A Constraint Store Based on Multivalued Decision Diagrams

Henrik Reif Andersen
Tarik Hadzic
Peter Tiedemann
IT University of Copenhagen

John Hooker
Carnegie Mellon University

INFORMS 2007

A strength and a weakness of CP

- **Strength** of CP – processes individual constraints.
  - Exploits problem substructure.
- **Weakness** of CP – processes individual constraints.
  - No global point of view.
  - Overlooks implications of combined constraints.
- Two compensating strategies:
  - **Global constraints**
  - **Domain store relaxation**

The constraint store

- Provides a **global** point of view.
  - …to some extent.
  - **Pools** results of individual constraint processing.
  - Basis for **constraint propagation**.
- In current practice, constraint store = **domain store**.

Advantages of domain store

- Provides **natural input** into filtering algorithms.
  - Filters start with current domains when processing a constraint.
- Guides **branching** (on variables) in a natural way.
  - Simply split the domain in the current domain store.

Disadvantages of domain store

- Transmits relatively **little information**.
- A **weak relaxation** of the problem.
  - Ignores interactions between variables.
  - Feasible set is simply a Cartesian product of domains.
- **Unbalanced tradeoff.**
  - Search trees **too large**.
  - **Too little processing** at nodes.

A richer constraint store…

- Should provide a much **stronger relaxation** than domain store.
  - Strength of relaxation should be **adjustable**, ranging from domain store to original problem.
- Should **guide branching** in a natural way.
- Should readily provide a **bound on optimal value**.
  - To allow branch-and-bound search in optimization problems.
- Should provide a **natural input** into constraint processing (“filtering”) algorithms.

Proposal – Relaxed MDD

- Relaxed **multivalued decision diagram** can serve as a constraint store.
  - Generalization of **binary decision diagram**, used in circuit verification, configuration problems, etc.
- An MDD is a **compact representation** of a search tree for the problem.
  - **Isomorphic subtrees** are merged.
  - MDD is **relaxed** by limiting its width.

Proposal – Relaxed MDD

- Satisfies **at least 3 of the 4 criteria** for a richer constraint store…

Proposal – Relaxed MDD

- *Should provide a much **stronger relaxation** than domain store.*
  - *Strength of relaxation should be **adjustable**, ranging from domain store to original problem.*

- By setting the maximum MDD width, we obtain a relaxation of **arbitrary strength**.
  - Ranging from domain store (width = 1) to original problem (no limit on width).

Proposal – Relaxed MDD

- *Should **guide branching** in a natural way.*

- Can easily infer a **reduced variable domain** on which to branch at any point in the search tree.

Proposal – Relaxed MDD

- *Should readily provide a **bound on optimal value**.*
  - *To allow branch-and-bound search in optimization problems.*

- The cost of a **shortest path** in the relaxed MDD is a bound on the optimal value.

Proposal – Relaxed MDD

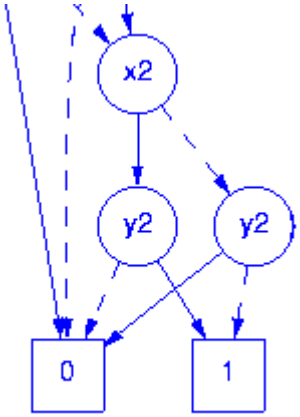
- *Should provide a **natural input** into constraint processing ("filtering") algorithms.*

- This is a **primary research issue**.
- We must **gradually** develop MDD-based "filtering" algorithms for global constraints.
  - As was done for domain filters.
  - "Filtering" algorithms **restructure** the relaxed MDD.

Proposal – Relaxed MDD

- *Should provide a **natural input** into constraint processing ("filtering") algorithms.*

- We developed and tested processors for **alldiff** and **integer knapsack** (inequality) constraints.

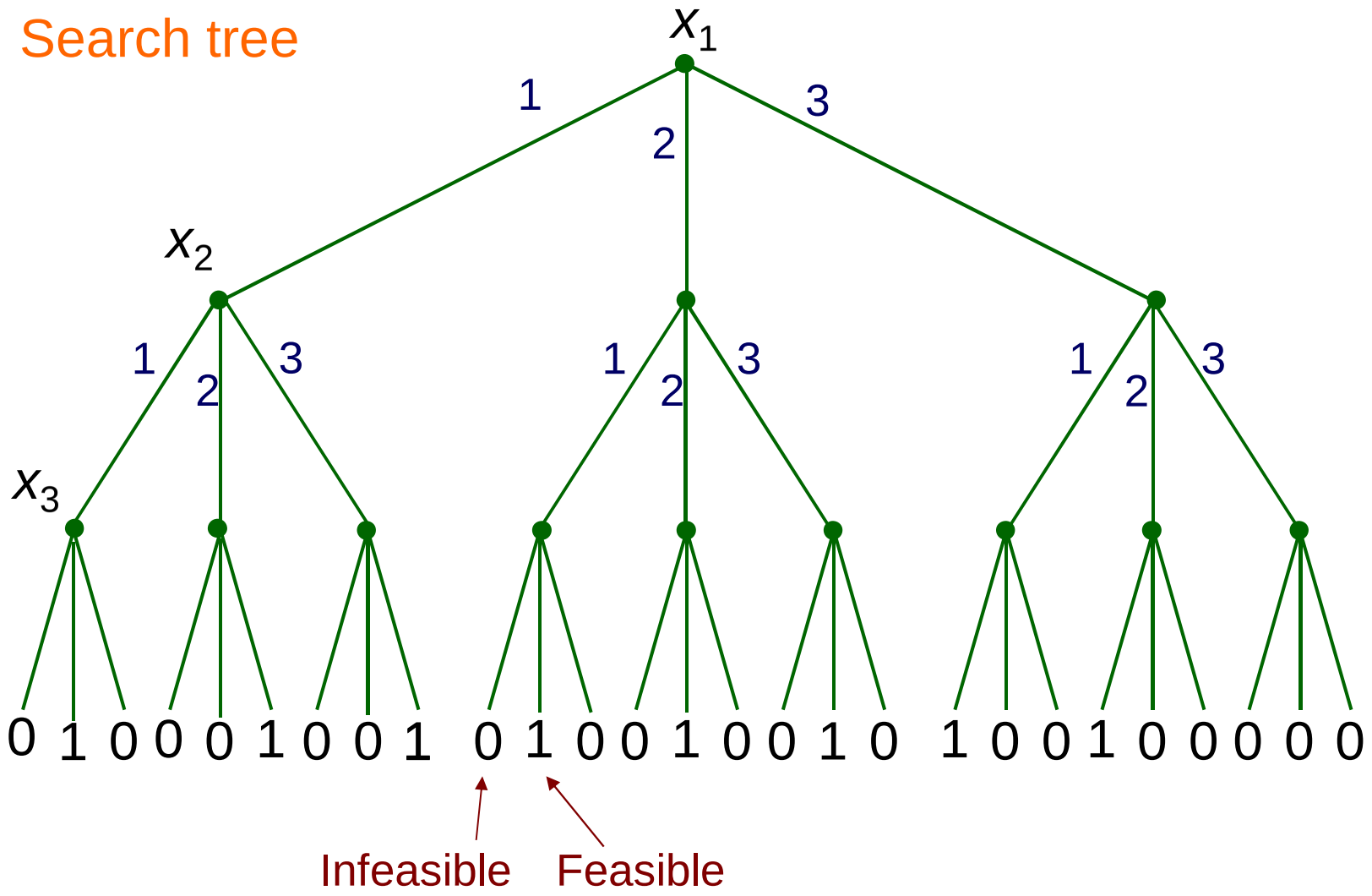


MDDs and how to relax them

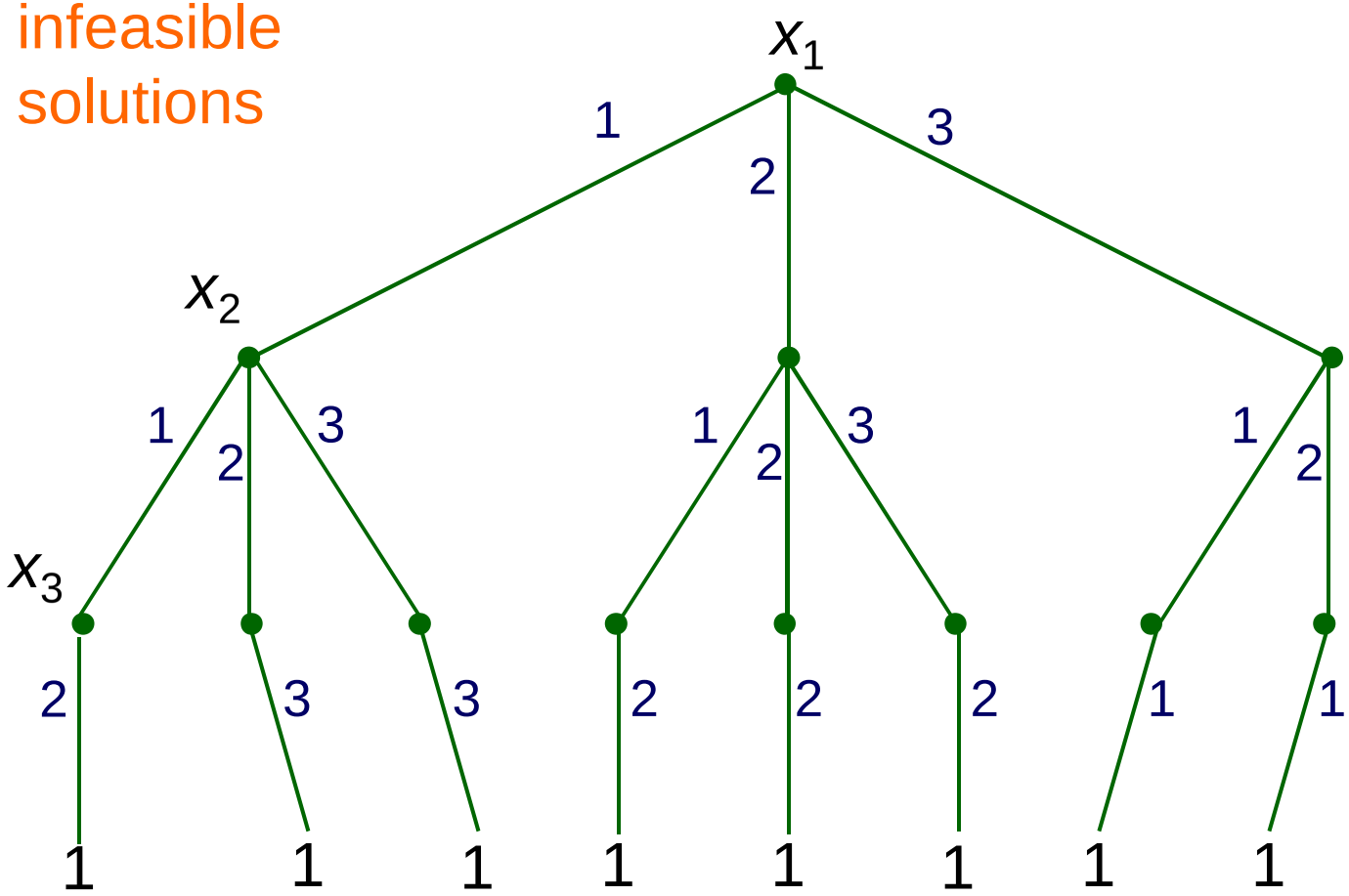
Example problem

$$\begin{pmatrix} x_1 + x_3 = 4 \\ 3 \le x_1 + x_2 \le 5 \end{pmatrix} \vee \; (x_1, x_2, x_3) = (1,1,2)$$

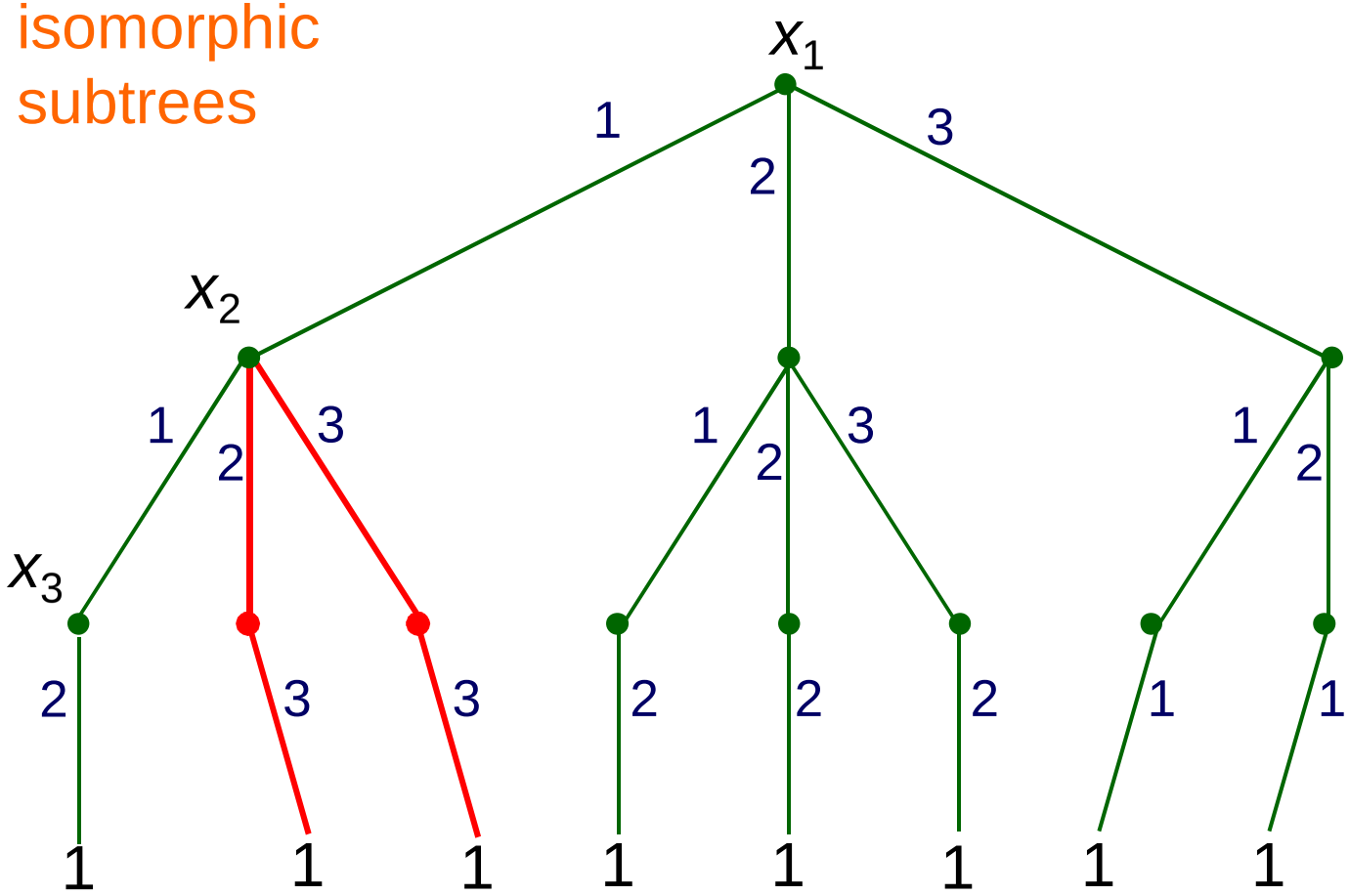
Search tree


x_1
1
2
3
x_2
1
2
3
1
2
3
1
2
3
x_3
0 1 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0
Infeasible
Feasible

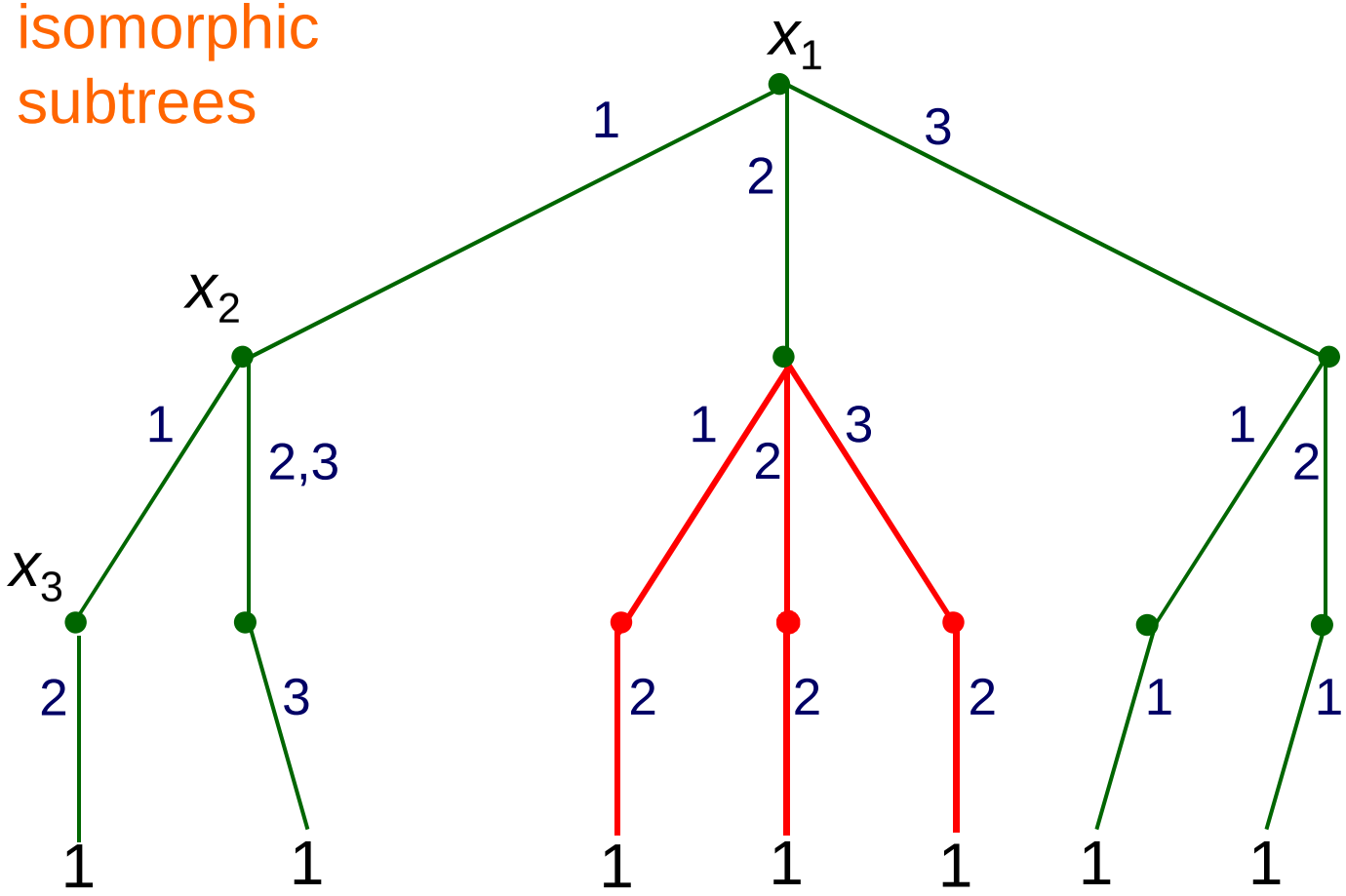
Remove infeasible solutions



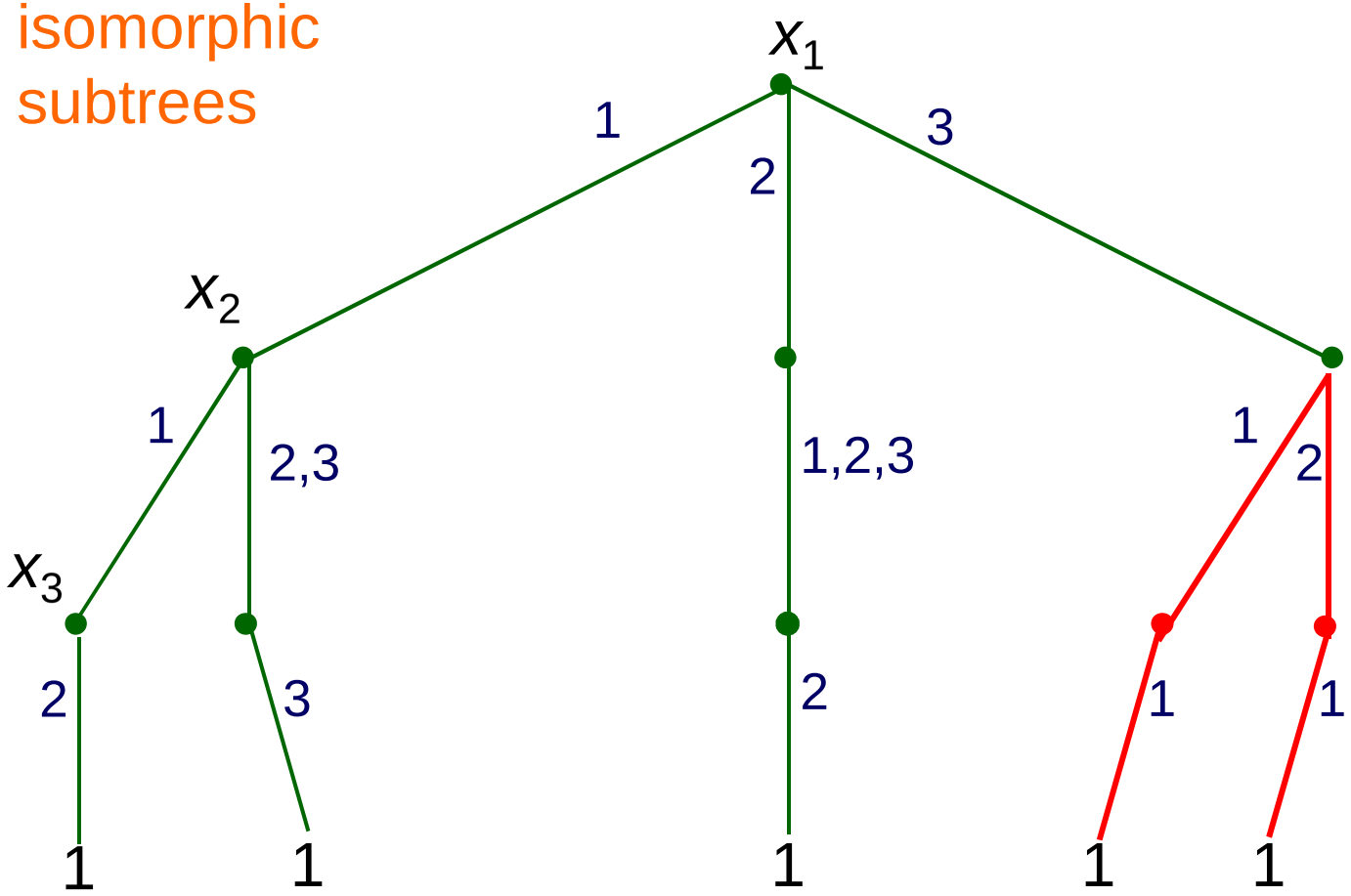
Merge isomorphic subtrees



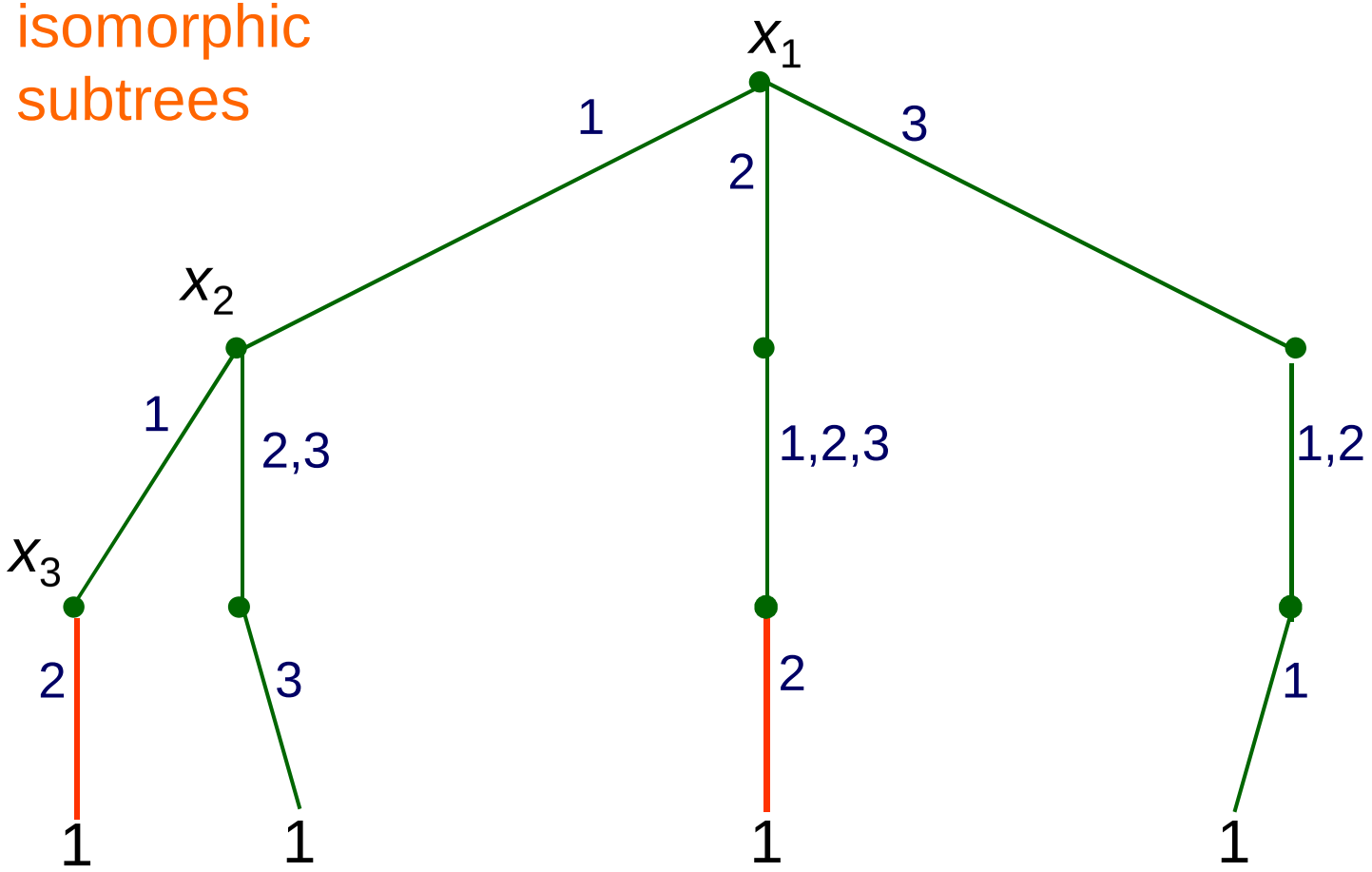
Merge isomorphic subtrees



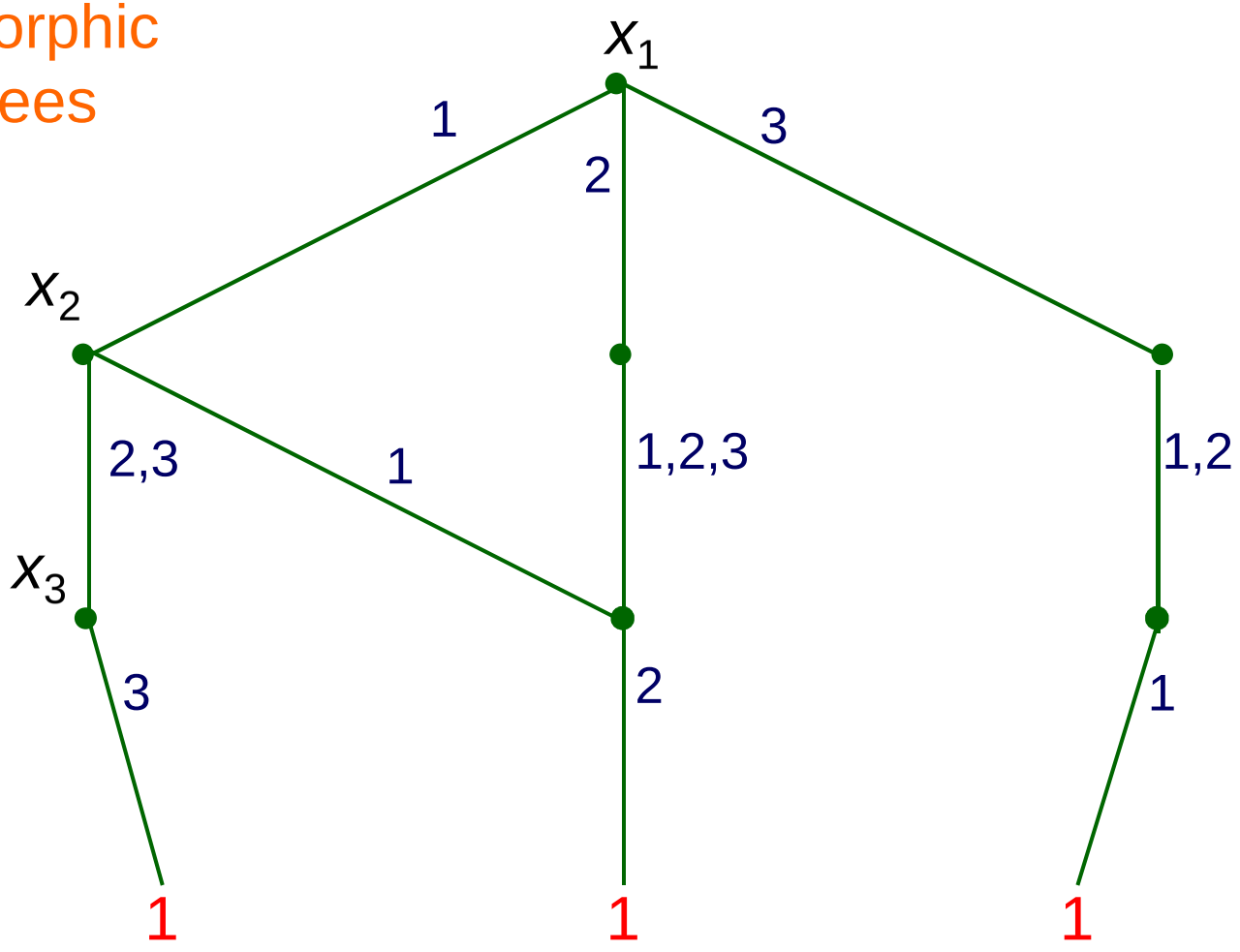
Merge isomorphic subtrees



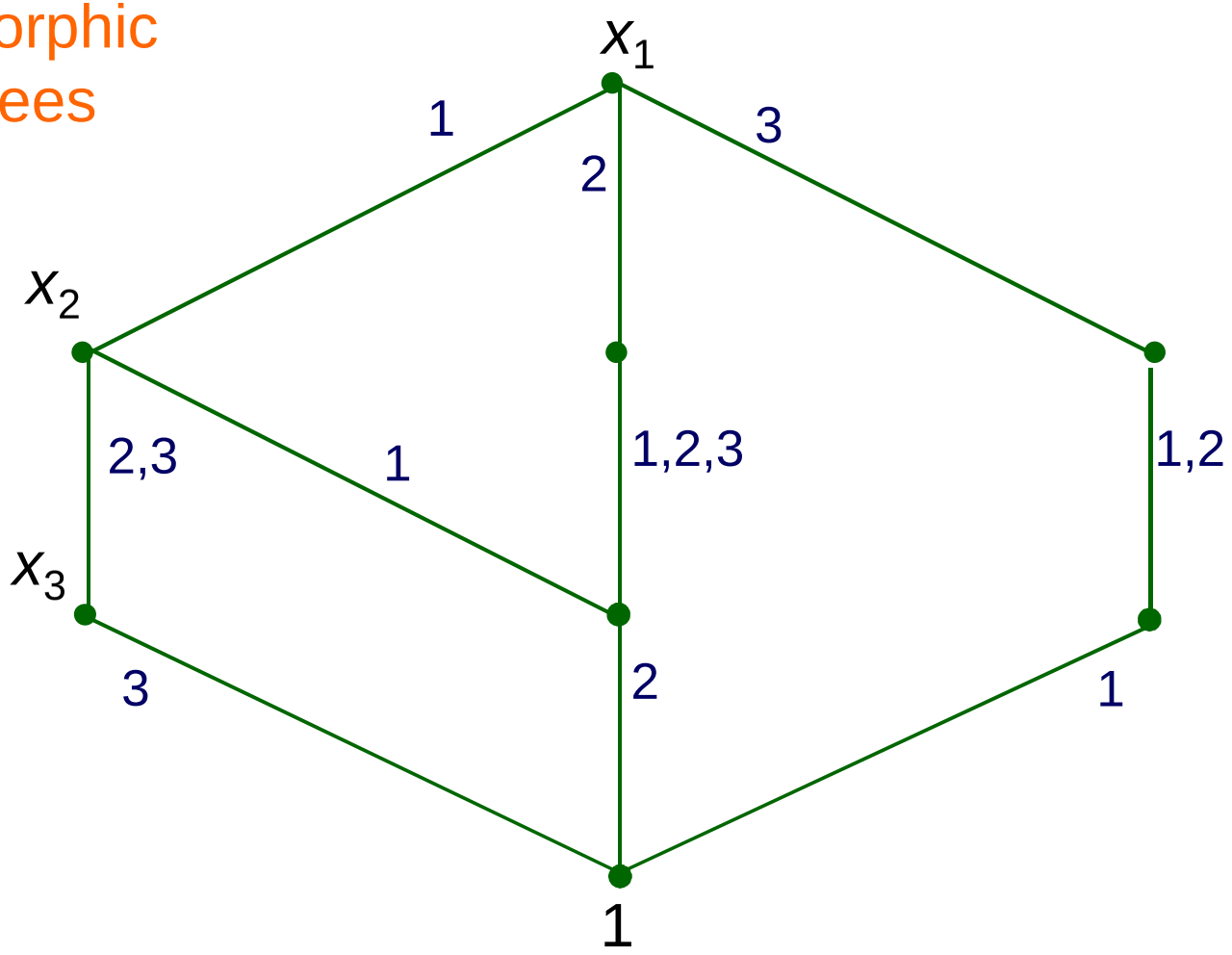
Merge isomorphic subtrees



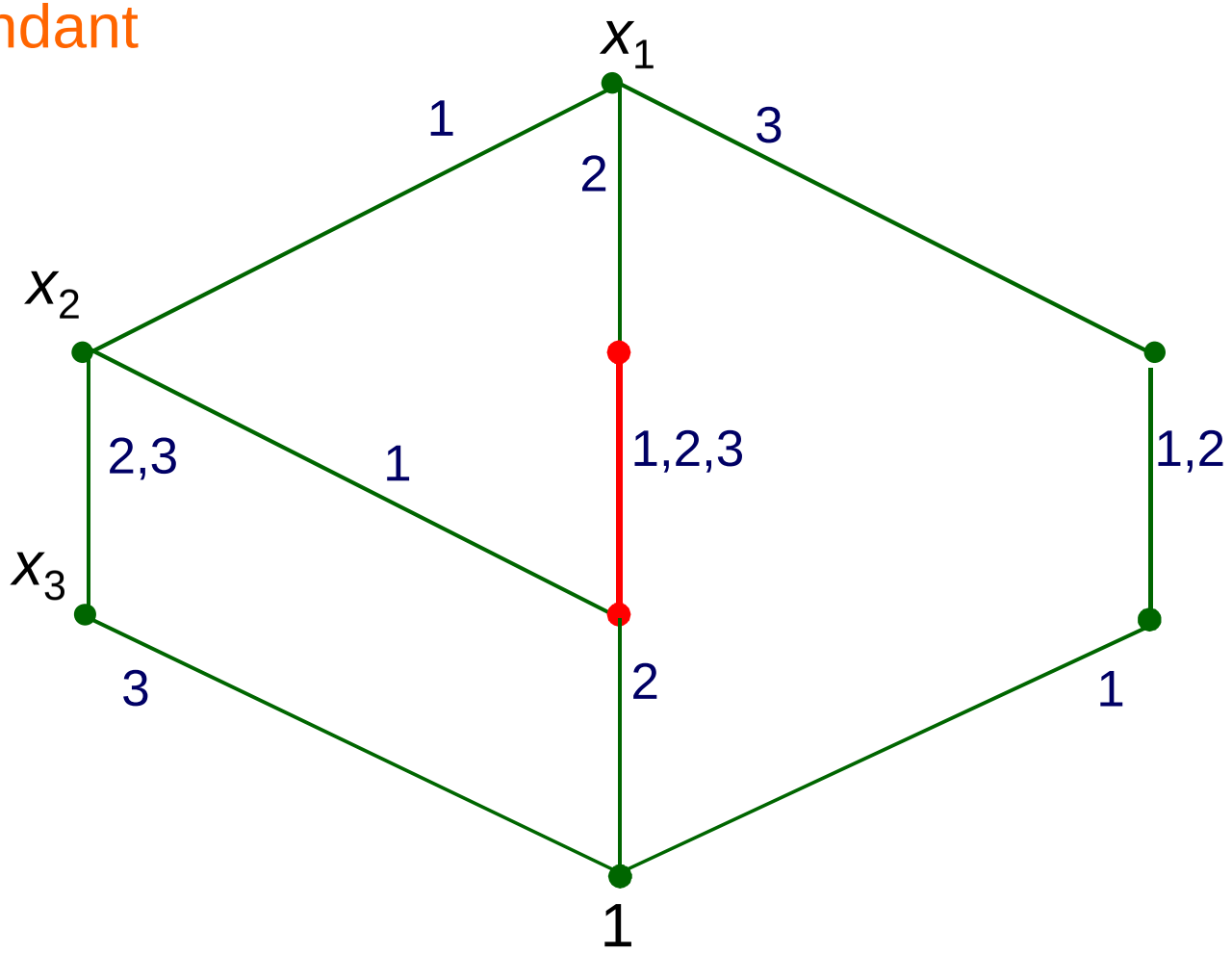
Merge isomorphic subtrees



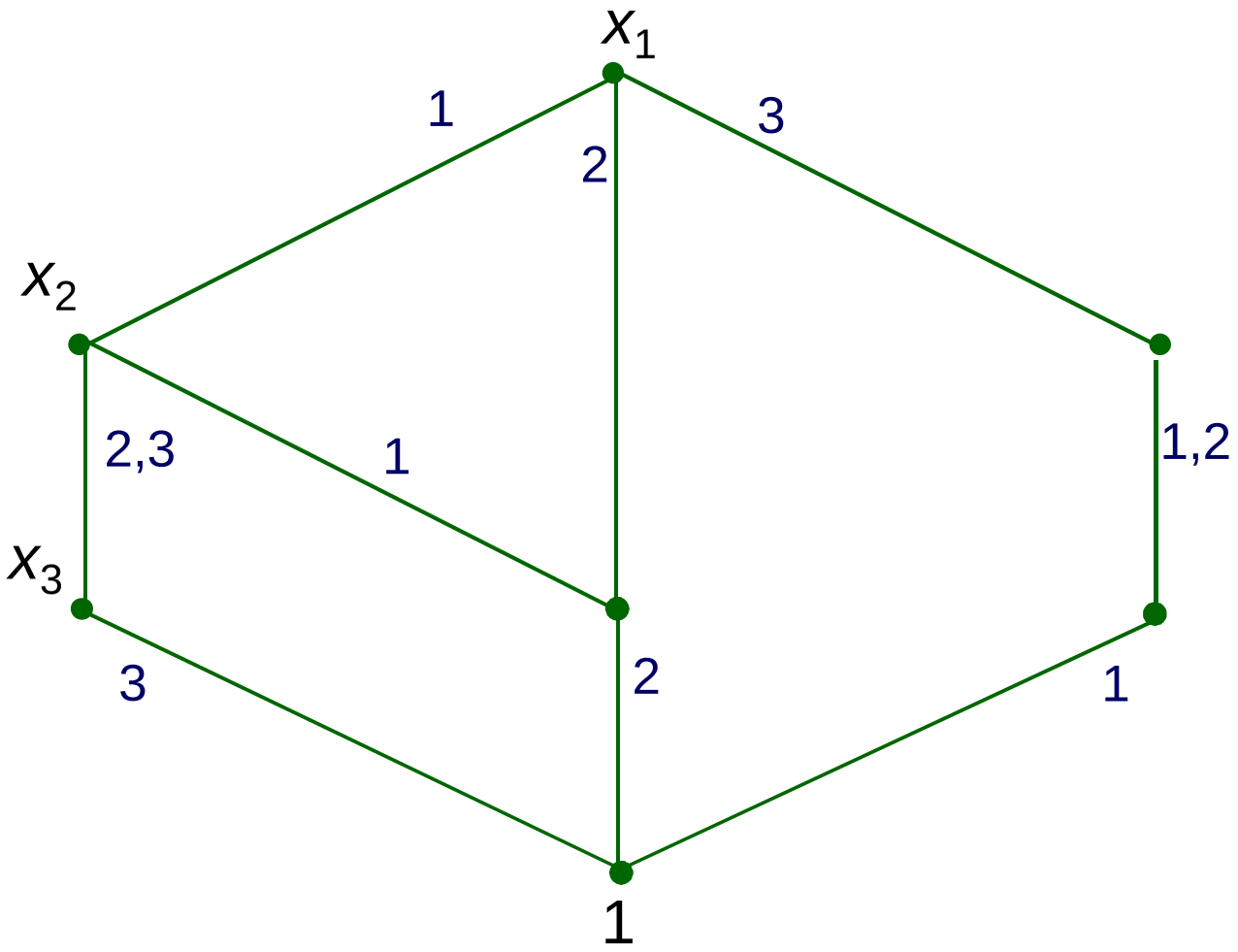
Merge isomorphic subtrees



Remove redundant edge

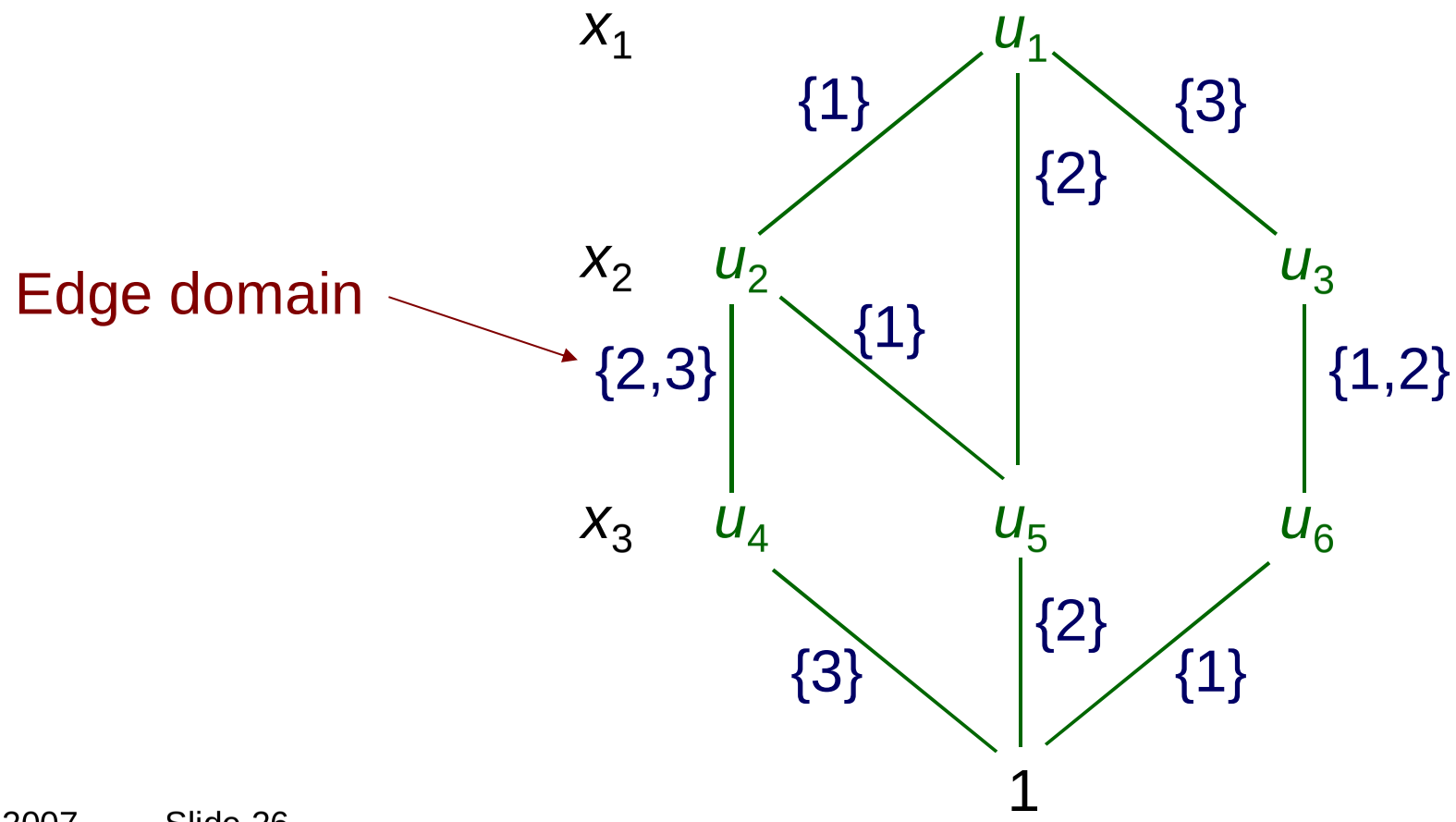


Reduced MDD



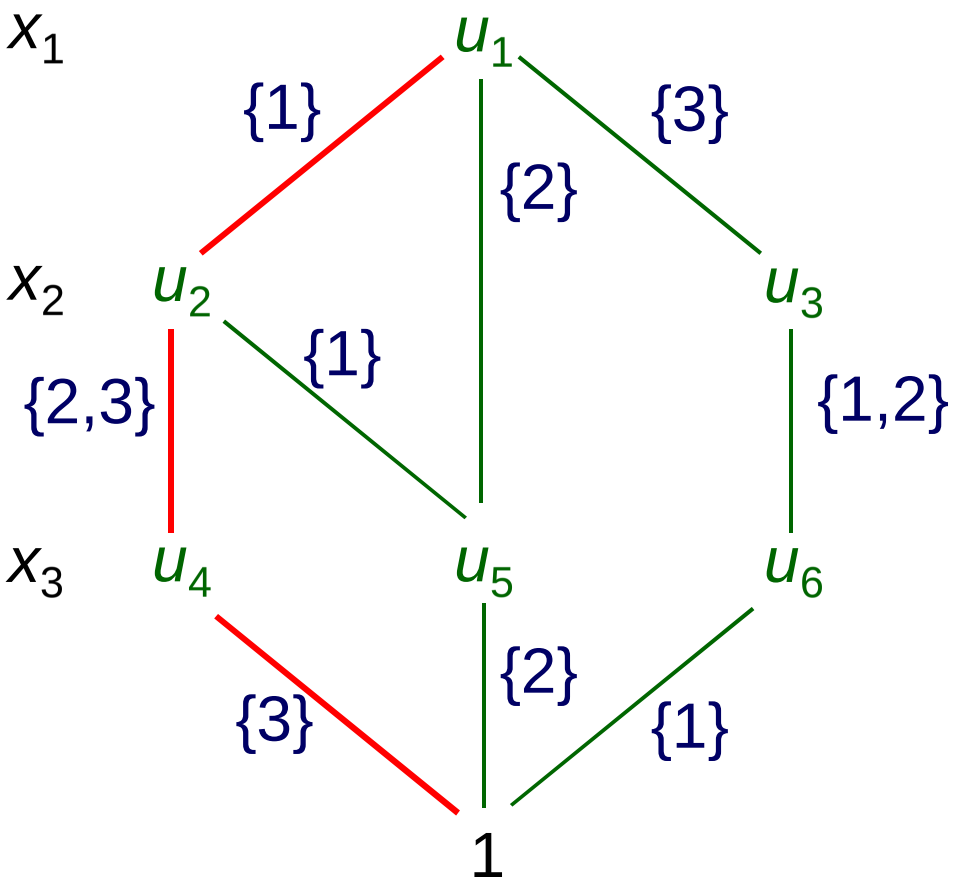
Example problem

$$\begin{pmatrix} x_1 + x_3 = 4 \\ 3 \le x_1 + x_2 \le 5 \end{pmatrix} \vee \; (x_1, x_2, x_3) = (1,1,2)$$



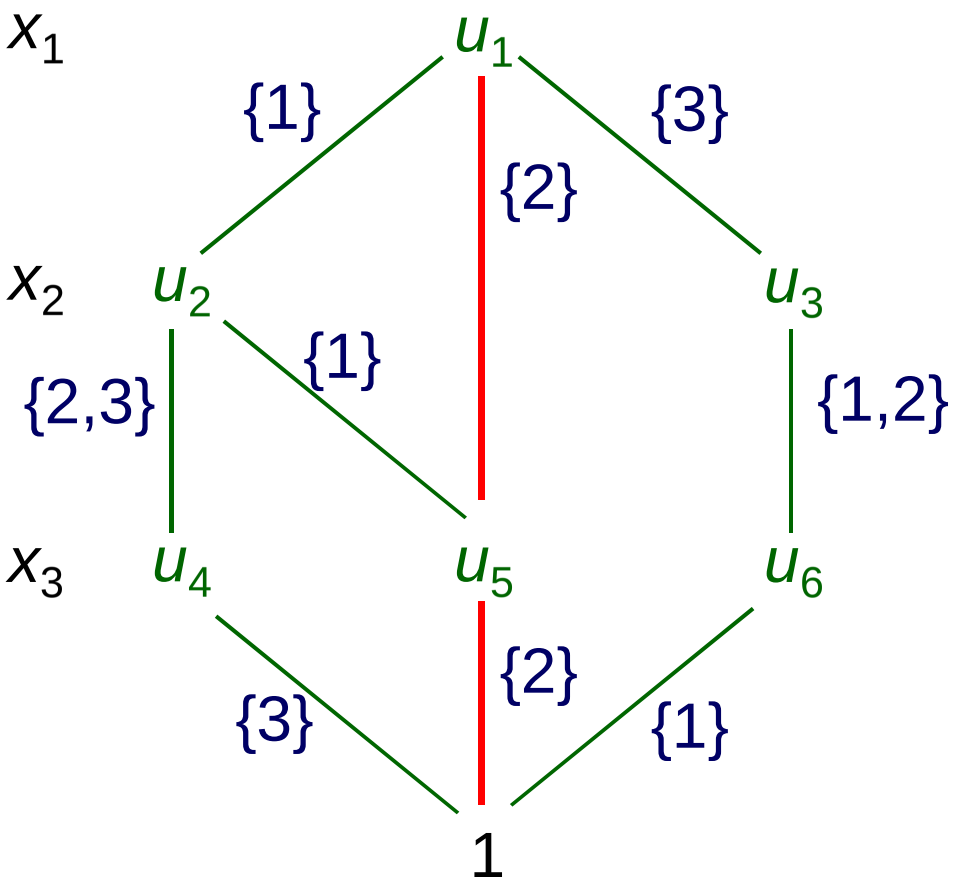
Each path represents a Cartesian product of solutions.

{1}×{2,3}×{3}



Each path represents a Cartesian product of solutions.

{2}×{1,2,3}×{2}



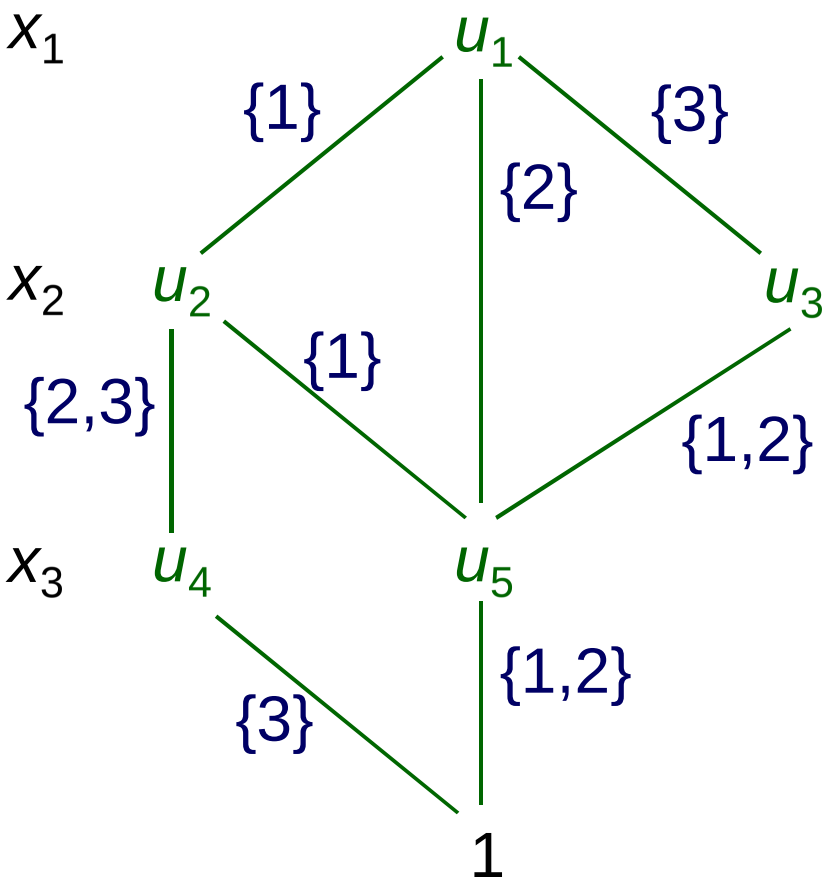
A relaxed MDD of width 2.

At most 2 nodes on each level.

Full MDD:
8 solutions

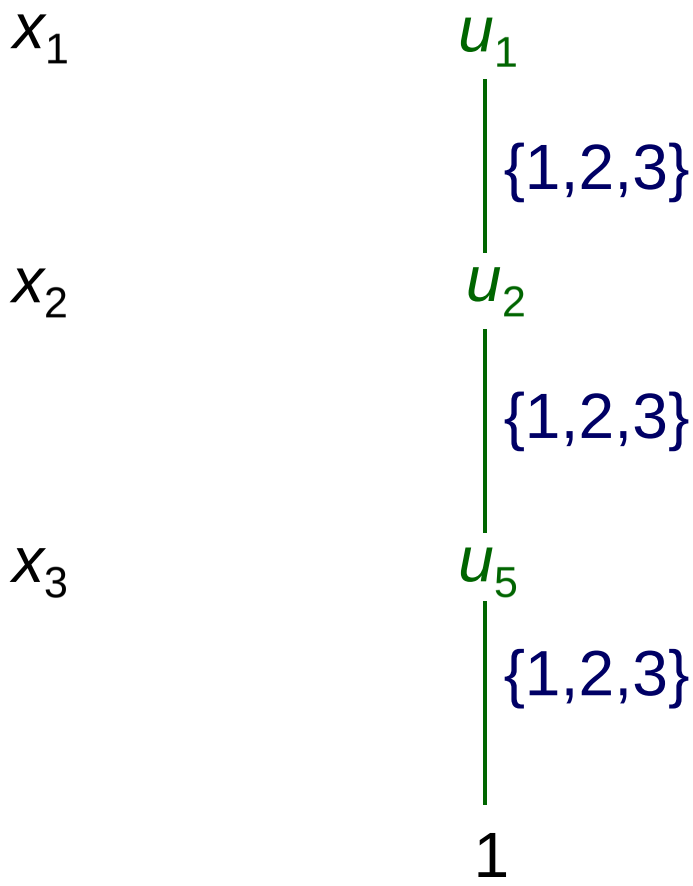
Relaxed MDD:
14 solutions

Domain store:
3×3×3 = 27 solutions



Relaxed MDD of width 1.

= Domain store



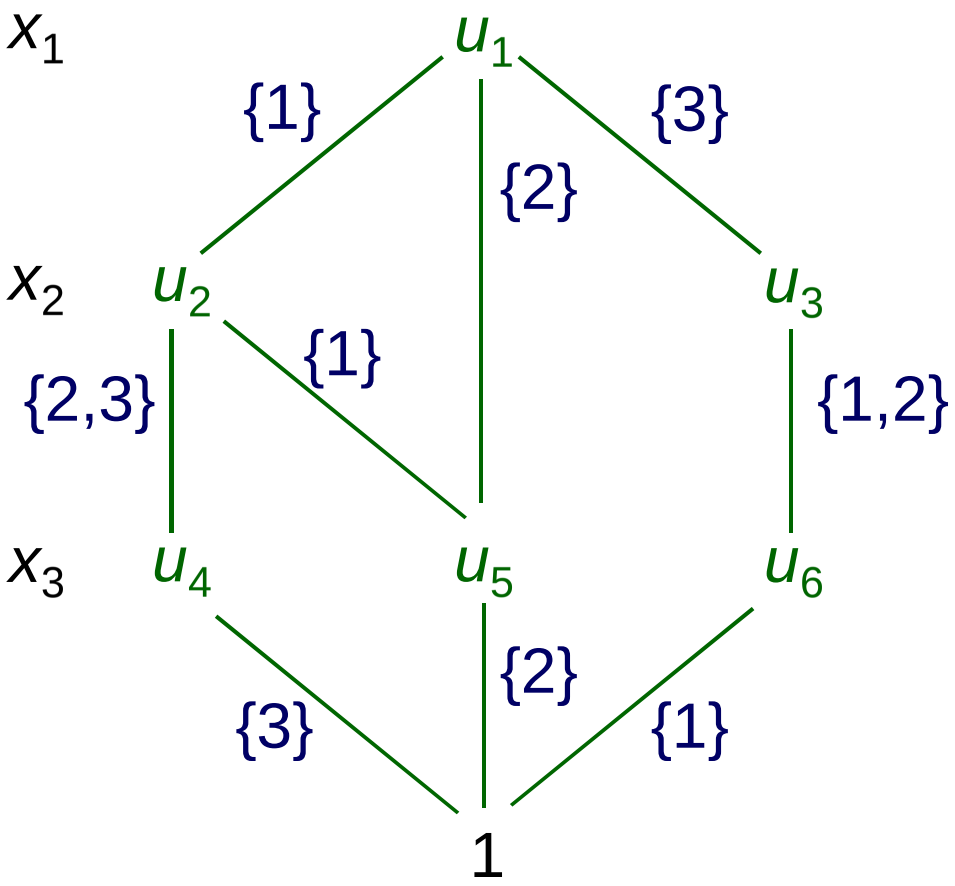
MDD consistency

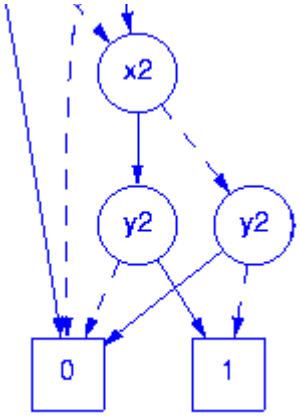
- A relaxed MDD and a constraint achieve **MDD consistency** if the edge domains cannot be further reduced without losing solutions of the constraint.
  - Reduces to GAC when the MDD has width 1.
- MDD consistency is **harder** to maintain than GAC.
  - Achieving MDD consistency for a constraint on an MDD of polynomial size can be **NP-hard**, even when achieving GAC for the constraint is **polynomial**.
  - Hamiltonian path problem can be reduced to achieving MDD consistency on polynomial-size MDD for *n*-walk constraint on a graph.

MDD consistency

$$\begin{pmatrix} x_1 + x_3 = 4 \\ 3 \le x_1 + x_2 \le 5 \end{pmatrix} \vee \; (x_1, x_2, x_3) = (1,1,2)$$

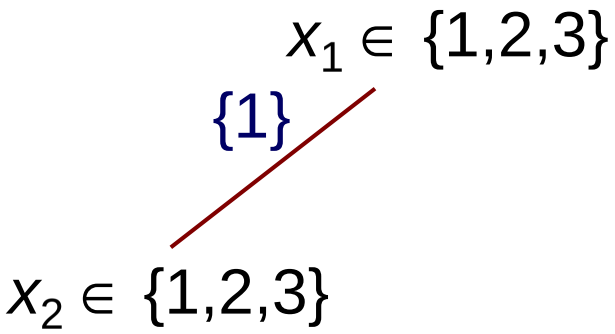
This constraint and MDD achieve MDD consistency.

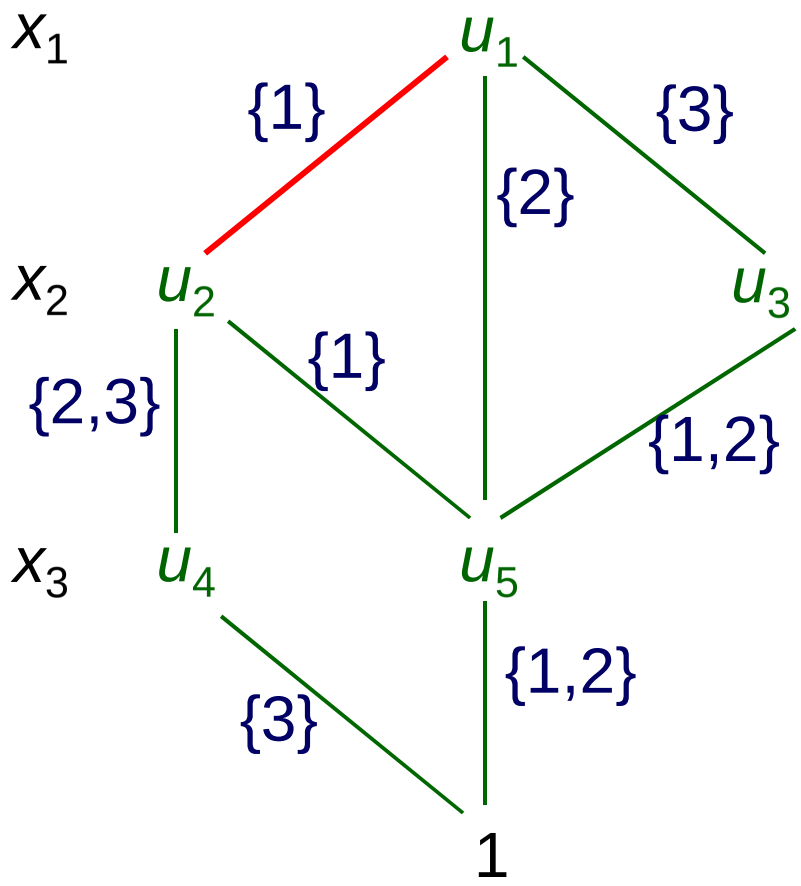




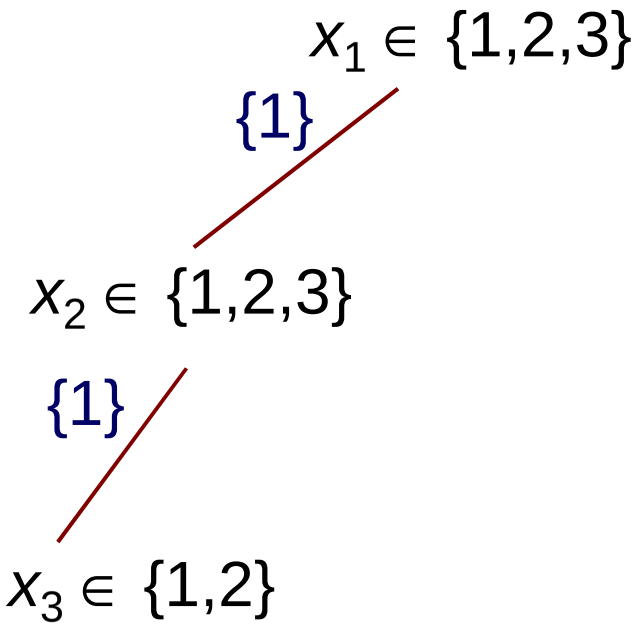
Solving problems with the help of a relaxed MDD

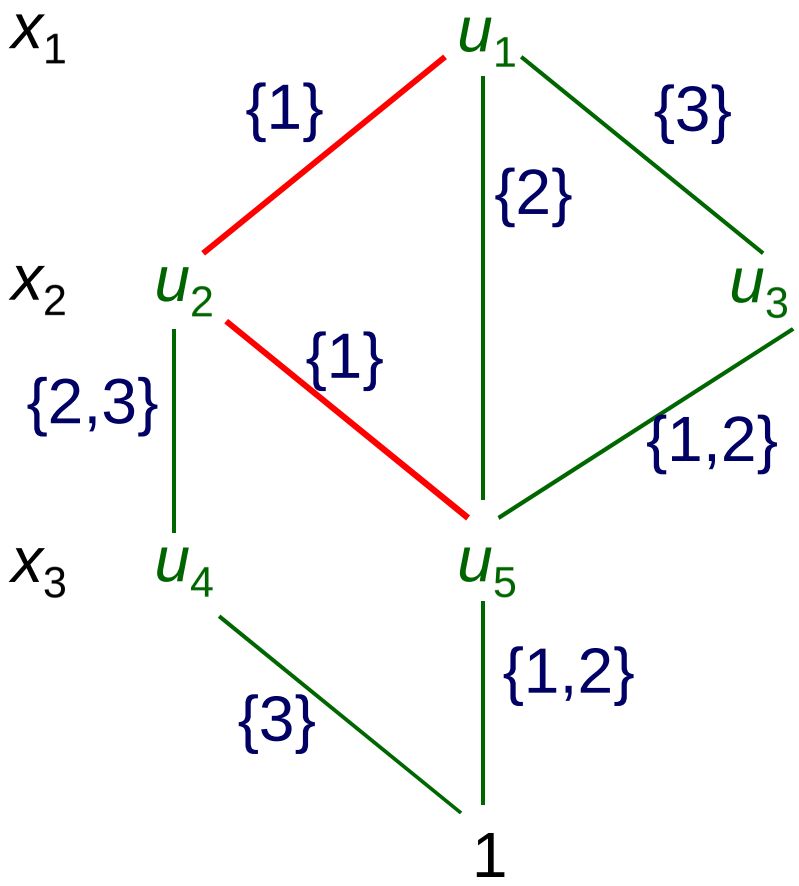
Branching search using relaxed MDD of width 2



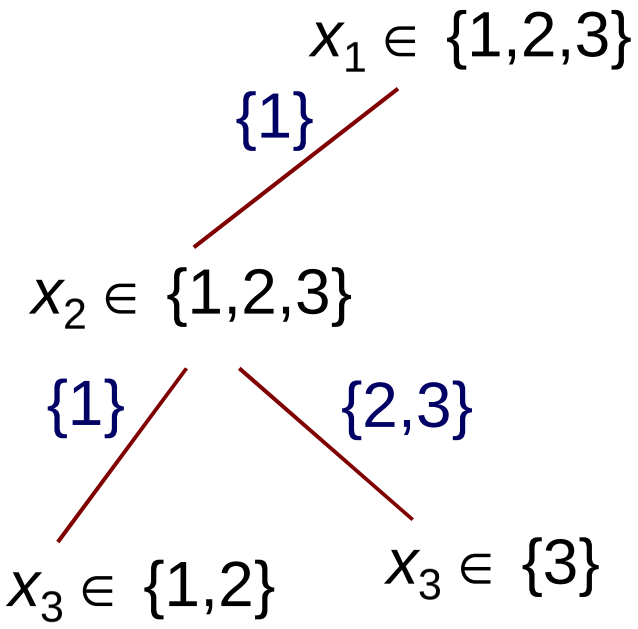


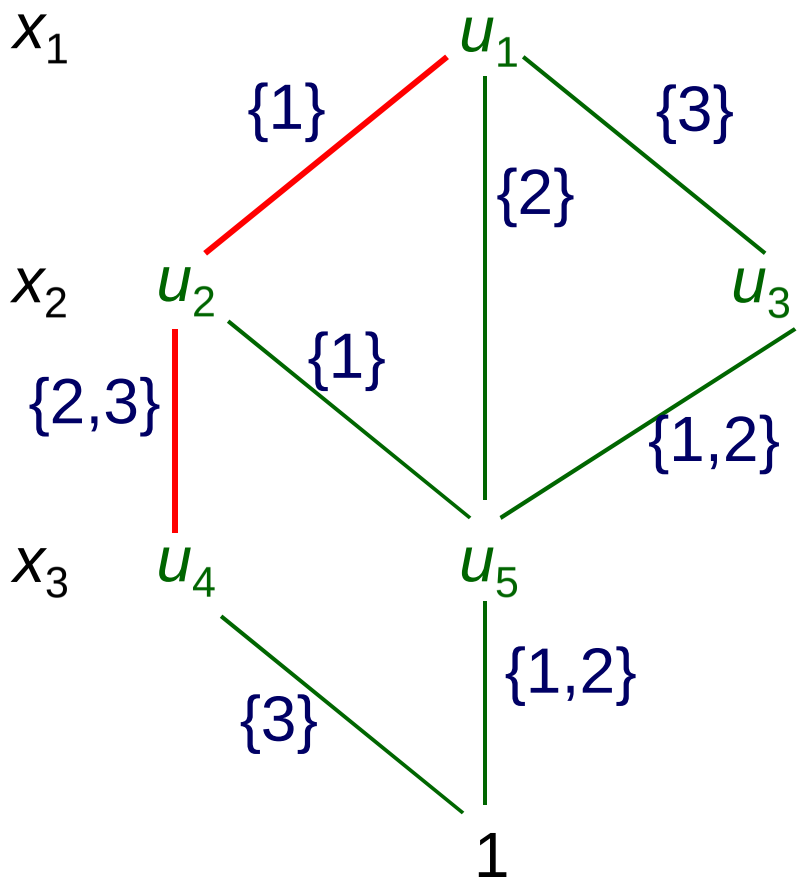
Branching search using relaxed MDD



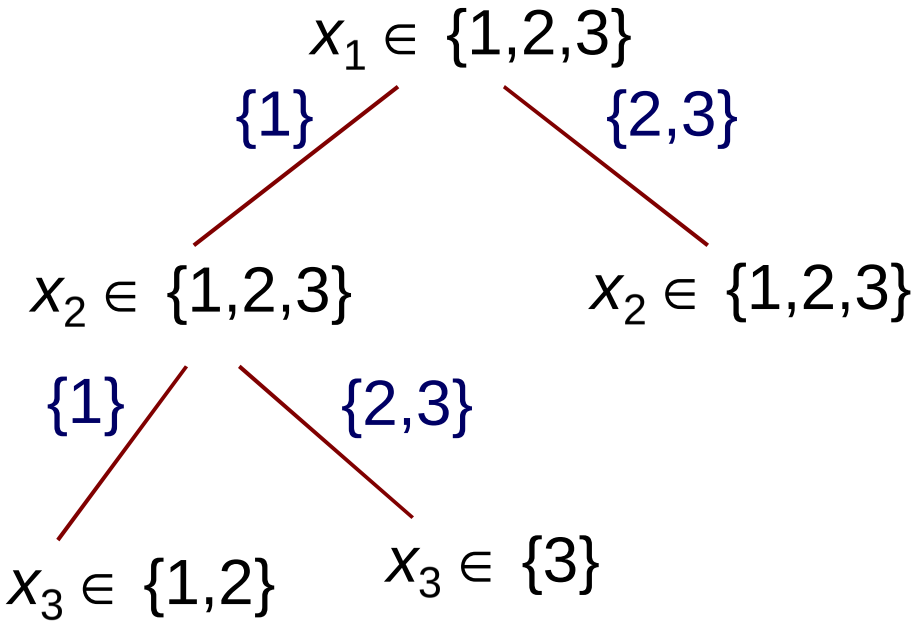


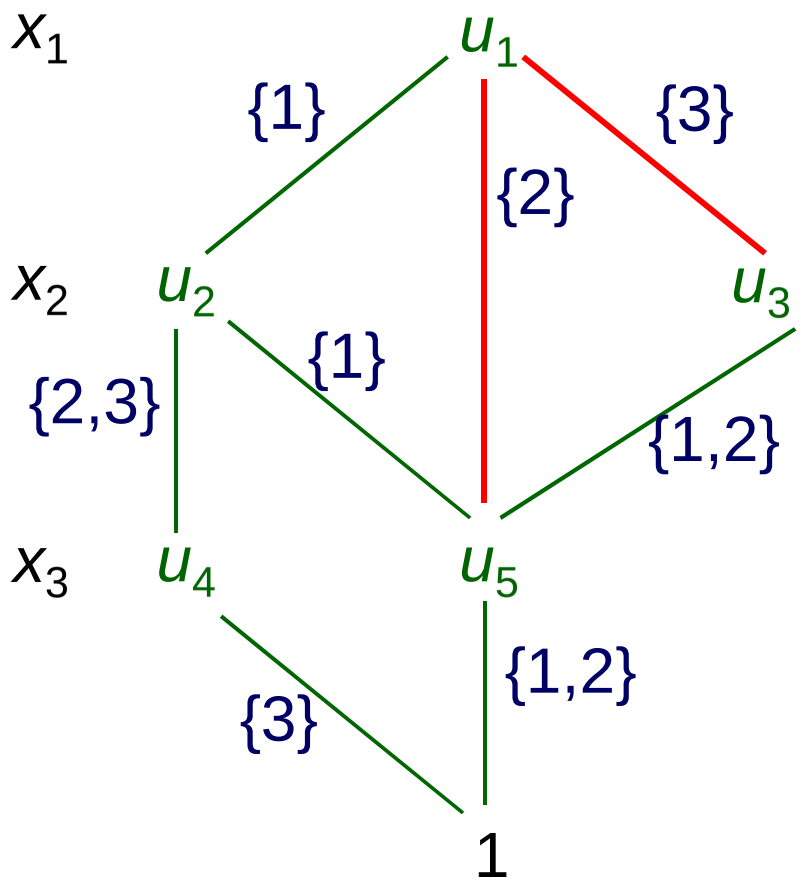
Branching search using relaxed MDD





Branching search using relaxed MDD





And so forth.

Branching search using relaxed MDD

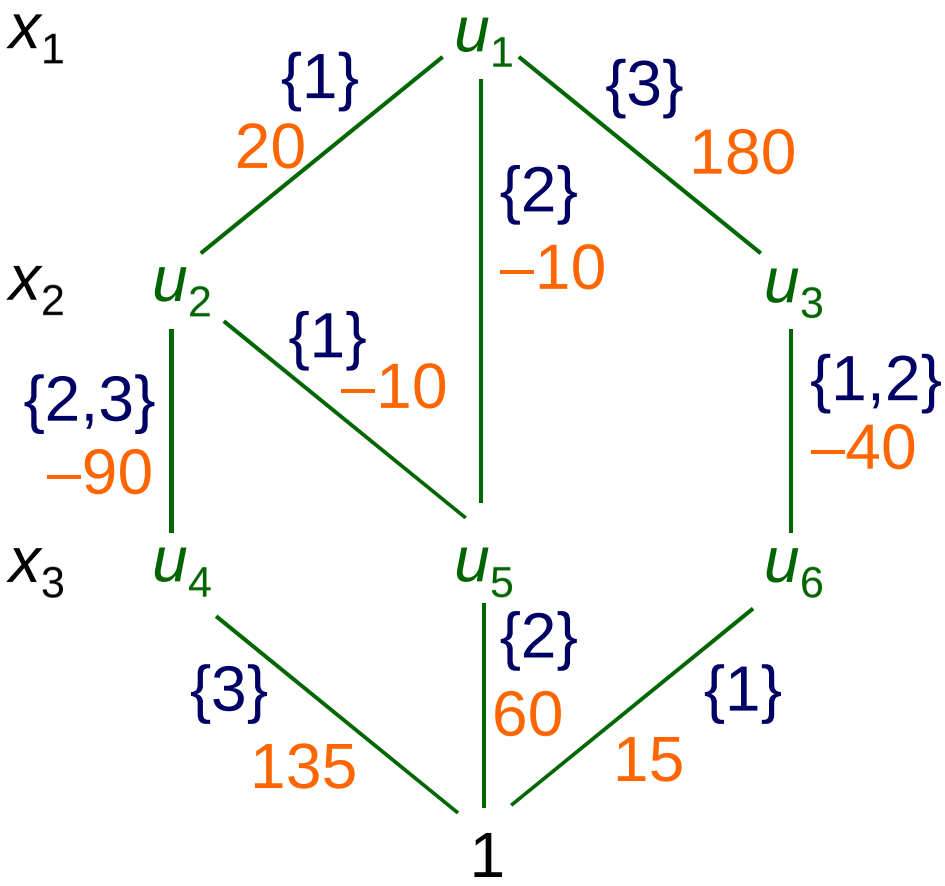
- Can also infer reduced branching domains when search tree order **differs** from MDD order.
  - Simplify the MDD as you move deeper into the tree.
  - Or keep track of “live” part of MDD.

Optimization on an MDD

$$\min 20x_1^2 - 10x_2^2 + 15x_3^2$$

$$\begin{pmatrix} x_1 + x_3 = 4 \\ 3 \le x_1 + x_2 \le 5 \end{pmatrix} \vee \; (x_1, x_2, x_3) = (1,1,2)$$

Objective function can have **any form** as long as it is **separable**.



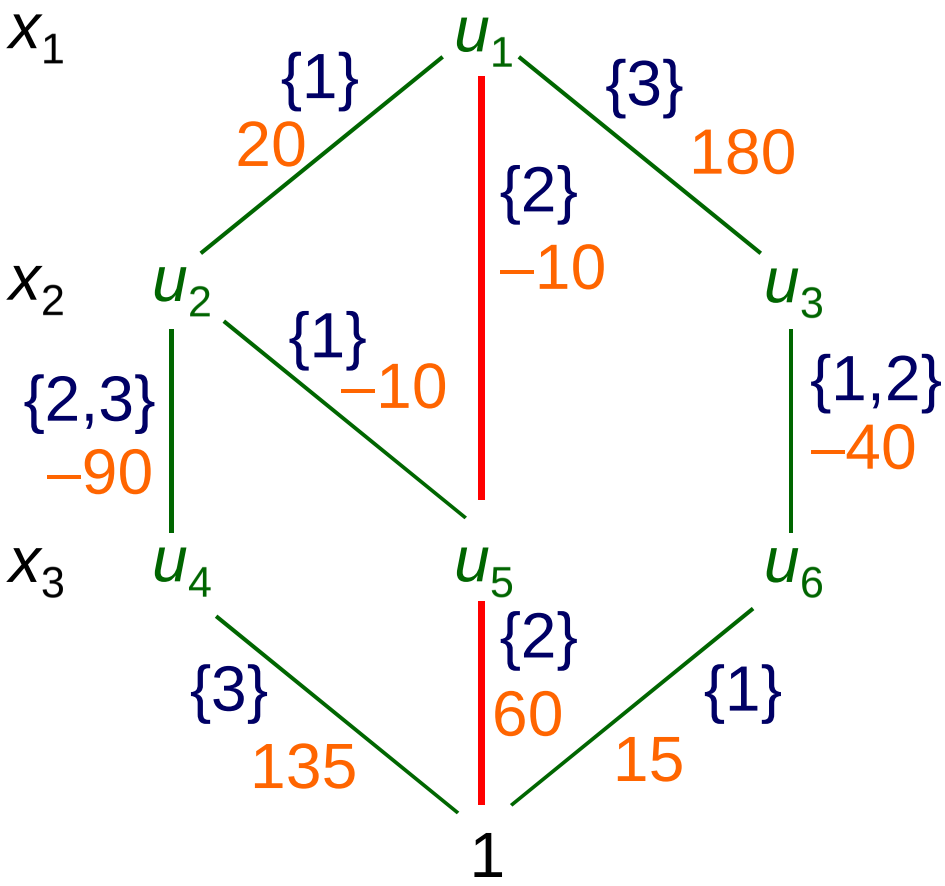
Optimization on an MDD

$$\min 20x_1^2 - 10x_2^2 + 15x_3^2$$

$$\begin{pmatrix} x_1 + x_3 = 4 \\ 3 \le x_1 + x_2 \le 5 \end{pmatrix} \vee \; (x_1, x_2, x_3) = (1,1,2)$$

Optimal value 50 is length of shortest path.

Optimal solution $(x_1,x_2,x_3) = (2,3,2)$



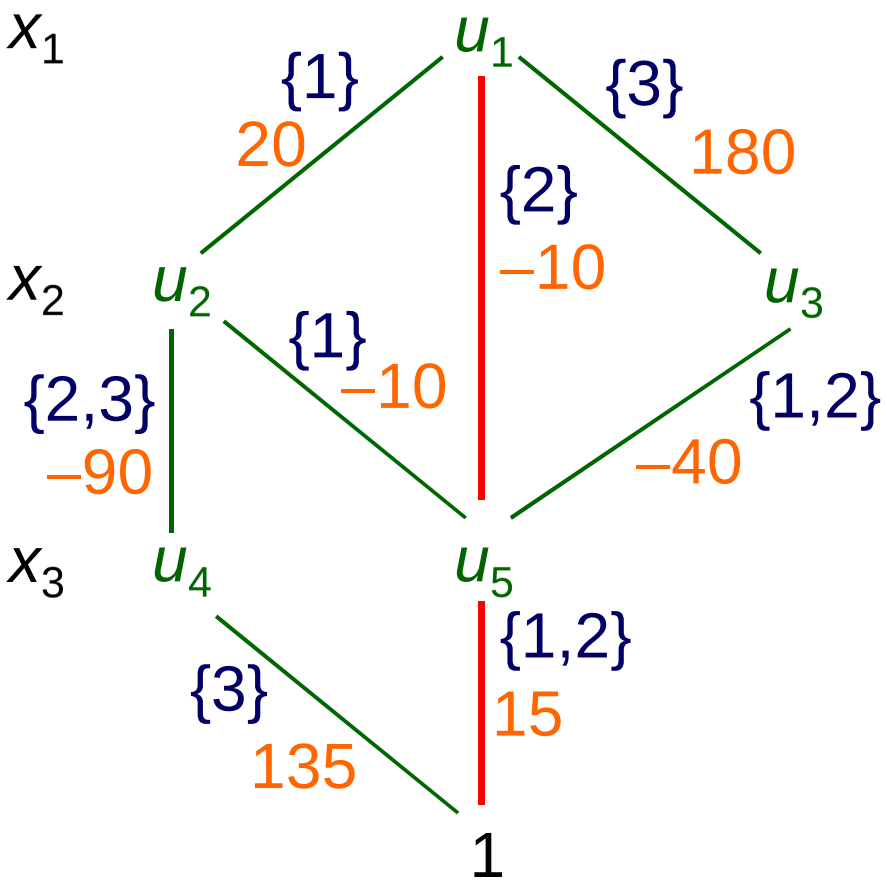
Lower bound from relaxed MDD

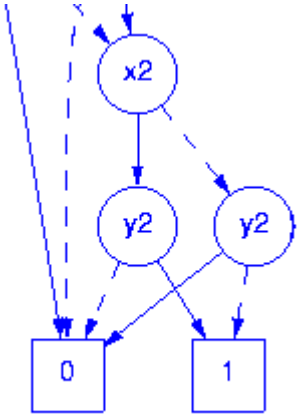
$$\min 20x_1^2 - 10x_2^2 + 15x_3^2$$

$$\begin{pmatrix} x_1 + x_3 = 4 \\ 3 \le x_1 + x_2 \le 5 \end{pmatrix} \vee \; (x_1, x_2, x_3) = (1,1,2)$$

Lower bound 5 is length of shortest path.

Constraint store provides lower bound of –55



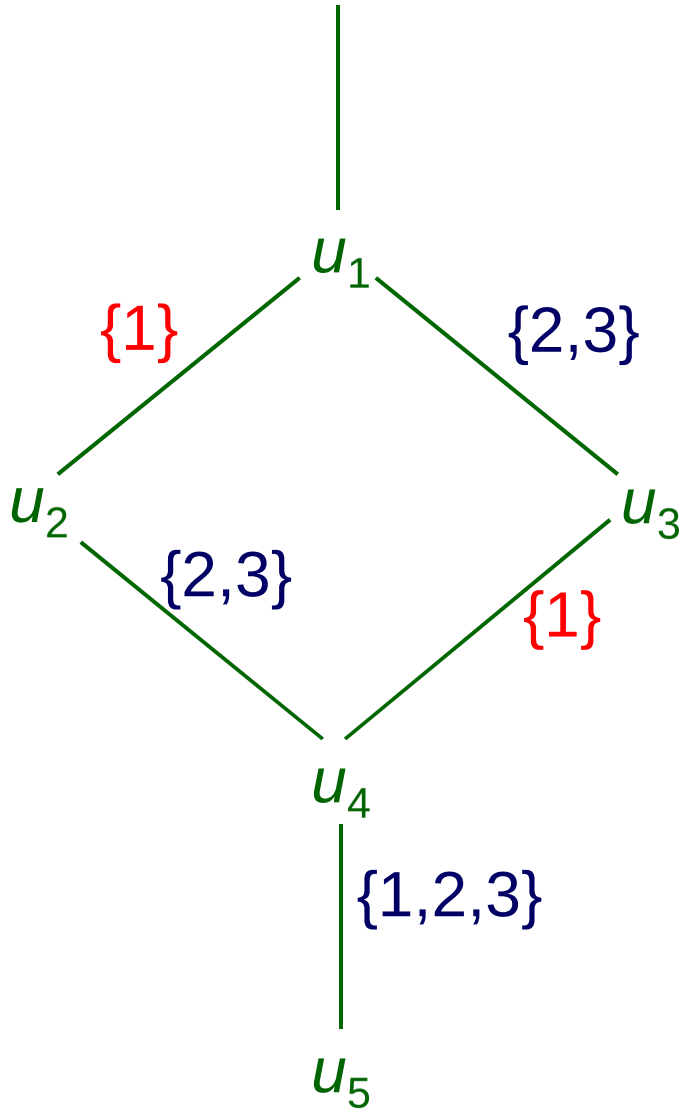


Propagating alldiff through an MDD

Reducing edge domains

Singleton elimination

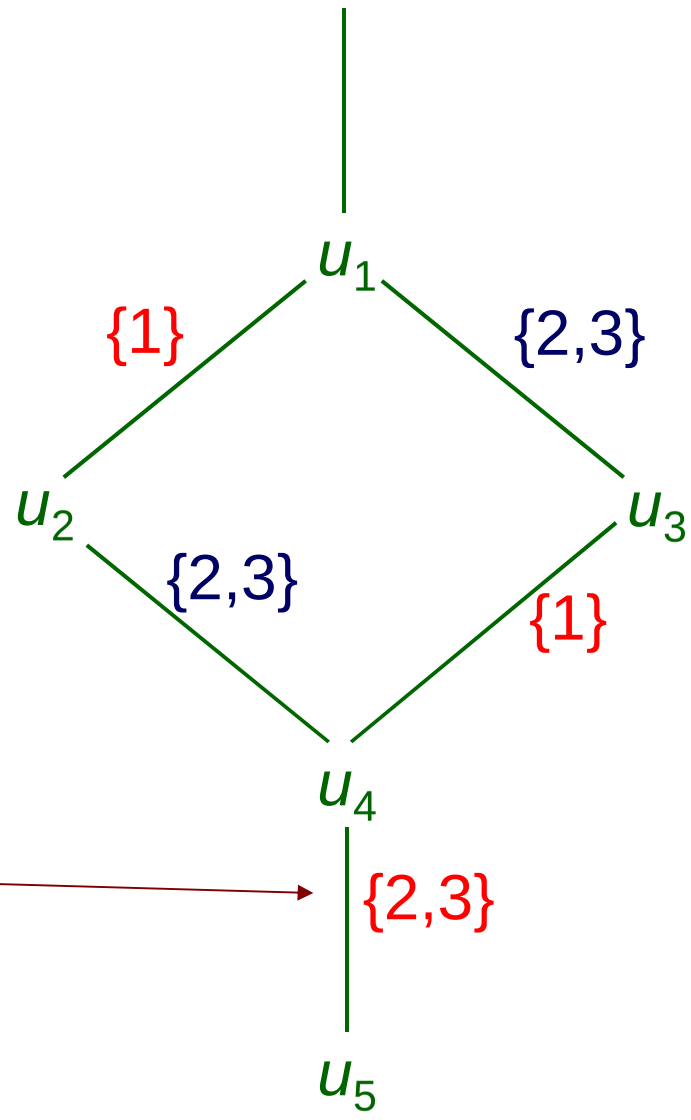
{1} is an edge domain on every path through (u_4,u_5).



Reducing edge domains

Singleton elimination

{1} is an edge domain on every path through (u_4,u_5).

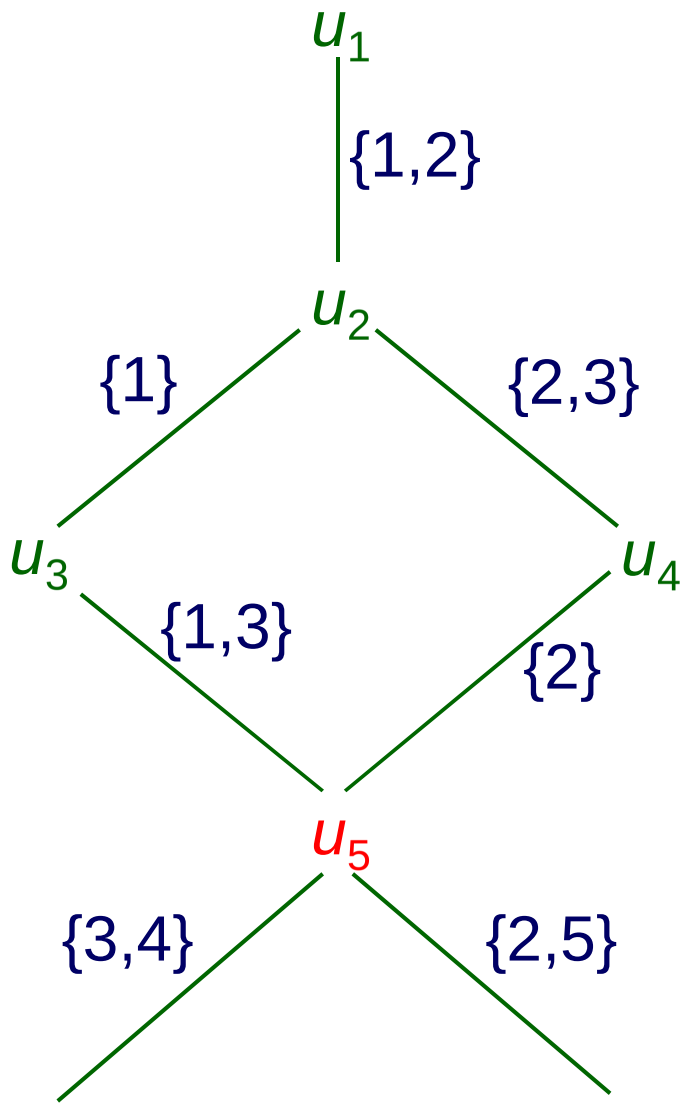


Eliminate 1 from edge domain of (u_4,u_5).

Reducing edge domains

Identifying Hall sets

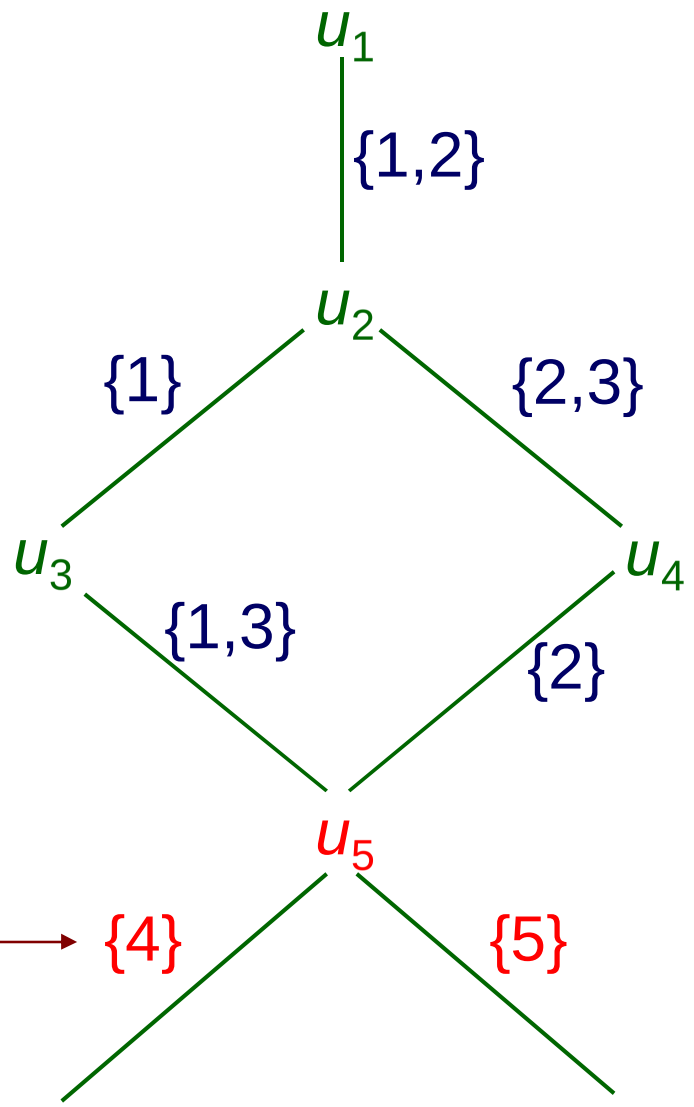
Number of values on paths into u_5 = number of variables above u_5



Reducing edge domains

Identifying Hall sets

Number of values on paths into u_5 = number of variables above u_5



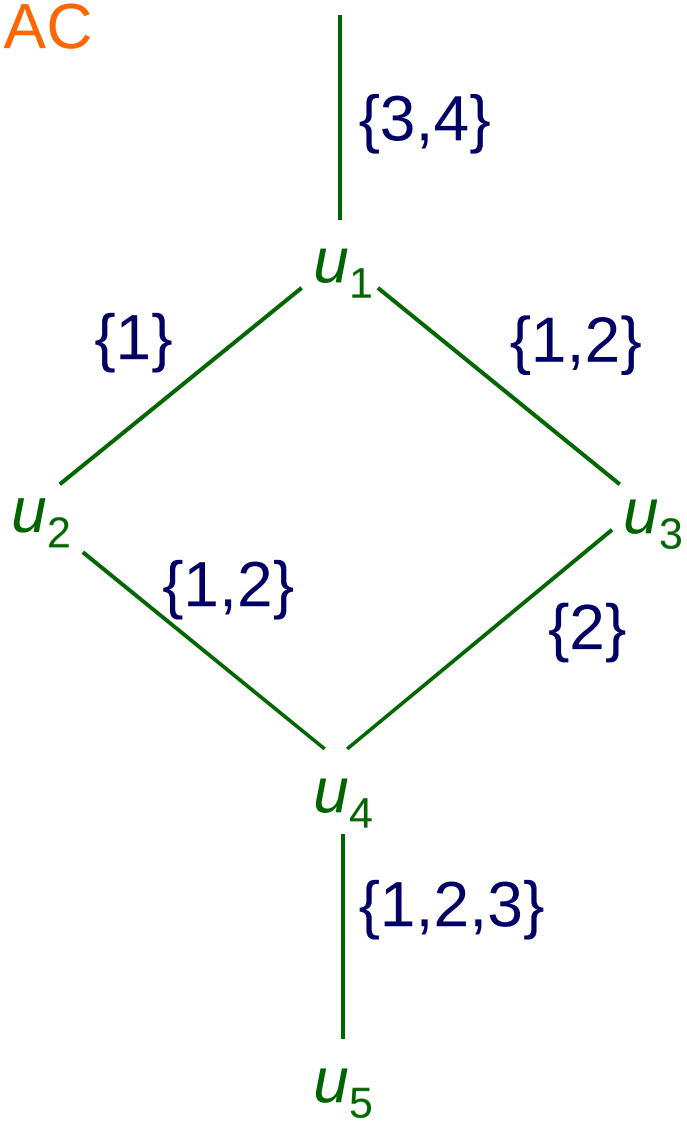
Remove these values from edge domains leaving u_5

Reducing edge domains

Repeated application of GAC

Apply GAC to combined variable domains on paths through (u_4,u_5):

{3,4}×{1,2}×{1,2}×{1,2,3}

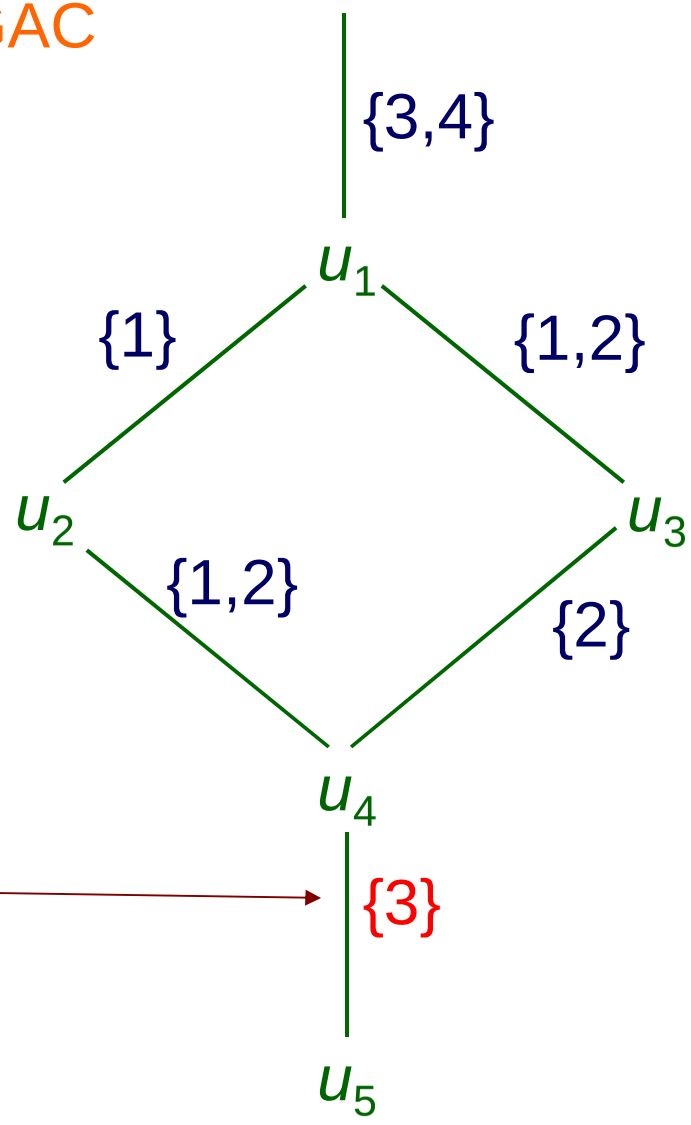


Reducing edge domains

Repeated application of GAC

Apply GAC to combined variable domains on paths through (u_4,u_5):

{3,4}×{1,2}×{1,2}×{1,2,3}



Remove redundant values from edge domain of (u_4,u_5) only.

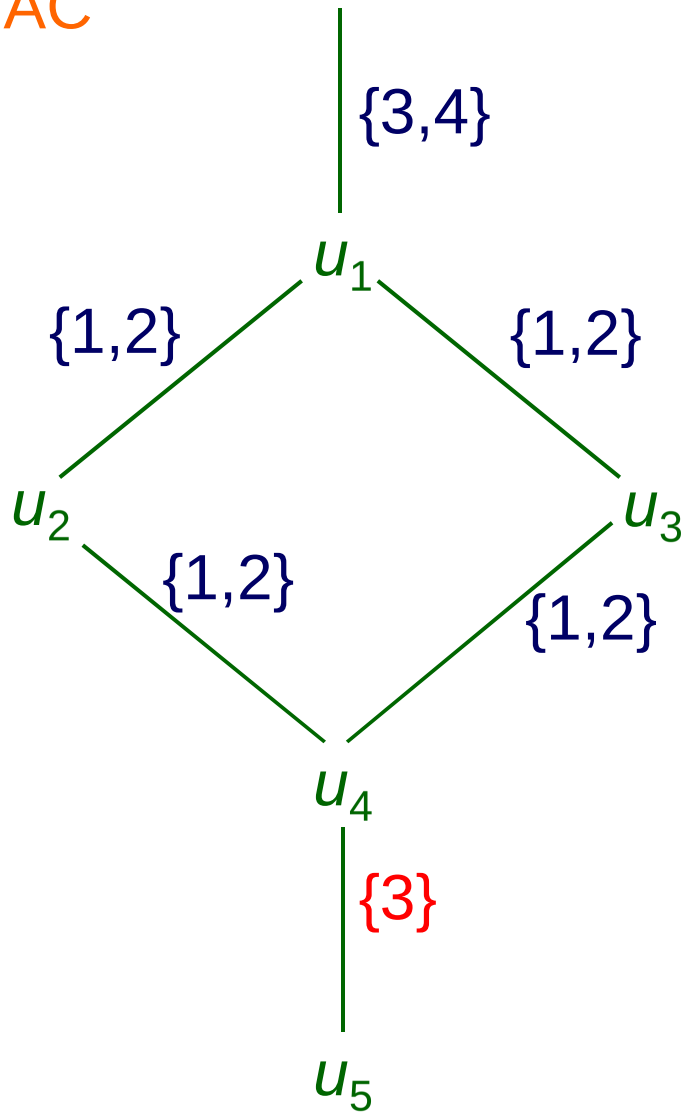
Reducing edge domains

Repeated application of GAC

Can be applied recursively.

Does not in general achieve MDD consistency.

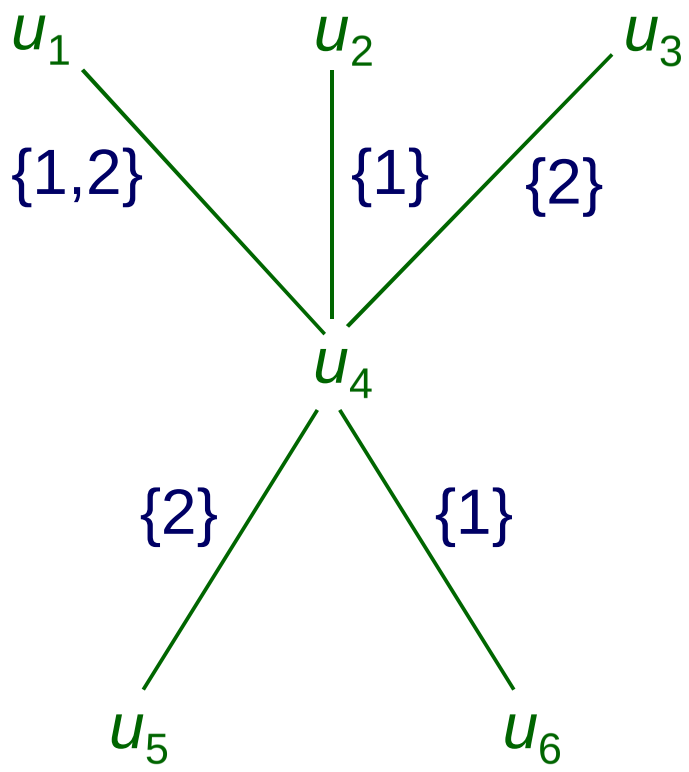
We did not use this technique due to computational cost.



Refining the MDD by node splitting

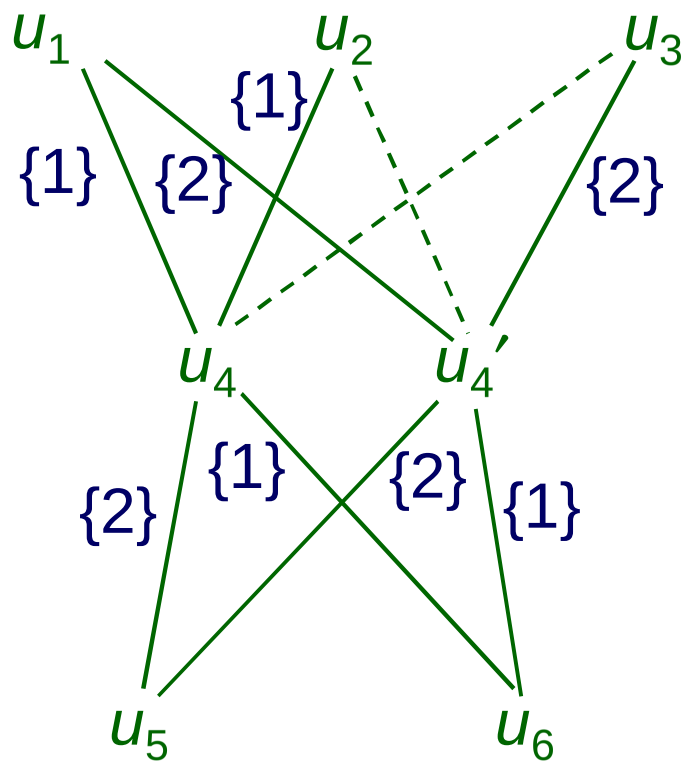
Split node u_4 and the incoming edge domains...

4 feasible solutions
(1,1), (1,2), (2,1), (2,2)



Refining the MDD by node splitting

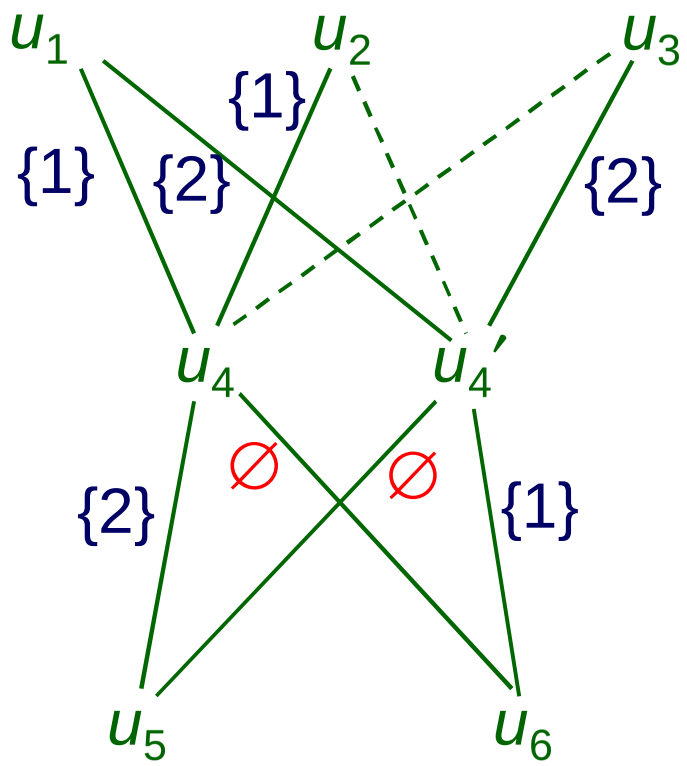
Split node u_4 and the incoming edge domains…



Refining the MDD by node splitting

Split node u_4 and the incoming edge domains…

Remove redundant values.

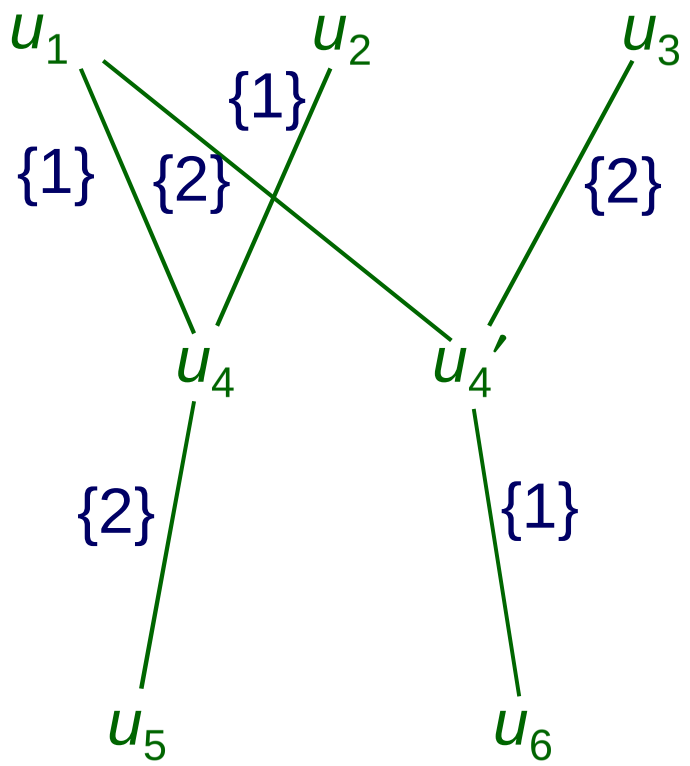


Refining the MDD by node splitting

Split node u_4 and the incoming edge domains…

Remove redundant values.

Delete edges with empty edge domains.

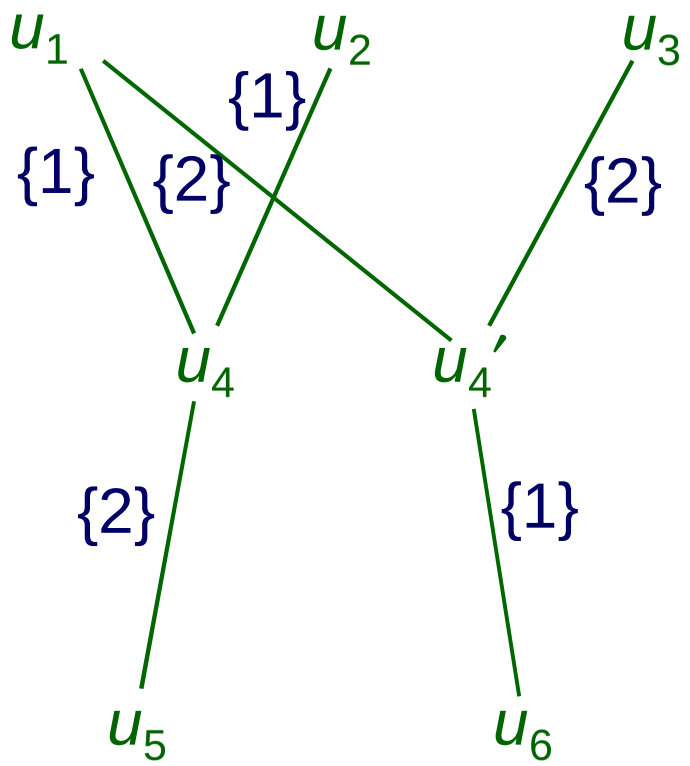


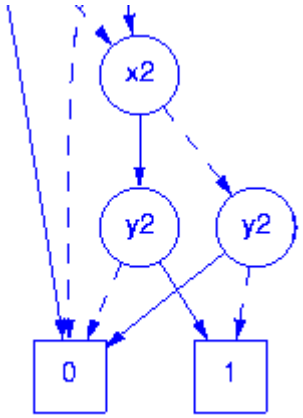
Refining the MDD by node splitting

Splits must not increase MDD width above the maximum.

2 feasible solutions (1,2), (2,1)

(4 solutions before splitting)



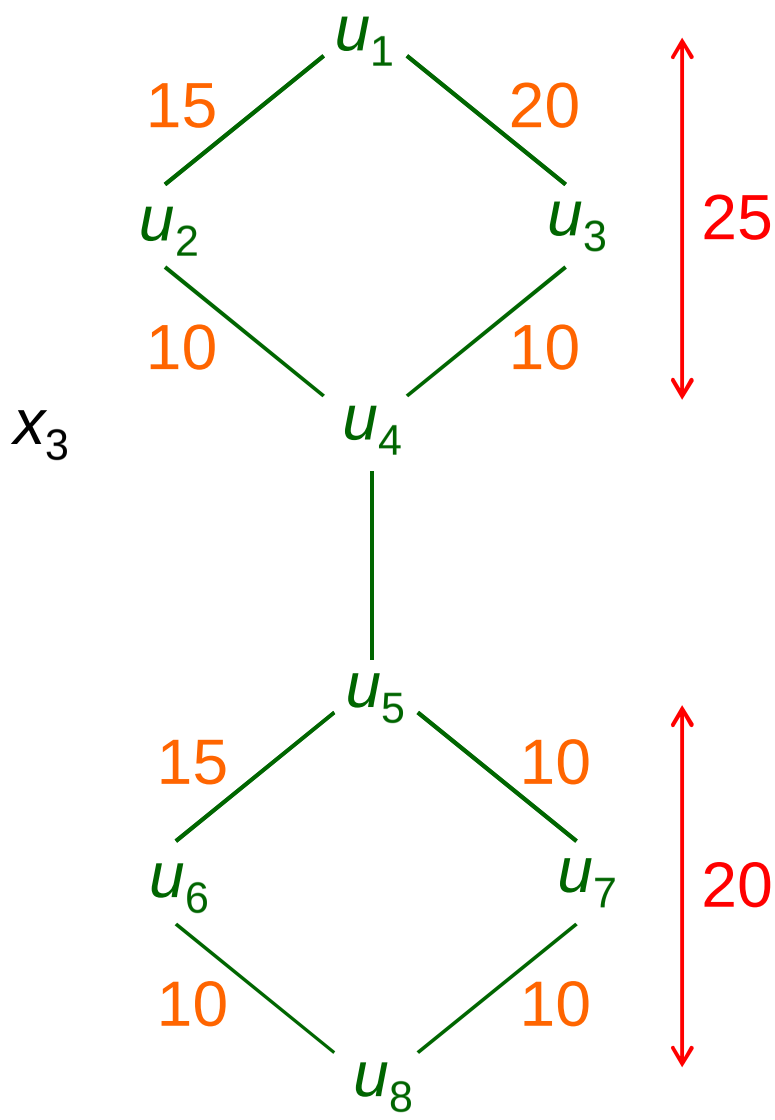


Propagating an integer knapsack constraint through an MDD

Propagating knapsack constraint

LHS can take any separable form.

$$\sum_{i=1}^{5} f_i(x_i) \leq 60$$

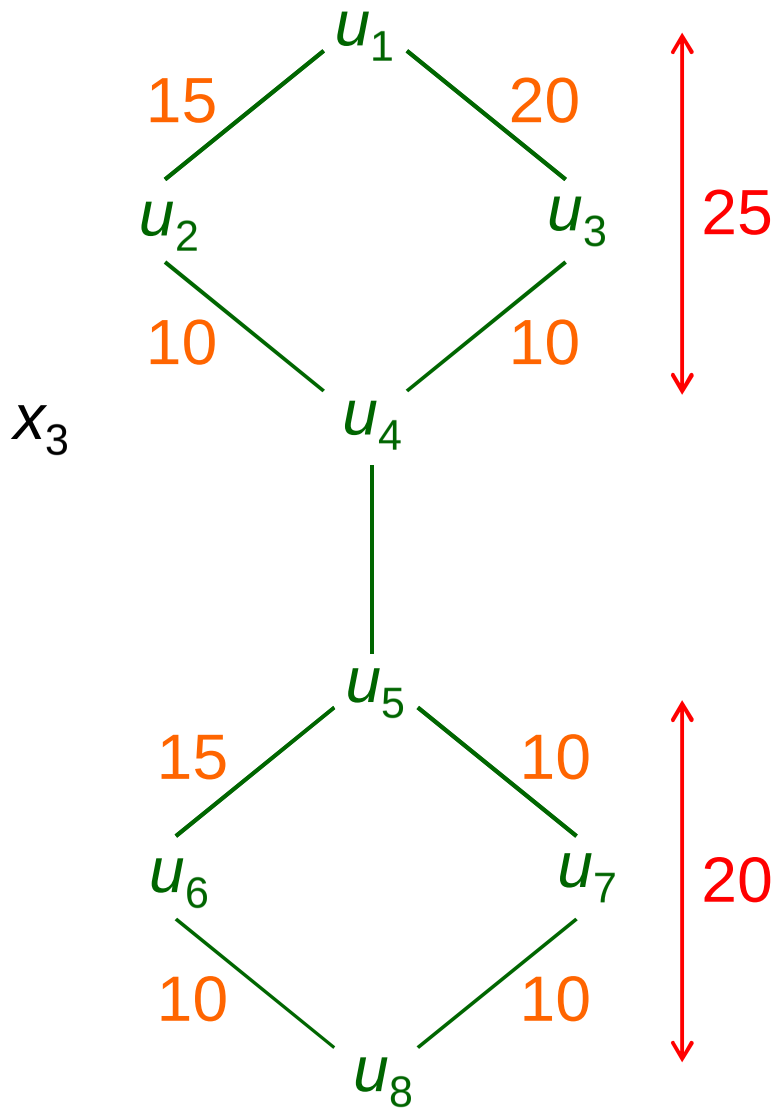


Propagating knapsack constraint

LHS can take any separable form.

$$\sum_{i=1}^{5} f_i(x_i) \leq 60$$

Compute shortest path down to u_4 and up to u_5.



Propagating knapsack constraint

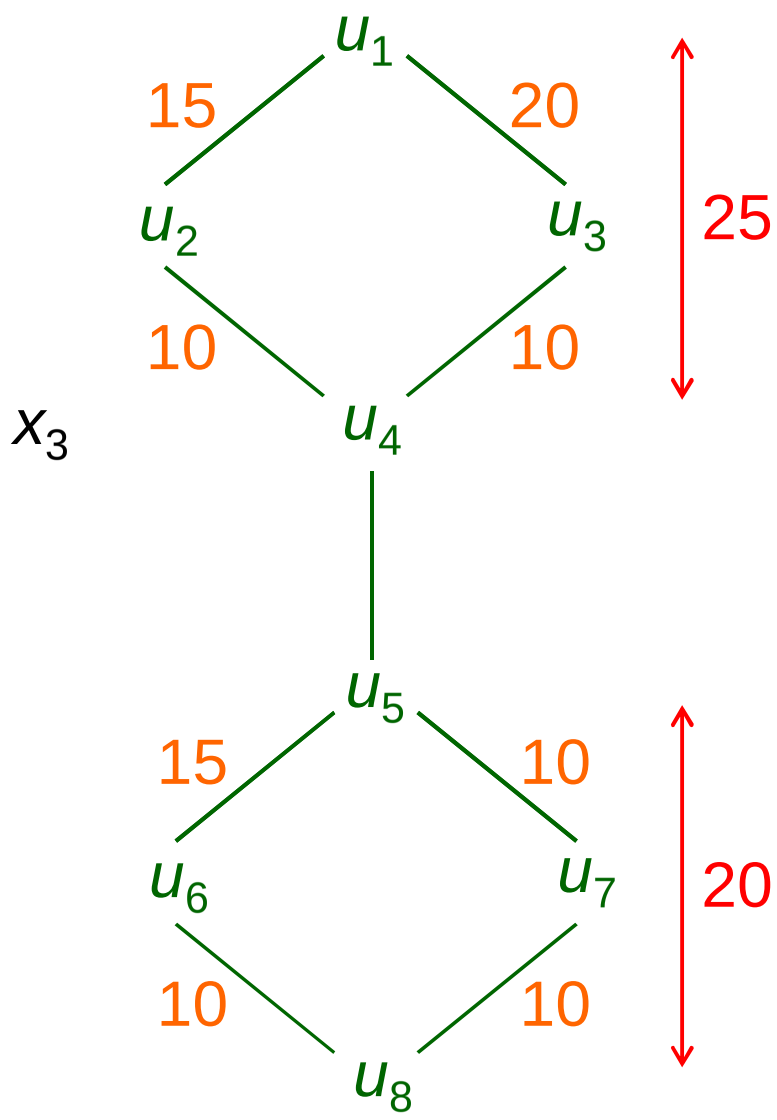
LHS can take any separable form.

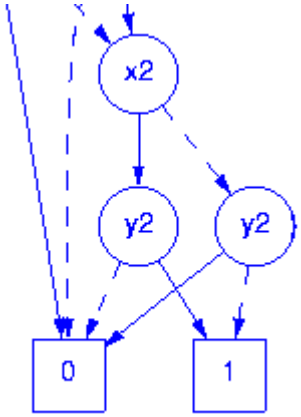
$$\sum_{i=1}^{5} f_i(x_i) \leq 60$$

Compute shortest path down to u_4 and up to u_5.

Delete $x_3 = v$ if

$$25 + f_3(v) + 15 > 60$$





Computational results

Coding

- MDD propagator integrated into Gecode.
  - MDD store implemented as a specialized global constraint.
  - MDD propagator run after other propagators.

Problems

- Multiple alldiffs.
  - Three overlapping alldiffs.
  - Branching order = MDD order.
  - Variables ordered in decreasing order of number of alldiffs containing them
- Integer knapsack optimization problems.

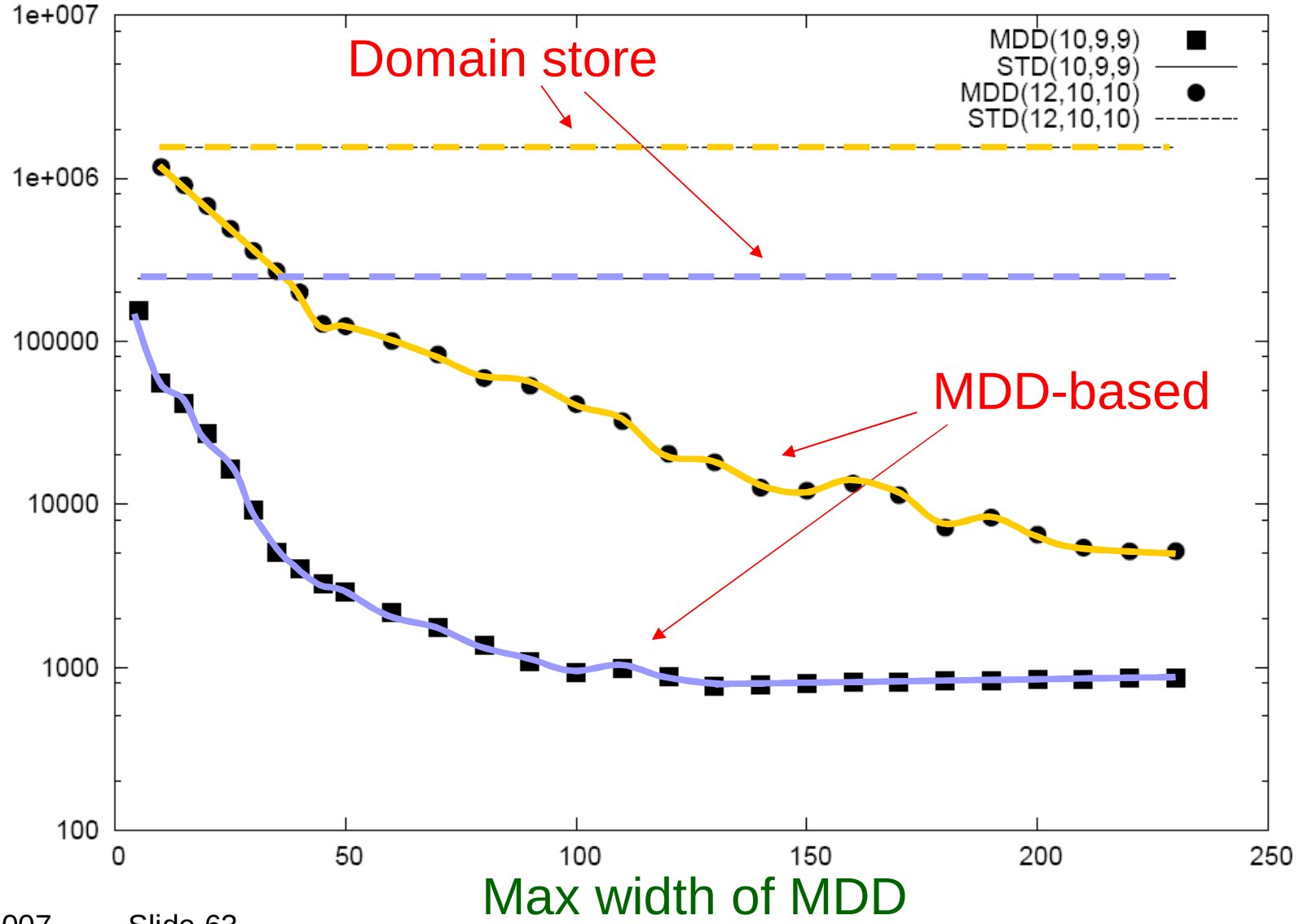
$$\min\left\{\sum_i c_i x_i \;\middle|\; \sum_i a_i x_i \le \alpha L\right\}$$

  - L = max value of LHS.
- 60 random instances of each problem size.

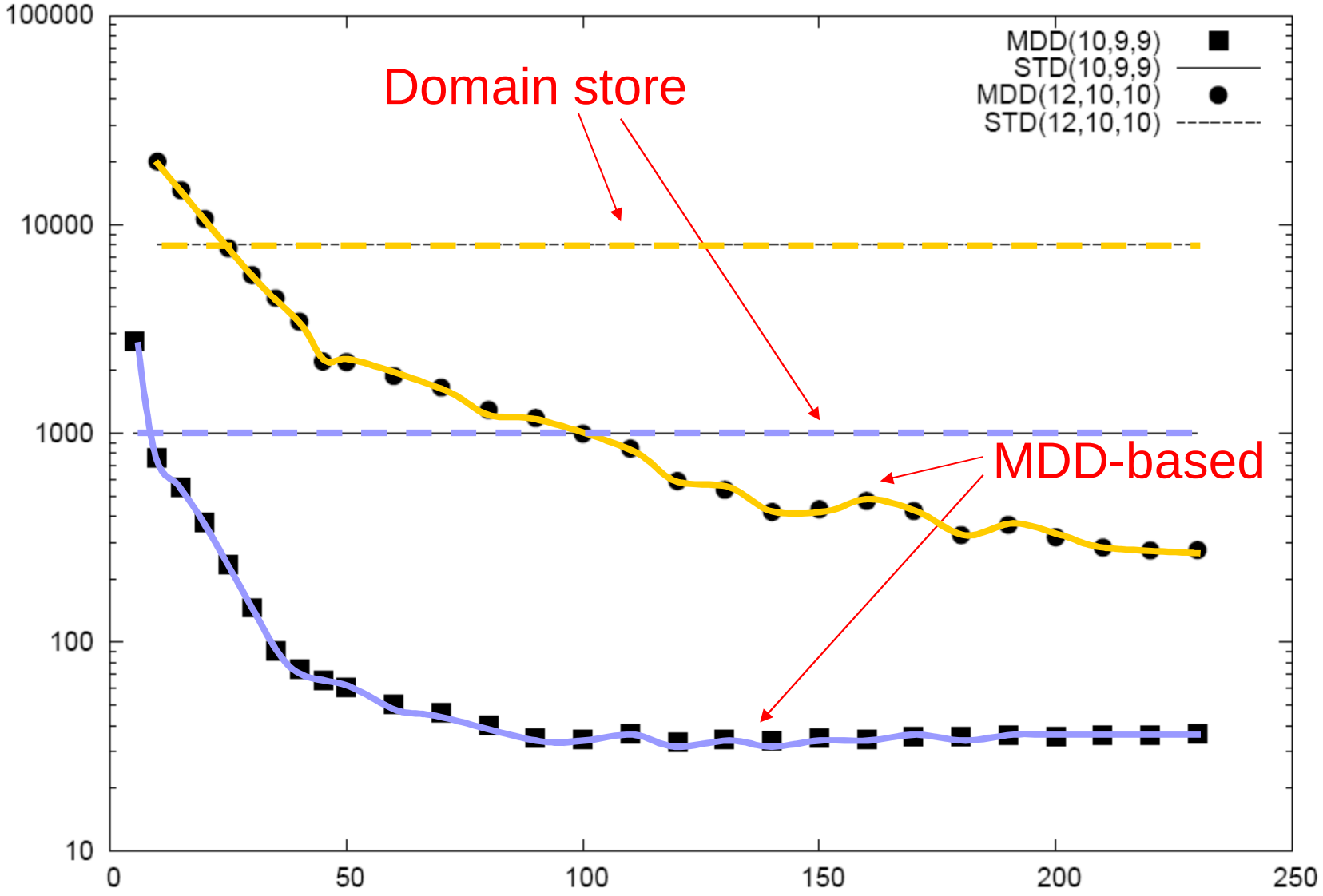
Results – Multiple alldiffs

- Conventional domain store – About a **million** search tree nodes.
- MDD constraint store **– No backtracking**.
  - Even for width as small as 5.
- **Wider** MDDs require **fewer splits**.

Number of branches + number of splits



Computation time (milliseconds)



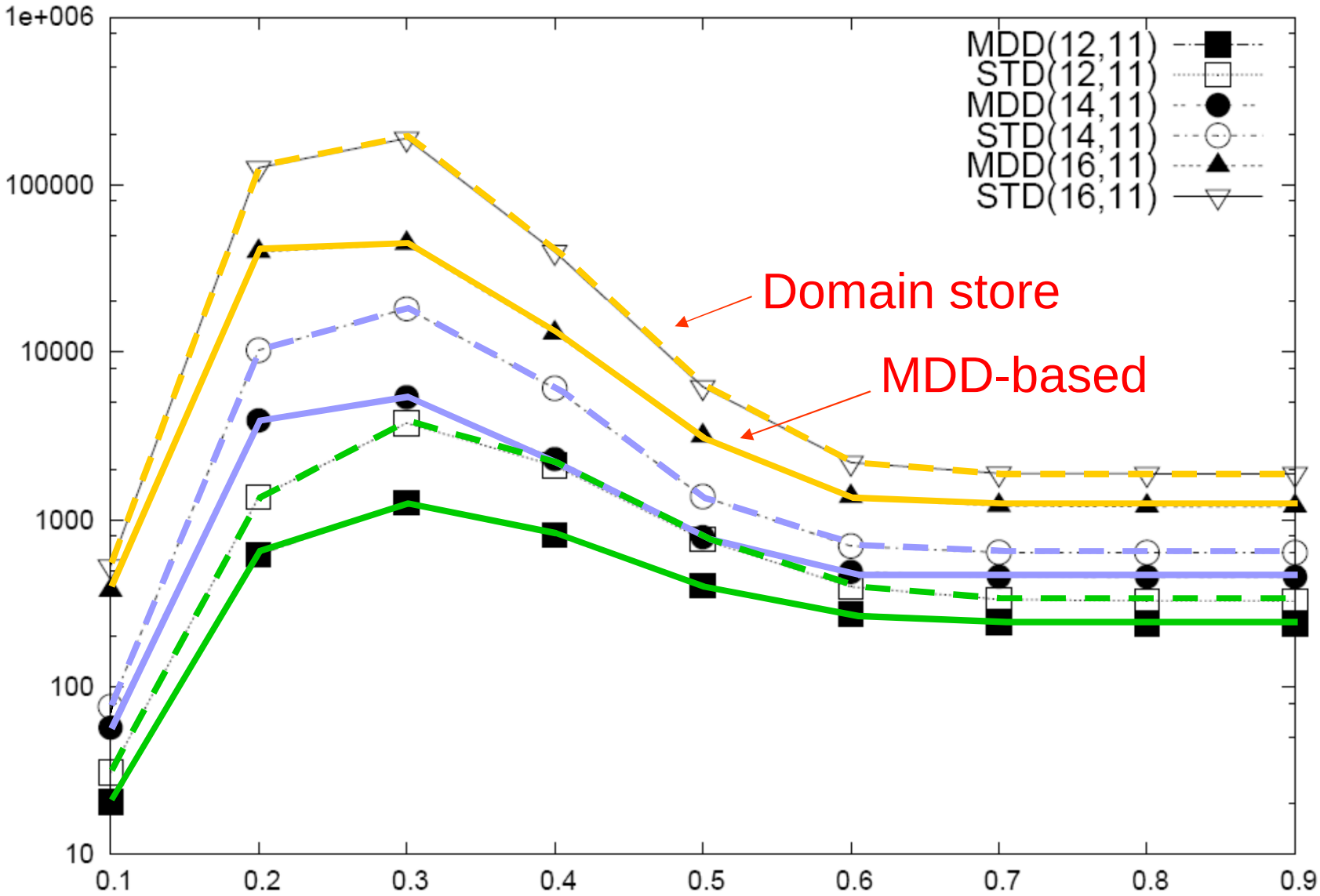
Results – Multiple alldiffs

- **MDDs are faster** for all but smallest widths.
- Speed advantage of MDDs **increases** with maximum **width**.
  - Up to a factor of about **30**.
  - Within the range of widths studied.

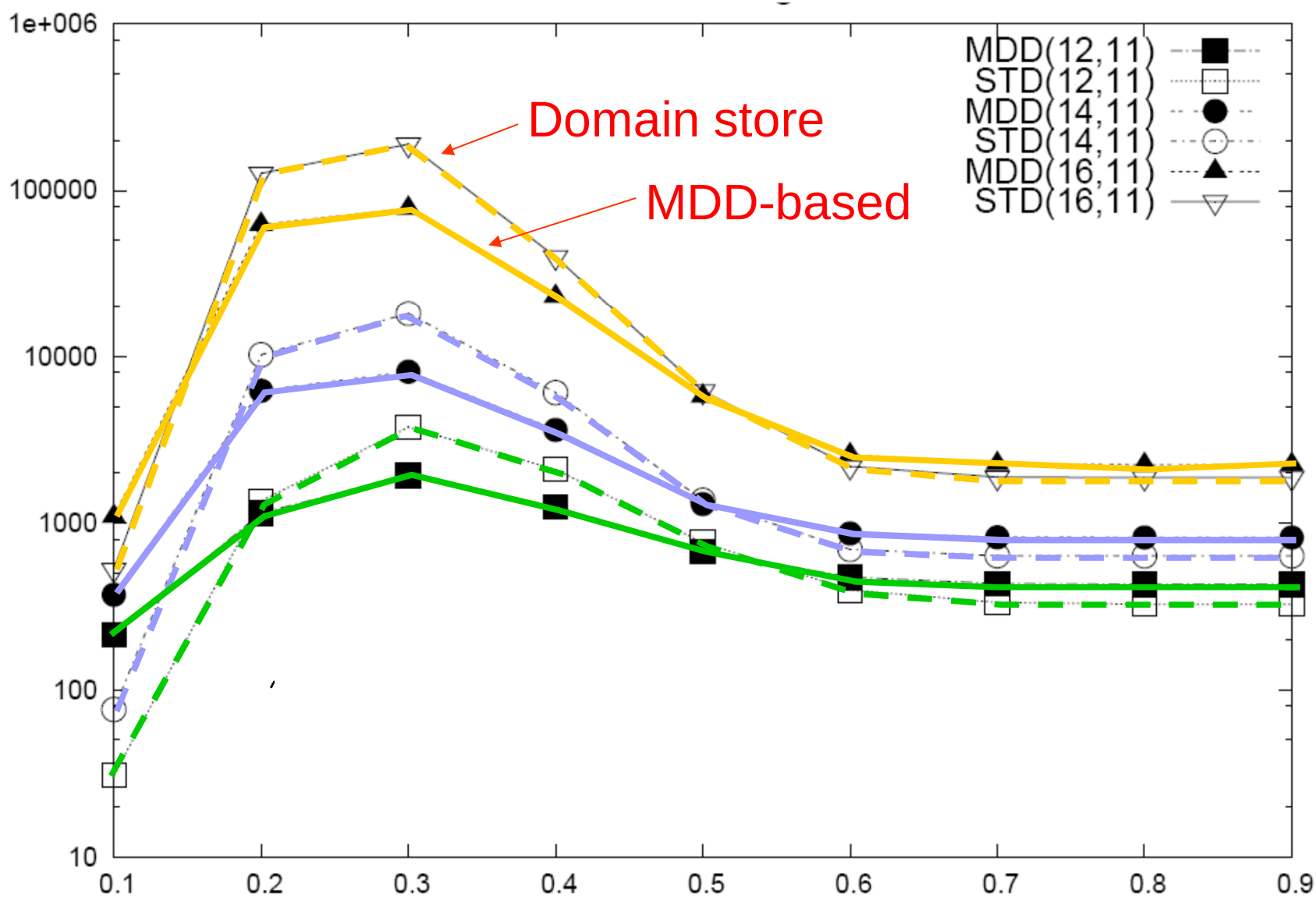
Results – Integer knapsack

- MDD store requires somewhat fewer search tree nodes.
  - As few as 20% as many nodes.
- MDD store usually requires less work, measured by number of nodes + number of splits.

Size of search tree



Number of nodes + number of splits

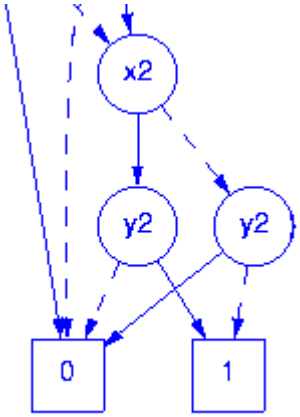


Results – Integer knapsack

- MDD store requires about three times as much **computation time**.
  - Must branch deeply into the search tree before finding feasible solutions.
- Similar results when minimizing linear expression subject to alldiff.

Counterintuitive result?

- Typically investment in node processing pays off more in **optimization** problems.
  - Here, the opposite is true.
  - **Why?**
- Branch-and-bound methods **solve relaxation** at each node of search tree.
  - Solution of relaxation is often **feasible**.
    - i.e., integral in integer programming problems.
  - We did **not** solve the **MDD relaxation**.



Conclusions and research issues

Conclusions

- MDD-based constraint store provides **substantial advantage** in **multiple alldiff** problems.
  - **Wider** MDDs yield greater speedups (factor of 30).
  - **No backtracking**, even with narrow MDDs.
- **Intensive processing** at search tree nodes can pay off when constraint store is richer.
- MDD-based constraint store is **slower** for **integer knapsack optimization problems**.
  - Probably because we did not solve the **MDD relaxation**.

Research Issues

- How to solve a **relaxed MDD**?
  - Does this make MDDs superior for **optimization problems**?
- How to adjust **width** of relaxed MDD?
  - Can always set width = 1 if MDDs not useful.
- How to propagate **other global constraints** in an MDD-based constraint store?


THE
END