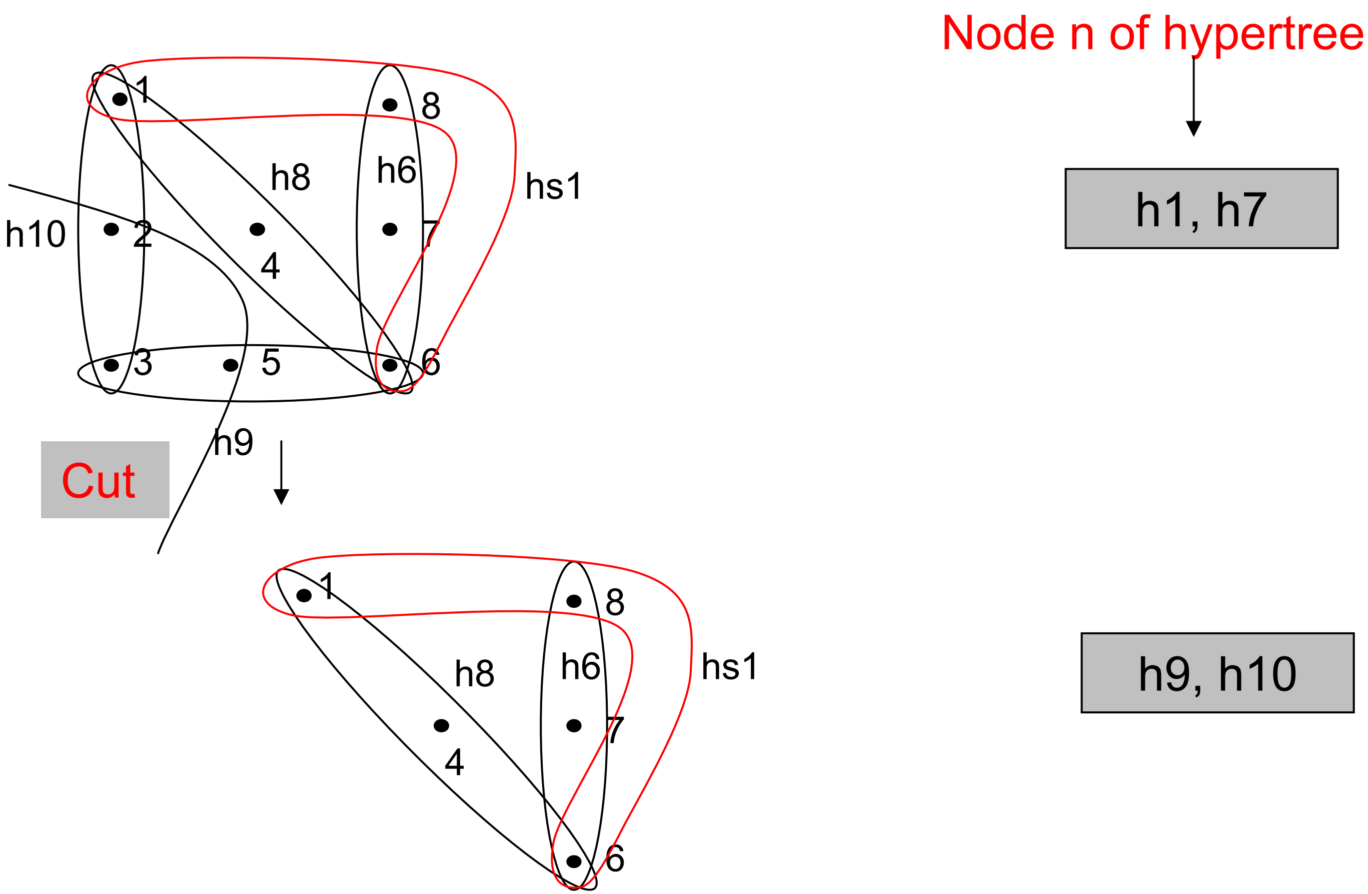
The weight of hyperedge hs1 is the sum of weights of hyperedges which cover this hyperedge: in this case w(hs1)=2

From partitioning to hypertree

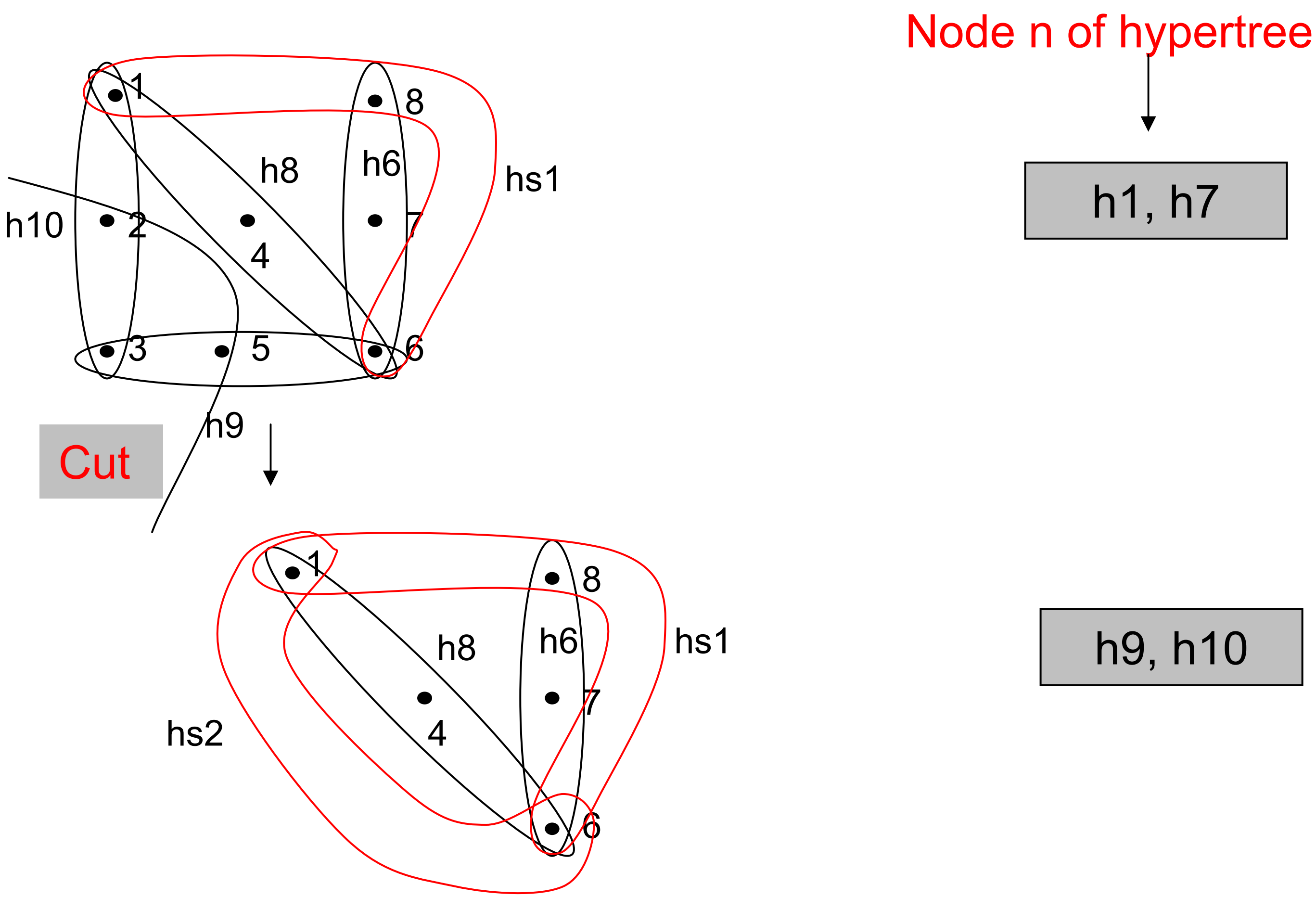




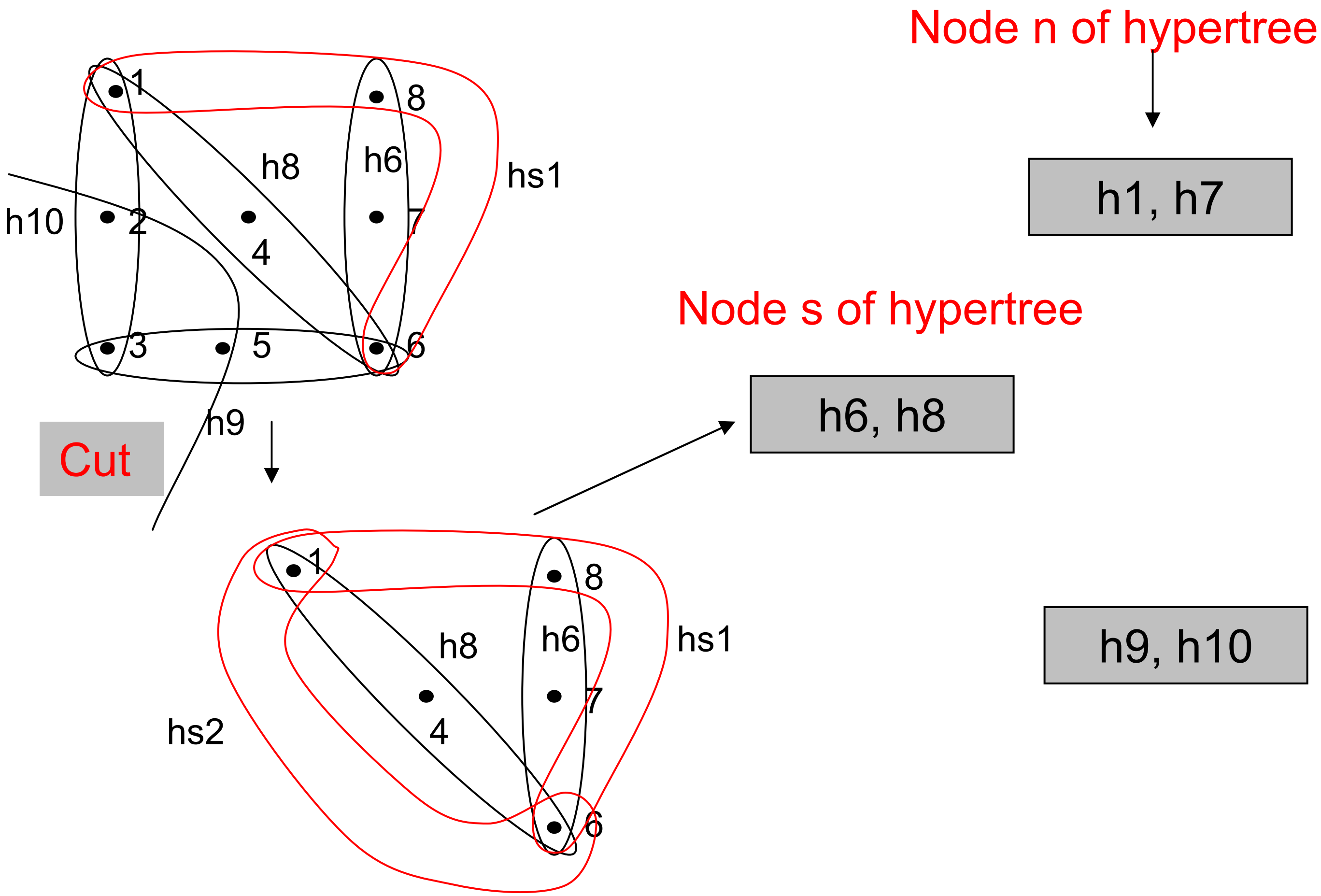
From partitioning to hypertree



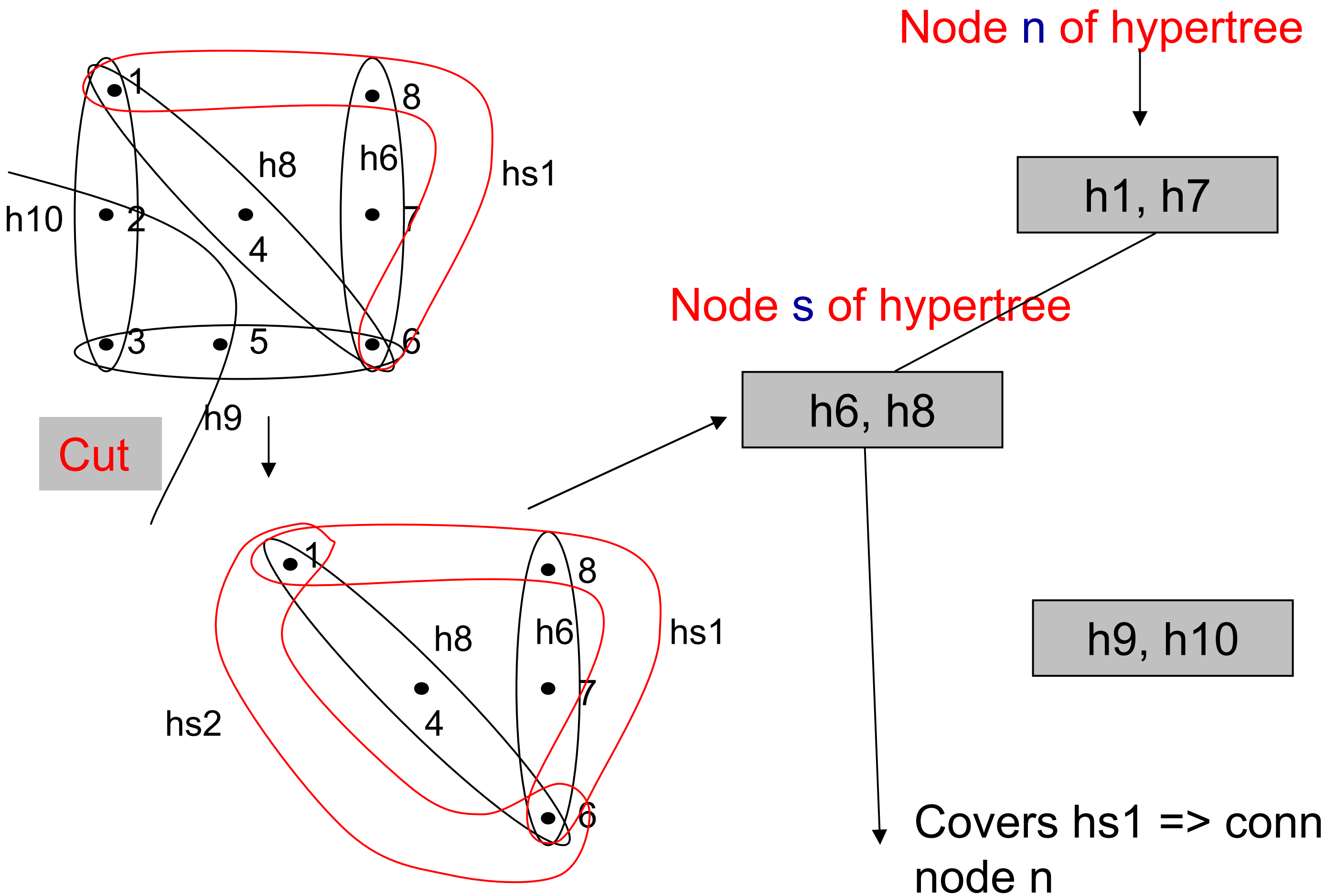
From partitioning to hypertree



From partitioning to hypertree

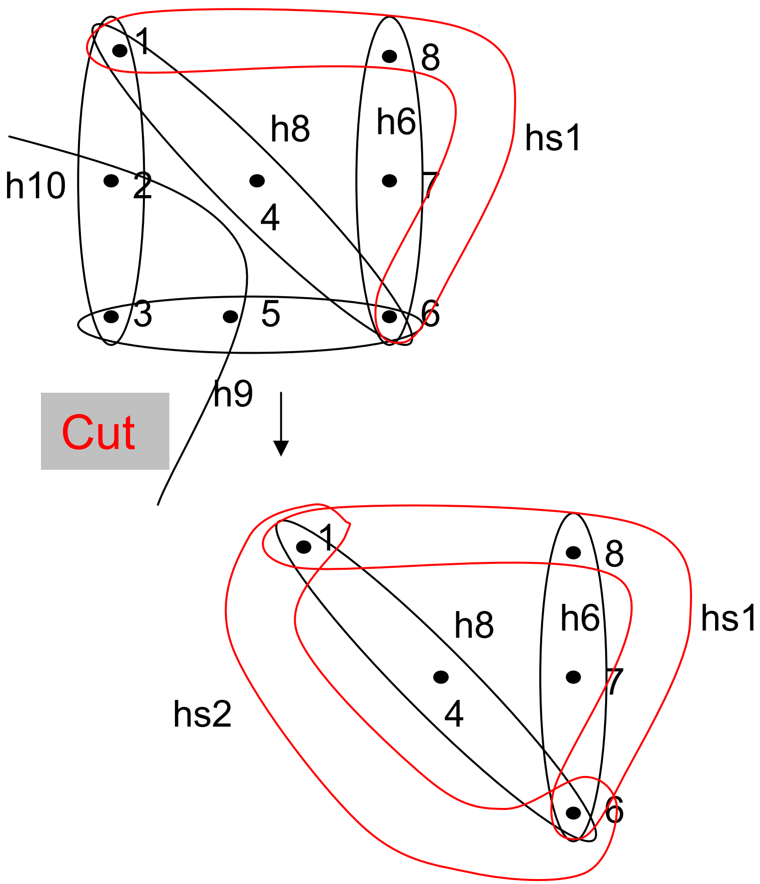


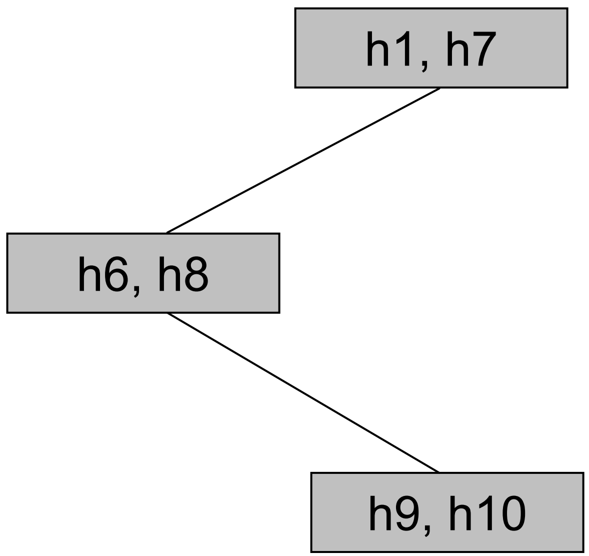
From partitioning to hypertree



Covers hs1 => connect to node n

From partitioning to hypertree





Hypergraph partitioning algorithms

- Mainly based on local search techniques
  - Kernighan-Lin heuristic
  - Fiduccia-Mattheyses (FM) heuristic
  - Simulated Annealing
  - …

Fiduccia-Mattheyses (FM) heuristic

- Start with two random halves (random split?)
- Repeat until no updates
  - Start with all vertices free
  - Repeat until no vertex free
    - Move vertex with largest gain (balance allows)
    - Update costs of neighbors
    - Lock vertex in place (record current cost)
  - Pick least cost point in previous sequence and use as next starting position
- Repeat for different random starting points

Partitioning algorithms used for hypertree decomposition

- Fiduccia-Mattheyses (FM) heuristic
- HMETIS package, which includes several implementation of heuristics

Instance (Vertices/Edges)	BE width	BE time	DBE width	DBE time	FM (W1) width	FM (W1) time	TS (W2) width	TS (W2) time	HM (W2) width	HM (W2) time	HM (best) width	HM (best) time
adder_15 (106/76)	**2**	0s	**2**	0s	**2**	0s	4	0s	**2**	3s	**2**	3s
adder_25 (176/126)	**2**	0s	**2**	0s	**2**	1s	4	0s	**2**	7s	**2**	6s
adder_50 (351/251)	**2**	0s	**2**	0s	**2**	6s	4	1s	**2**	13s	**2**	12s
adder_75 (526/376)	**2**	0s	**2**	0s	**2**	21s	5	2s	**2**	21s	**2**	19s
adder_99 (694/496)	**2**	0s	**2**	0s	**2**	53s	5	3s	**2**	28s	**2**	25s
bridge_15 (137/137)	**3**	0s	**3**	0s	8	1s	8	1s	4	7s	**3**	6s
bridge_25 (227/227)	**3**	0s	**3**	0s	13	1s	6	1s	4	11s	**3**	11s
bridge_50 (452/452)	**3**	0s	**3**	0s	29	5s	10	3s	4	24s	4	22s
bridge_75 (677/677)	**3**	0s	**3**	0s	44	10s	10	5s	4	39s	**3**	35s
bridge_99 (893/893)	**3**	0s	**3**	0s	64	18s	10	7s	4	48s	4	45s
NewSystem1 (142/84)	**3**	0s	**3**	0s	4	1s	6	1s	4	5s	**3**	5s
NewSystem2 (345/200)	**4**	0s	**4**	0s	9	2s	6	2s	**4**	14s	**4**	13s
NewSystem3 (474/278)	**5**	0s	**5**	0s	17	4s	11	4s	**5**	19s	**5**	18s
NewSystem4 (718/418)	**5**	0s	**5**	0s	22	8s	12	7s	**5**	31s	**5**	29s
atv_partial_system (125/88)	**3**	0s	4	0s	4	0s	5	1s	4	6s	4	6s
NASA (579/680)	**21**	0s	56	13s	56	20s	98	34s	33	90s	32	84s
c432 (196/160)	**9**	0s	**9**	0s	15	3s	24	4s	13	20s	12	19s
c499 (243/202)	**13**	0s	20	0s	18	3s	27	5s	18	30s	17	28s
c880 (443/383)	**19**	0s	25	0s	31	8s	41	7s	29	50s	25	46s
c1355 (587/546)	**13**	0s	22	0s	32	10s	55	14s	22	66s	22	61s
c1908 (913/880)	32	0s	33	1s	65	23s	70	26s	**29**	86s	**29**	77s
c2670 (1350/1193)	**33**	0s	35	1s	66	56s	78	46s	38	119s	38	106s
c3540 (1719/1669)	**63**	2s	73	11s	97	133s	129	104s	73	166s	73	149s
c5315 (2485/2307)	**44**	3s	61	24s	120	250s	157	157s	72	242s	68	214s
c6288 (2448/2416)	**41**	10s	45	77s	148	478s	329	245s	45	210s	45	186s
c7552 (3718/3512)	37	4s	**35**	8s	161	514s	188	351s	37	365s	37	309s
s27 (17/13)	**2**	0s	**2**	0s	**2**	0s	3	0s	**2**	0s	**2**	0s
s208 (115/104)	**7**	0s	**7**	0s	**7**	1s	11	1s	**7**	10s	**7**	9s
s298 (139/133)	**5**	0s	8	0s	7	1s	17	2s	7	11s	6	10s
s344 (184/175)	**7**	0s	8	0s	8	2s	12	2s	8	21s	**7**	19s

Comparison of hypertree decomposition algorithms:

A. Dermaku, T. Ganzow, G. Gottlob, B. McMahan, N. Musliu, M. Samer. *Heuristic Methods for Hypertree Decompositions. MICAI 2008: Lecture Notes in Artificial Intelligence, Volume 5317, pages 1-11, 2008, Springer.*

Instance (Vertices/Edges)	BE width	BE time	DBE width	DBE time	FM (W1) width	FM (W1) time	TS (W2) width	TS (W2) time	HM (W2) width	HM (W2) time	HM (best) width	HM (best) time
grid2d_10 (50/50)	**5**	0s	6	0s	**5**	0s	8	0s	**5**	3s	**5**	3s
grid2d_15 (113/112)	**9**	0s	**9**	0s	10	1s	12	1s	10	11s	10	10s
grid2d_20 (200/200)	12	0s	**11**	0s	15	2s	18	2s	14	29s	12	28s
grid2d_25 (313/312)	**15**	0s	**15**	0s	18	5s	26	5s	**15**	50s	**15**	43s
grid2d_30 (450/450)	19	0s	20	0s	21	11s	29	8s	**16**	70s	**16**	58s
grid2d_35 (613/612)	23	0s	23	0s	30	20s	41	13s	**19**	87s	**19**	73s
grid2d_40 (800/800)	26	0s	25	0s	28	38s	41	20s	**22**	108s	**22**	91s
grid2d_45 (1013/1012)	31	1s	30	1s	40	58s	47	31s	**25**	130s	**25**	109s
grid2d_50 (1250/1250)	33	1s	32	1s	44	88s	52	41s	**28**	154s	**28**	130s
grid2d_60 (1800/1800)	42	2s	39	3s	55	203s	75	75s	**34**	209s	**34**	178s
grid2d_70 (2450/2450)	49	4s	47	4s	65	347s	65	119s	**41**	283s	**41**	239s
grid2d_75 (2813/2812)	52	6s	50	7s	70	504s	99	158s	**44**	324s	**44**	274s
grid3d_4 (32/32)	**6**	0s	**6**	0s	**6**	0s	12	0s	**6**	1s	**6**	1s
grid3d_5 (63/62)	9	0s	10	0s	**8**	1s	18	1s	11	4s	10	3s
grid3d_6 (108/108)	14	0s	14	0s	**12**	1s	25	2s	15	9s	14	9s
grid3d_7 (172/171)	20	0s	20	0s	18	2s	33	5s	19	27s	**16**	24s
grid3d_8 (256/256)	25	0s	27	0s	25	5s	44	9s	21	48s	**20**	40s
grid3d_9 (365/364)	34	0s	26	0s	34	9s	56	14s	**24**	67s	**24**	56s
grid3d_10 (500/500)	42	1s	40	1s	41	20s	67	26s	**31**	93s	**31**	77s
grid3d_11 (666/665)	52	1s	53	2s	40	36s	83	43s	**37**	119s	**37**	99s
grid3d_12 (864/864)	63	3s	62	3s	53	61s	98	66s	45	150s	**44**	127s
grid3d_13 (1099/1098)	78	5s	68	6s	60	107s	122	101s	**53**	186s	**53**	158s
grid3d_14 (1372/1372)	88	10s	93	10s	86	161s	176	162s	**69**	230s	**69**	196s
grid3d_15 (1688/1687)	104	15s	103	15s	93	253s	151	245s	**76**	278s	**76**	244s
grid3d_16 (2048/2048)	120	24s	131	24s	100	400s	174	328s	87	339s	**82**	303s
grid4d_3 (41/40)	**8**	0s	**8**	0s	**8**	0s	20	0s	9	2s	**8**	2s
grid4d_4 (128/128)	**17**	0s	18	0s	**17**	1s	40	3s	19	13s	18	12s
grid4d_5 (313/312)	35	0s	37	0s	32	8s	78	17s	**28**	58s	**28**	48s
grid4d_6 (648/648)	64	3s	71	2s	58	40s	140	67s	**47**	123s	**47**	106s
grid4d_7 (1201/1200)	109	14s	110	14s	89	134s	182	194s	74	229s	**71**	208s
grid4d_8 (2048/2048)	164	62s	166	62s	120	441s	310	581s	**107**	408s	**107**	393s
grid5d_3 (122/121)	**18**	0s	20	0s	**18**	1s	49	4s	20	11s	19	10s

Which algorithm to use ..

- Tree Decomposition:
  - If the width is not critical then use MCS, or Min-Fill
  - For exact solutions and small examples use branch and bound developed by Gogate and Dechter or A* algorithms
  - For larger examples and better width than MCS and Min-Fill, use Iterated local search or GA
- Hypertree Decomposition
  - MCS, Min-Fill
  - Hypergraph Partitioning for too large examples