MDD-based propagation of `among` constraints

Samid Hoda Willem-Jan van Hoeve John N. Hooker
Carnegie Mellon University

INFORMS Annual Meeting
San Diego – October 14, 2009

Outline

- Introduction
  - Domain stores and MDD stores
  - Propagation
- Experiments
  - Generating all solutions
  - Finding the first feasible
- Conclusion and research issues

INTRODUCTION

Local vs global

- This tension pervades science and mathematics
  - ✓ Pros of local structure: **simplicity**
  - ✗ Cons of local structure: **limited**, lacks global pt of view

- Strategies for extending local reasoning
  - Expand the notion of locality
  - Combine summaries of local structure

Constraint programming

- In CP local structure = processing individual (basic) constraints
  - ✓ Pros: able to **exploit structure** of (basic) constraints
  - ✗ Cons: **overlooks implications** of combined constraints

- Strategies
  - Expanding notion of locality: **global constraints**
  - Combining summaries of local structure: **domain store**

Domain store

- Stores the current variable domains
  - Values that occur in *some* feasible solution
- The domain store relaxation
  - Provides a (weak) **summary** of global structure
  - **Combines** local structure (can be very lossy)
  - Basis for constraint **propagation**

Domain store: advantages

✓ Simple structure

- Provides **natural input** to filtering algorithms
- **Minimal overhead** when embedded in search

✓ **Guides branching** (on variables) in a natural way

- Just split variables

Domain store: disadvantages

✗ Transmits relatively **little information** between constraints

✗ A **weak relaxation** of the problem

- Ignores variable interaction
- Relaxation is a Cartesian product of domains

✗ Result

- Search trees **too large**
- **Too little** processing at each node

A stronger relaxation

- Enrich the constraint store
  - Use a relaxed *multivalued decision diagram* (MDD)
  - With binary domains MDD = BDD (binary decision diagram)

- An MDD is a **compact representation** of the search tree
  - Isomorphic subtrees are merged
  - An MDD is **relaxed** by limiting width

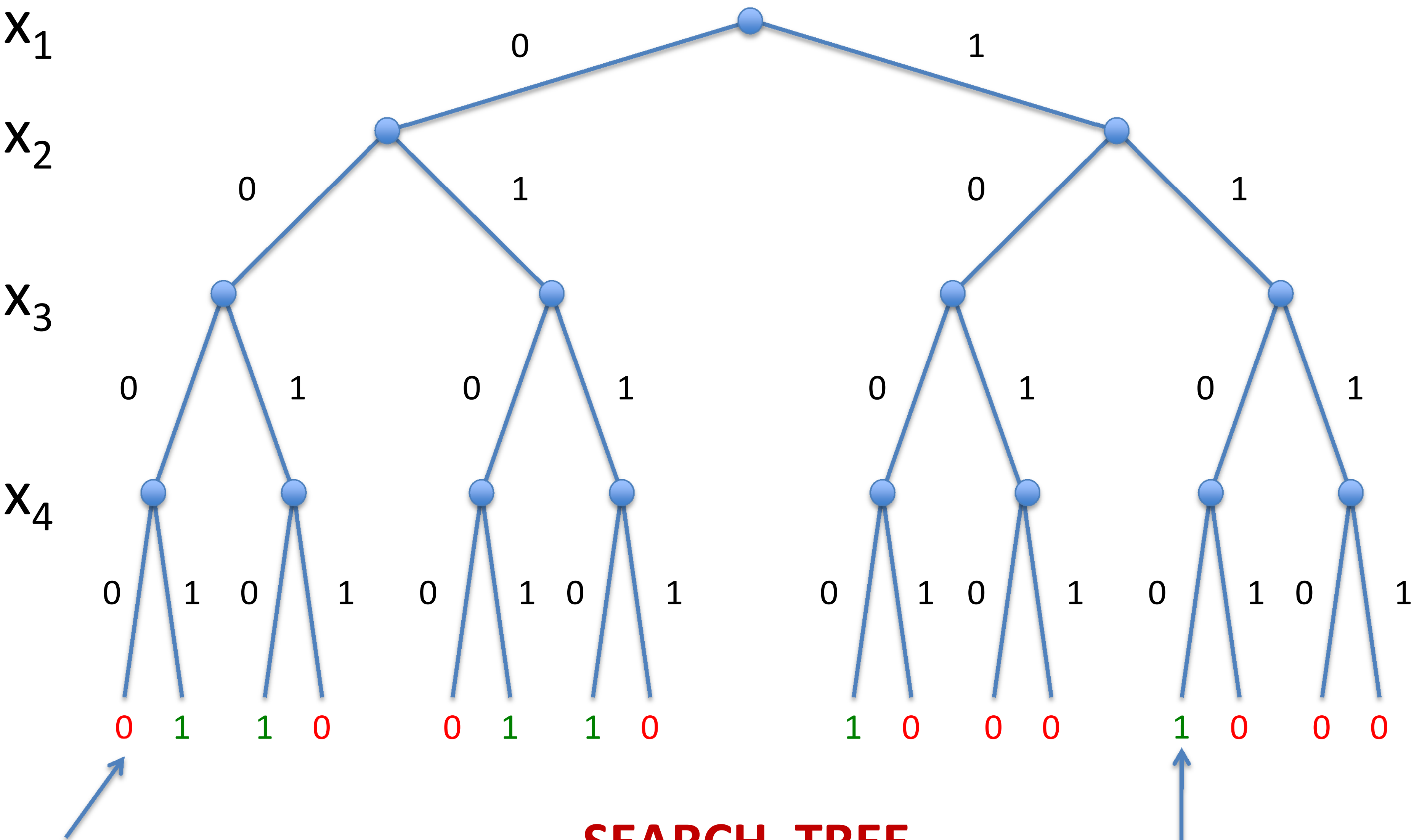
Advantages of a BDD store

- ✓ Transmits **more** information than a domain store
  - Strength is **adjustable**: depends on width
- ✓ **Guides branching** in a natural way
  - Representation is closer to branching tree
- ✓ Results
  - **Smaller** search trees
  - Justifies **more** processing per node
  - Better **integration** of CP/IP

Global constraints and MDD stores

- Global constraints
  - **Static**
  - Modeler **imposes** structure
- MDD store
  - **Dynamic**
  - **Identifies** structure as the solution process evolves
- Best of both
  - Propagating global constraints through MDD

EXAMPLE



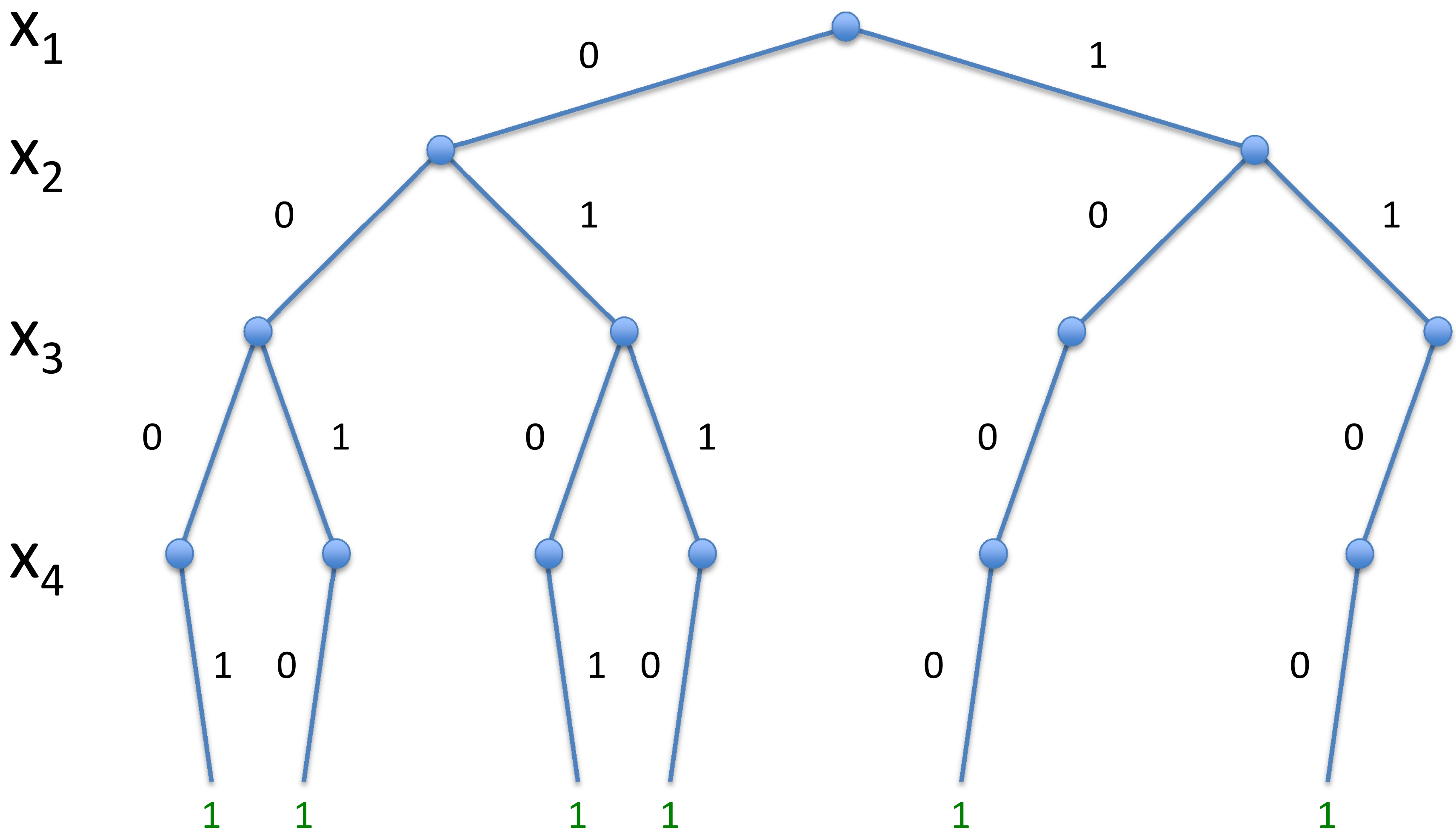
SEARCH TREE

INFEASIBLE

FEASIBLE

$\texttt{among}(\{x_1,x_3,x_4\},\{1\},2,2)$

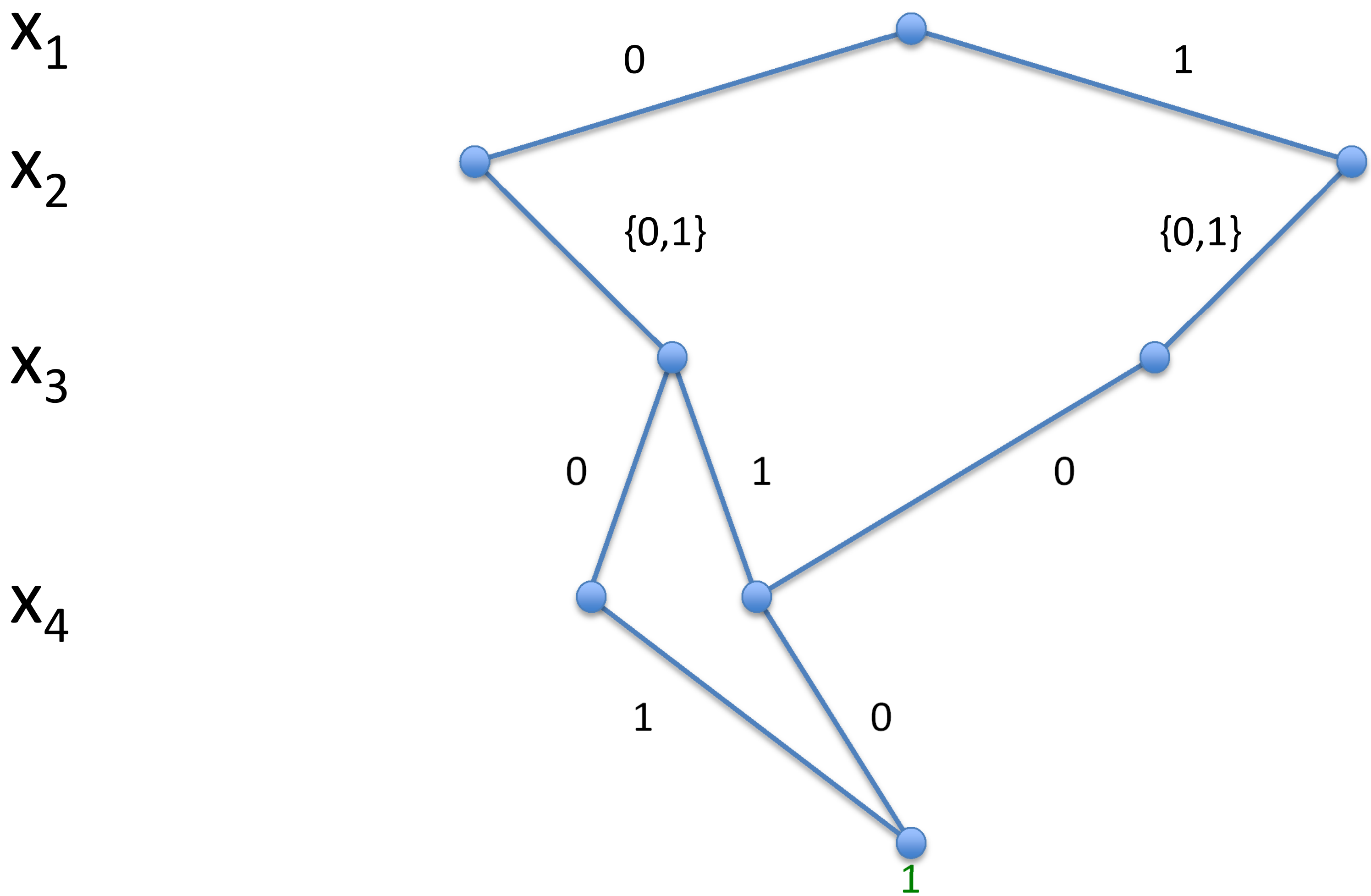
EXAMPLE



REMOVE INFEASIBLE SOLUTIONS

$\texttt{among}(\{x_1,x_3,x_4\},\{1\},2,2)$

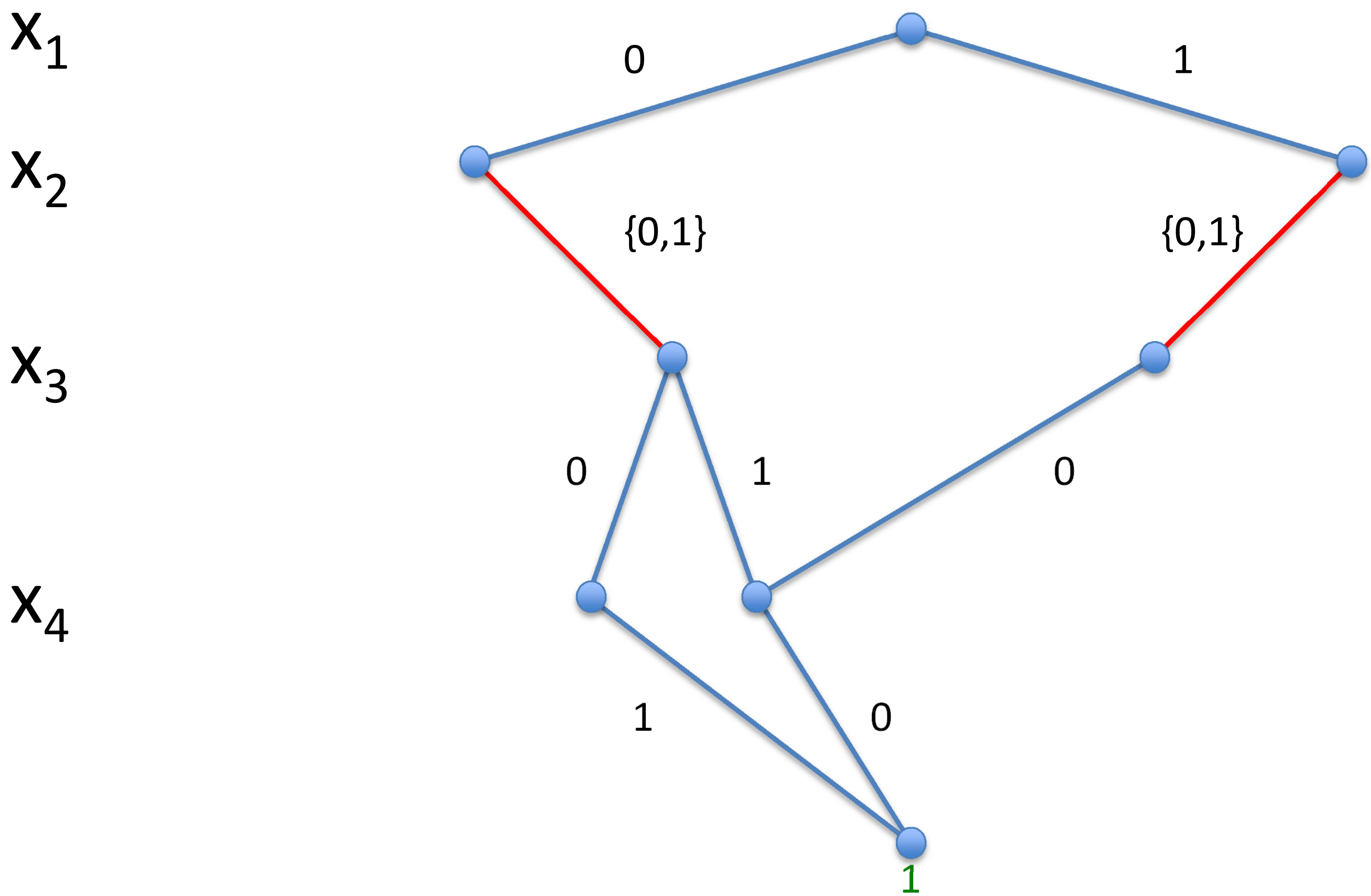
EXAMPLE



MERGE ISOMORPHIC SUBTREES

$\texttt{among}(\{x_1,x_3,x_4\},\{1\},2,2)$

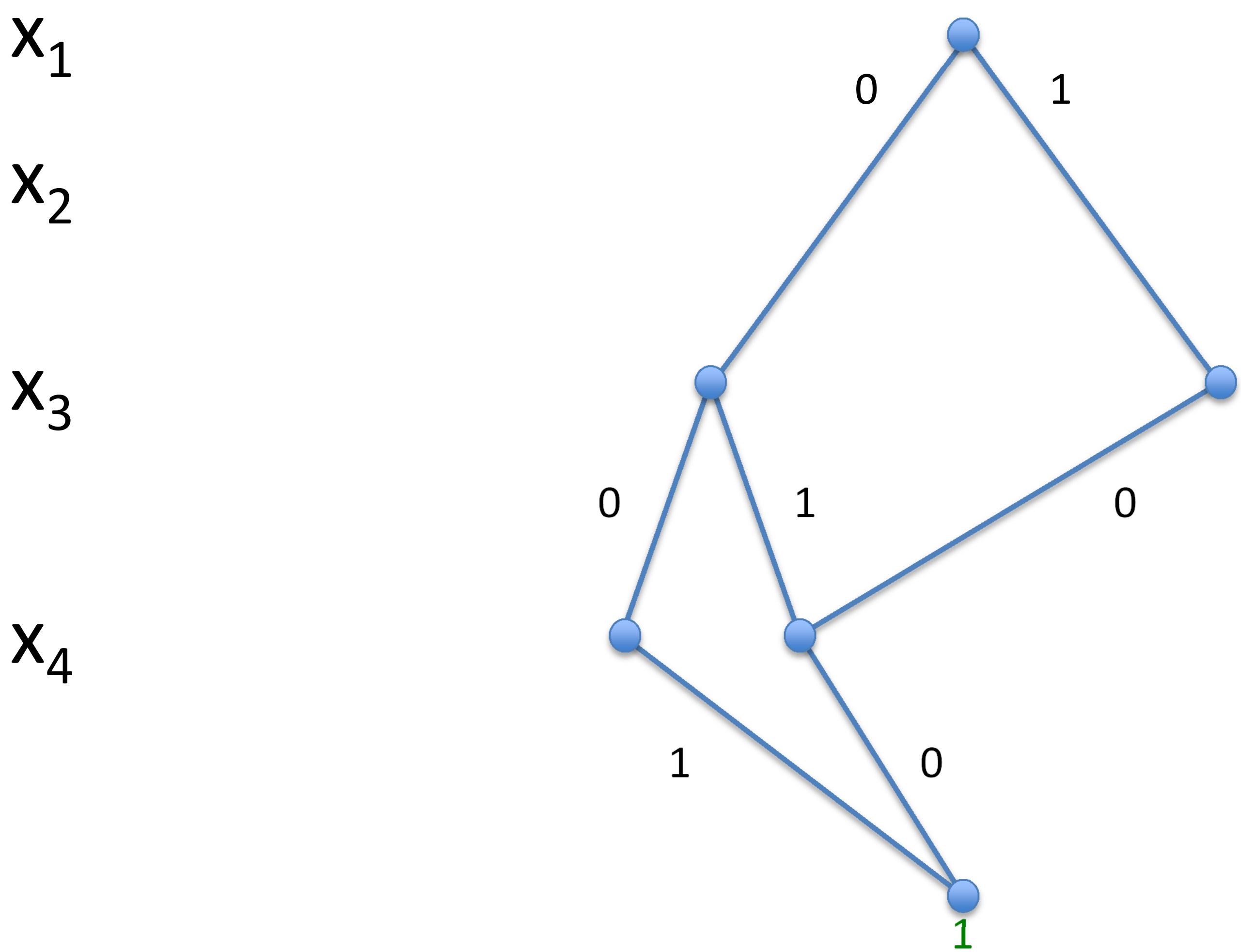
EXAMPLE



REMOVE REDUNDANT EDGES

$\texttt{among}(\{x_1,x_3,x_4\},\{1\},2,2)$

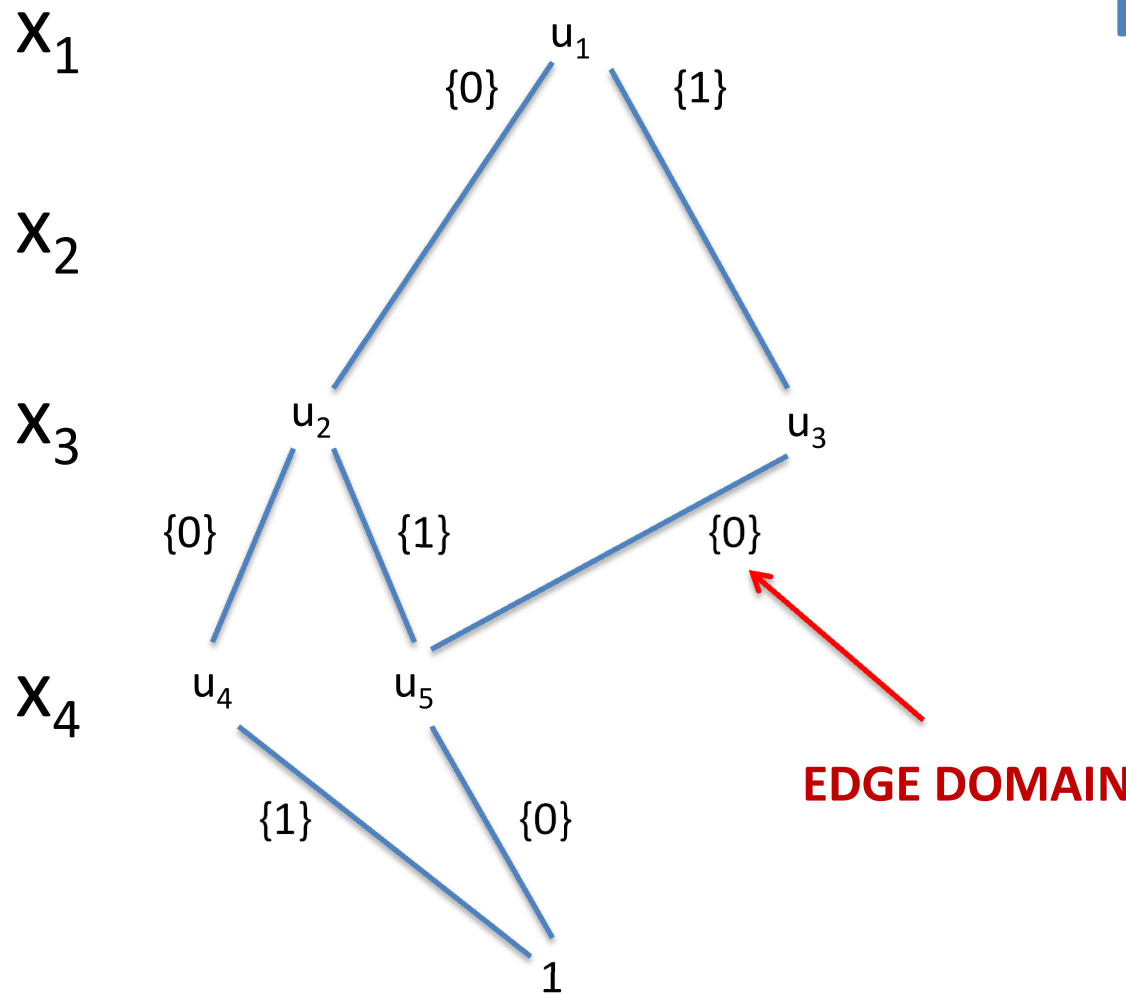
EXAMPLE



REDUCED BDD

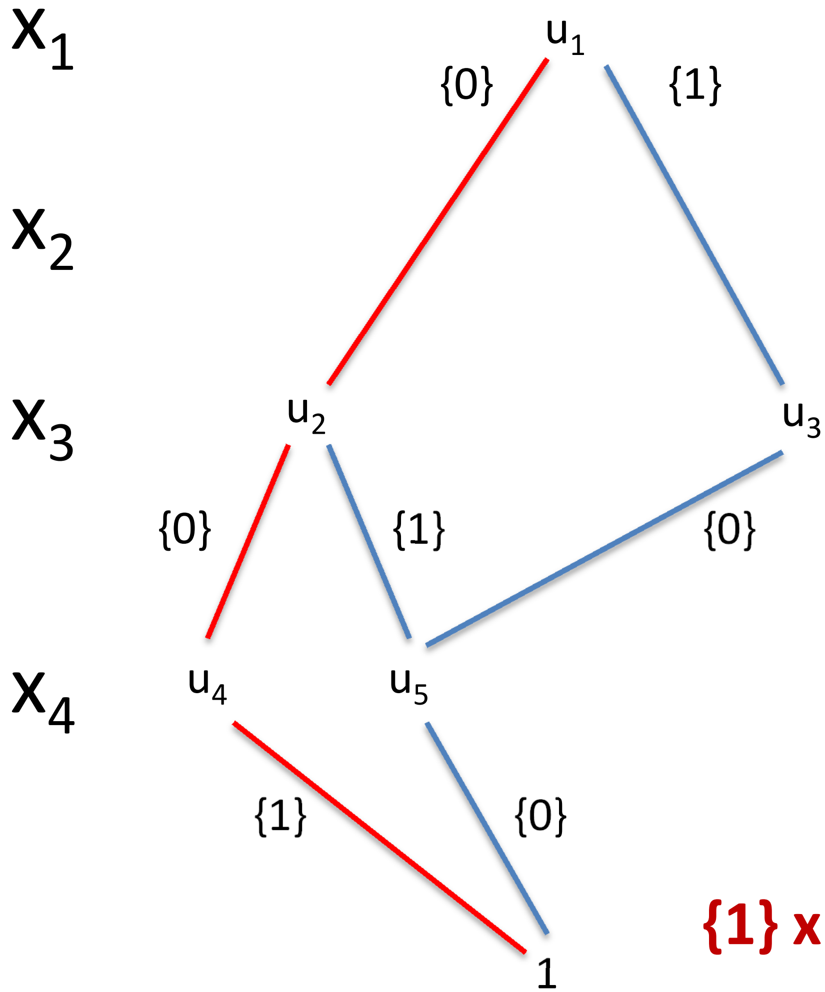
$\texttt{among}(\{x_1,x_3,x_4\},\{1\},2,2)$

EXAMPLE



$\texttt{among}(\{x_1,x_3,x_4\},\{1\},2,2)$

EXAMPLE

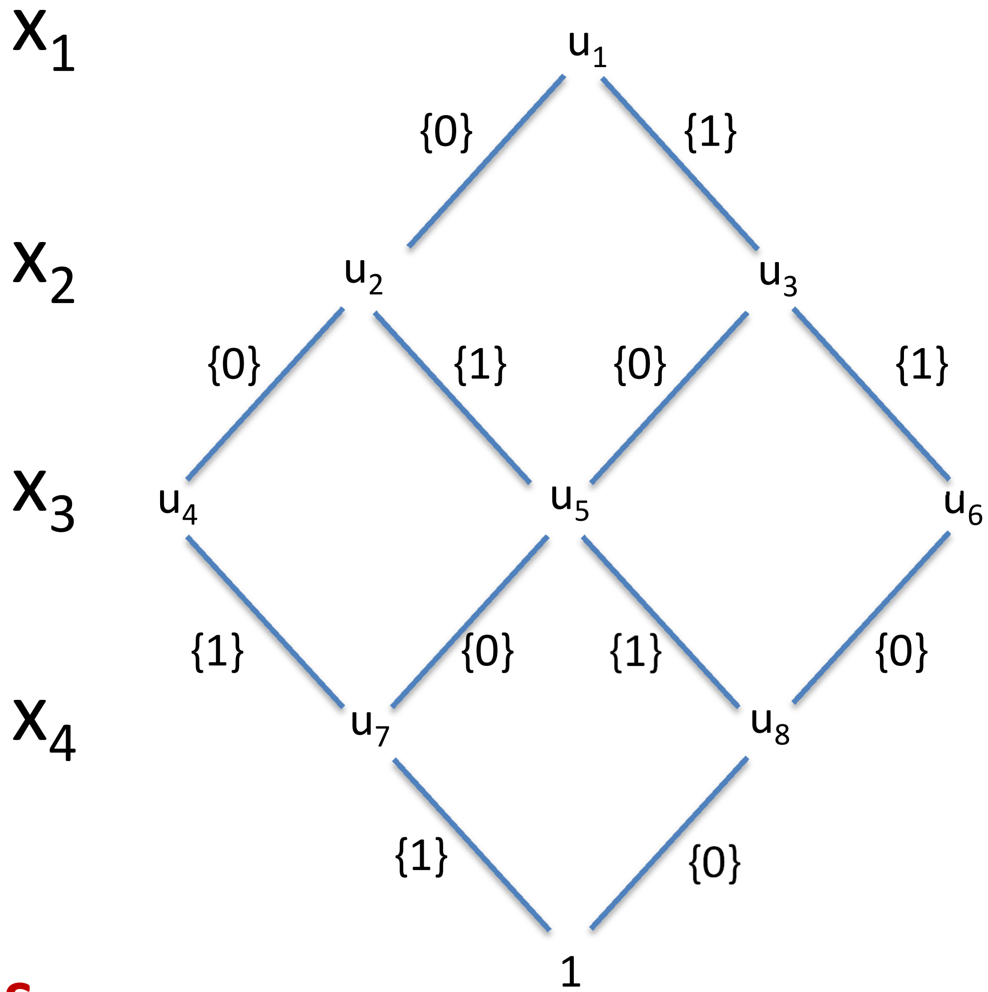


Each path corresponds to a Cartesian product of solutions

{1} x {0,1} x {0} x {0}

$\texttt{among}(\{x_1,x_3,x_4\},\{1\},2,2)$

EXACT BDD



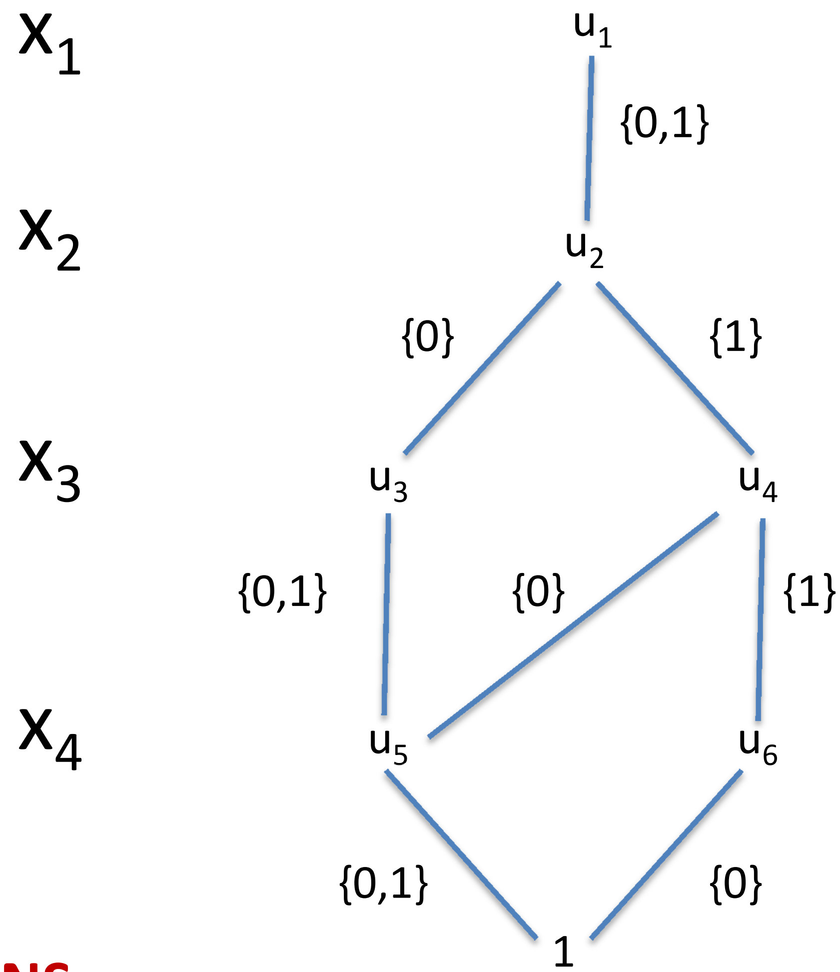
6 SOLUTIONS

NEW CONSTRAINT

$$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$$

RELAXED BDD

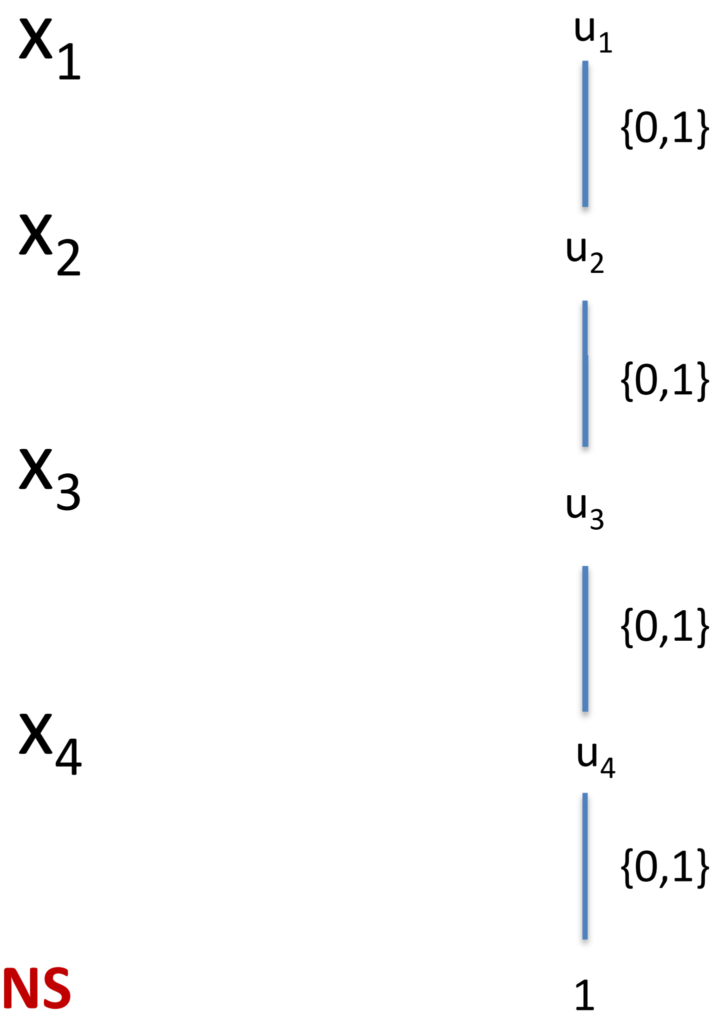
Lets use a BDD of
maximum width 2



14 SOLUTIONS

$$\mathtt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$$

RELAXED BDD

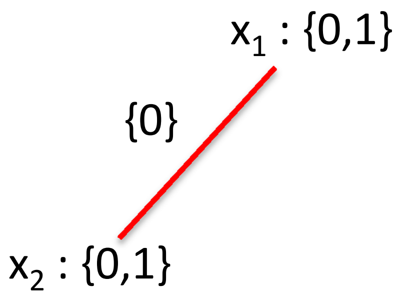


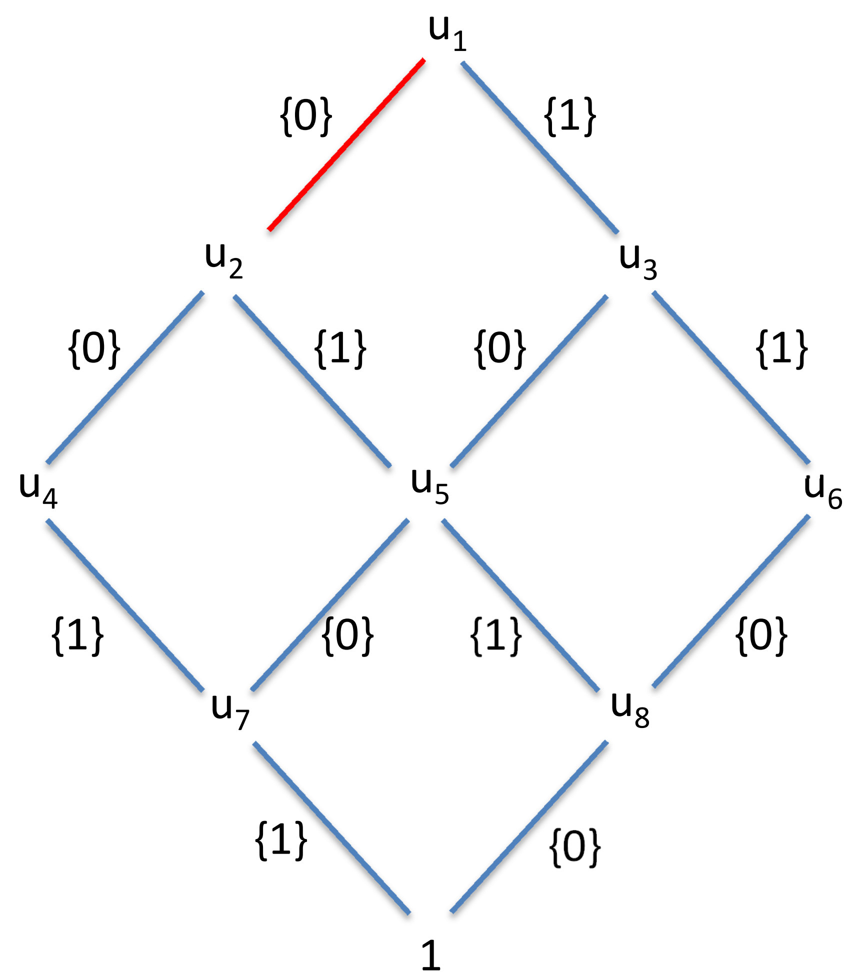
A BDD with maximum **width 1** is just the **domain store**

16 SOLUTIONS

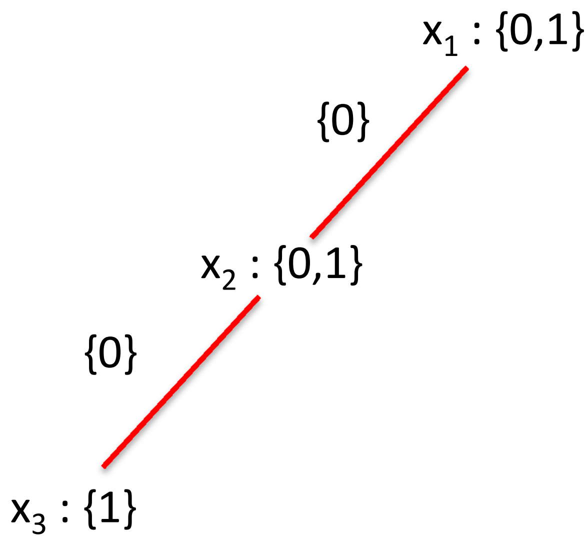
$$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$$

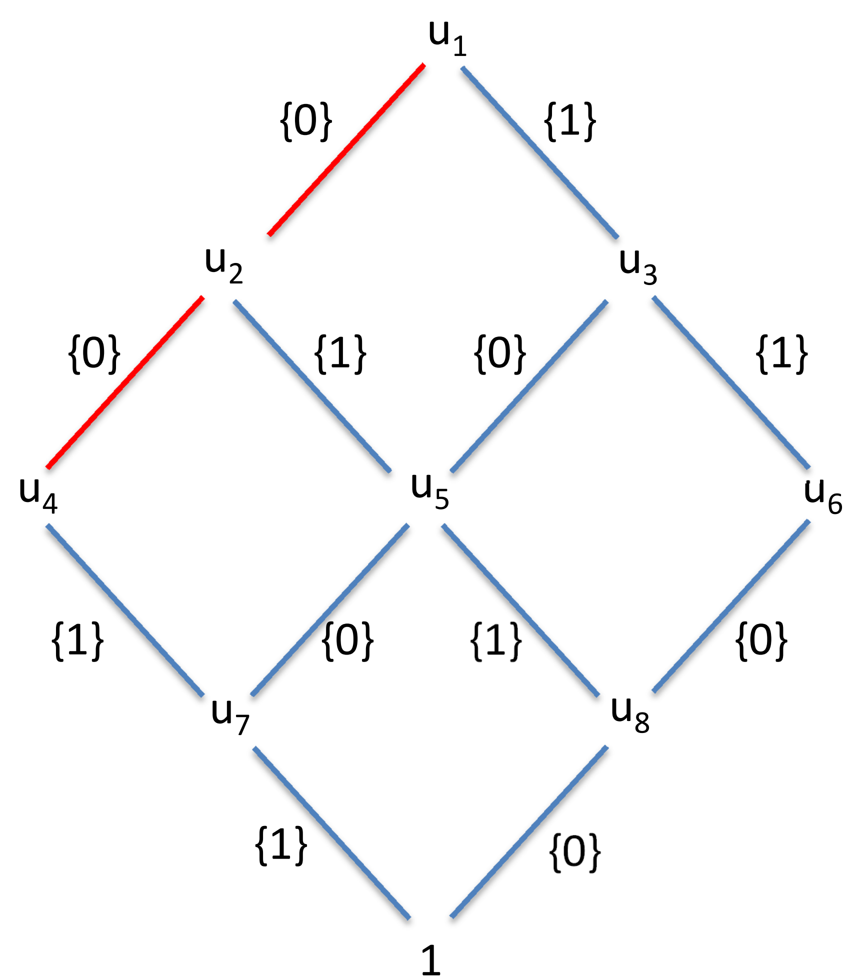
BRANCHING SEARCH



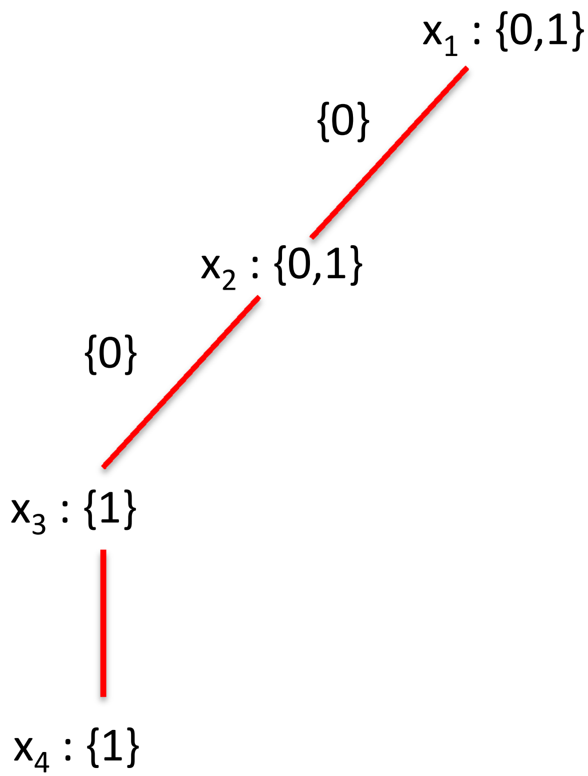


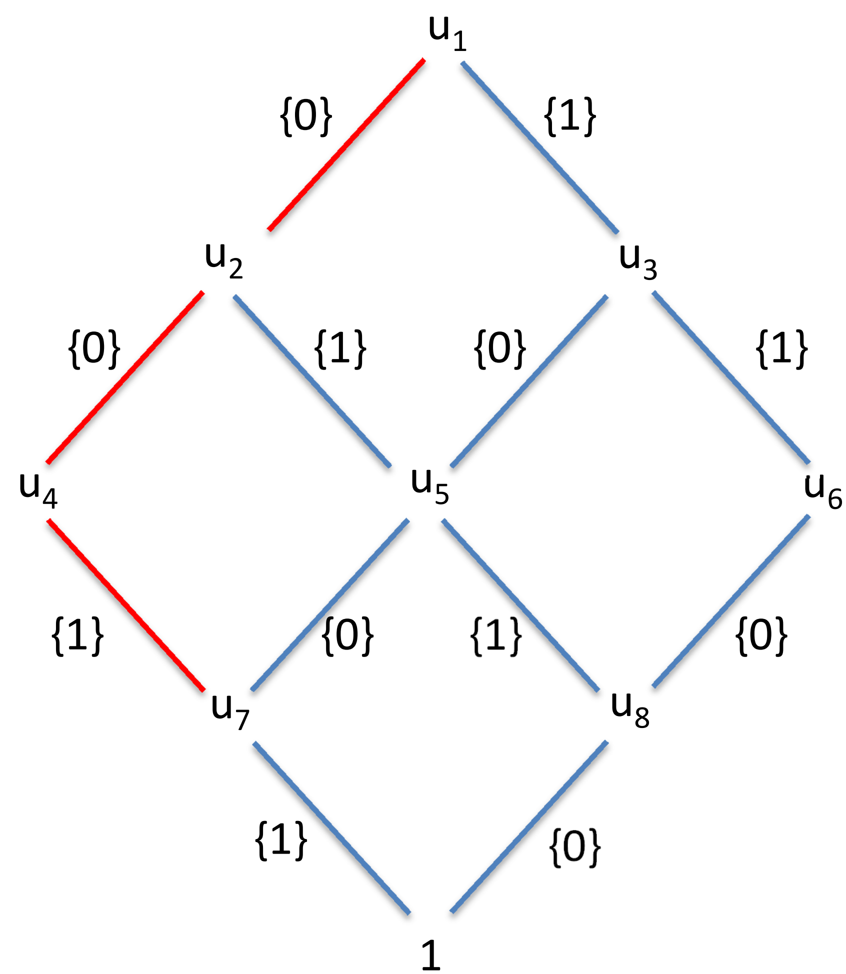
BRANCHING SEARCH



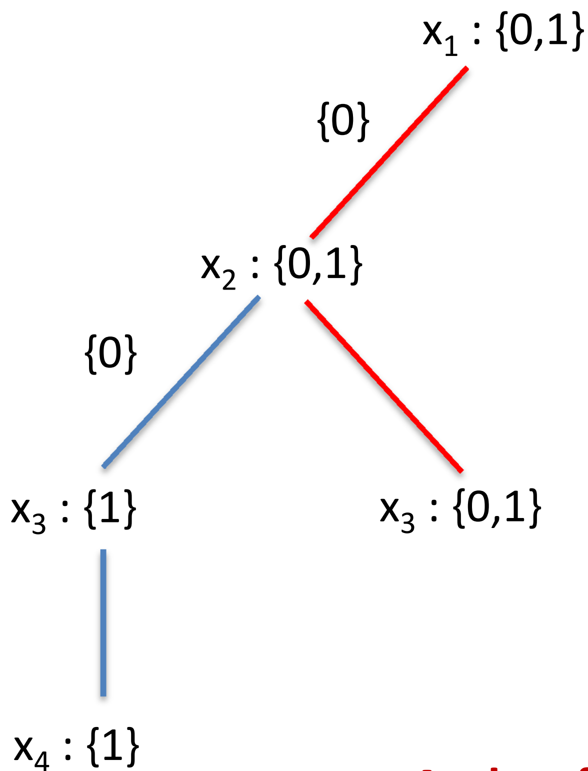


BRANCHING SEARCH



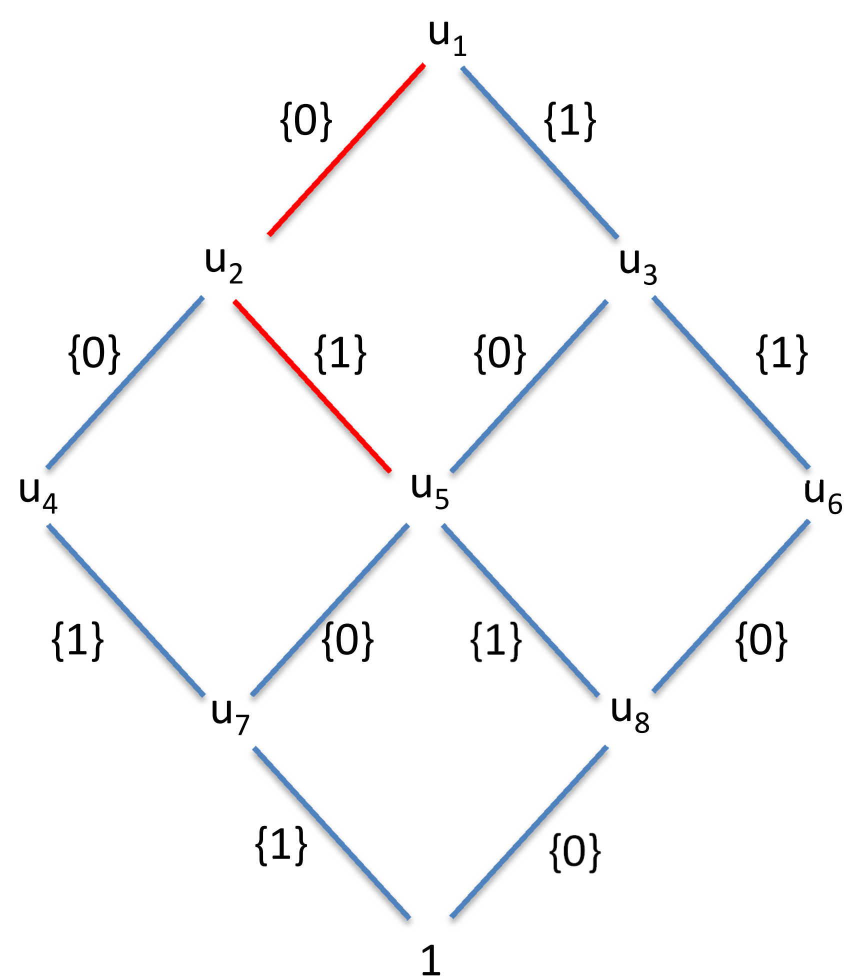


BRANCHING SEARCH



And so forth.

Less branching than with domain store.



Complete (domain store) filter for `among`

- Special case: `among`($\{x_1,\ldots,x_n\},\{1\},L,U$) where $D(x_i) \subseteq \{0,1\}$
  - Let $SP=|\{x_i : 0 \in D(x_i)\}|$ and $LP=|\{x_i : 1 \in D(x_i)\}|$

✗ If LP < L or SP > U then inconsistent

✓ If LP=L then filter 0 from non-singleton domains

✓ If SP=U then filter 1 from non-singleton domains

- General case can be reduced to special case
  - Use `element` constraint

Propagation in MDDs

- Propagate in a MDD using
  - edge domain filtering, and
  - refinement (node splitting)
  - without exceeding maximum width

- Example:
  - We will propagate `among`($\{x_1,x_2,x_3,x_4\},\{1\},2,2$) through a BDD of maximum width 3

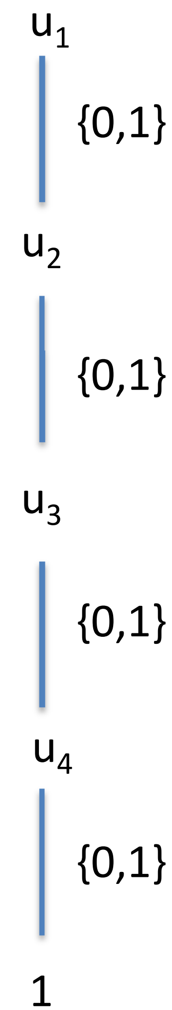
EXAMPLE

Try to filter edge domain (u_1,u_2)

SP using $(u_1,u_2,\{0,1\})$ has length < U
LP using $(u_1,u_2,\{0,1\})$ has length > L

Can't filter

Path lengths are from the root u_1 to the sink 1



L

U

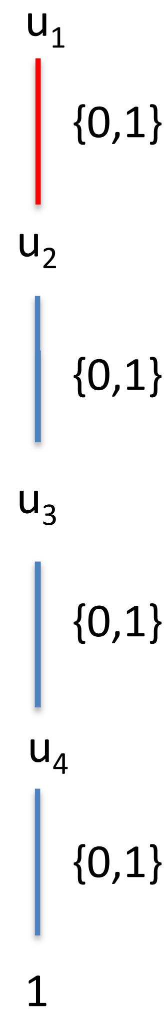
$$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$$

EXAMPLE

Split u_2?

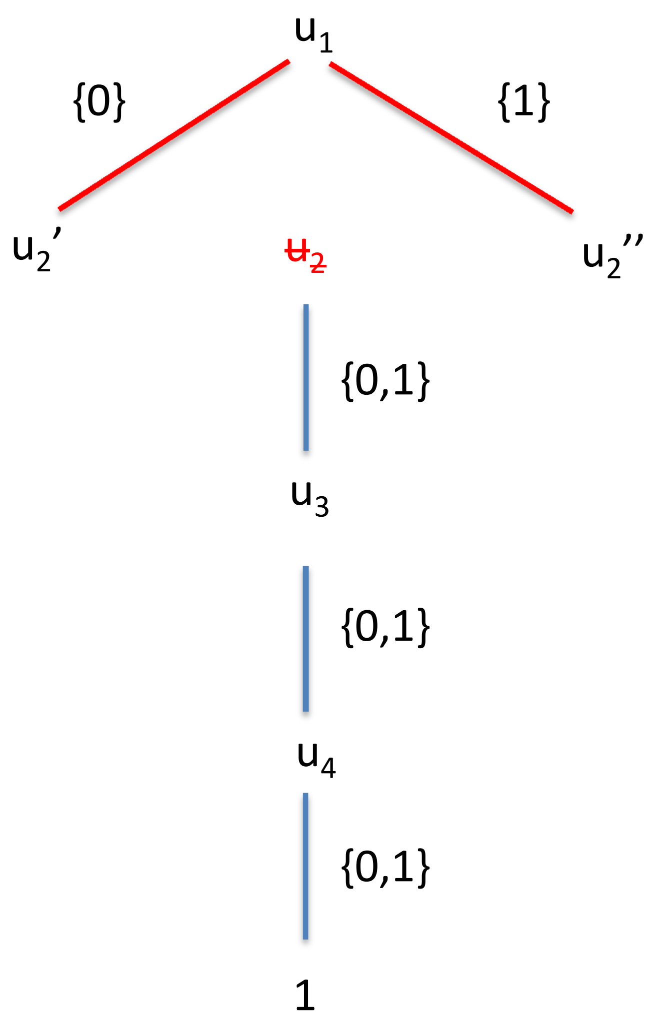
SP using $(u_1,0) = 0$
SP using $(u_1,1) = 1$

Incoming edge-value pairs are not equivalent: so split u_2



$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$

EXAMPLE

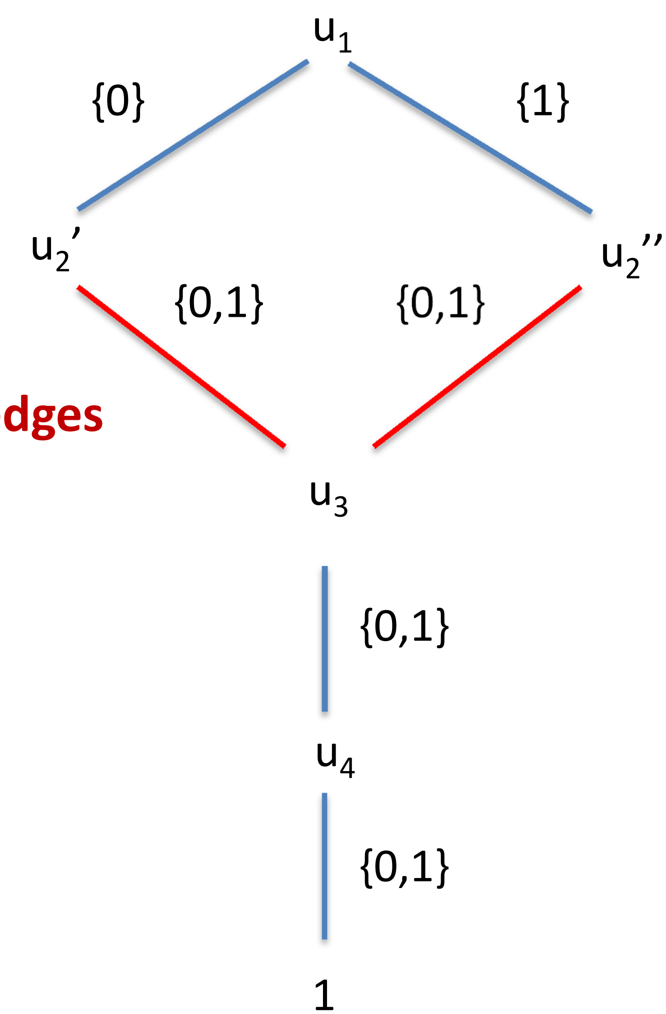


SPLIT u_2 into two classes (less than maximum width)

$$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$$

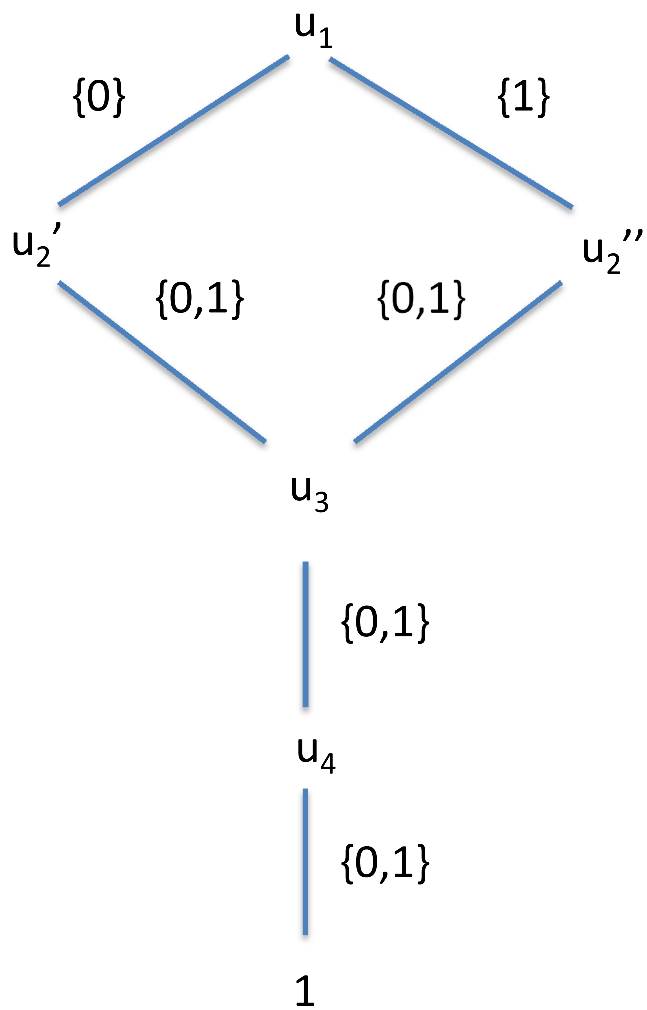
EXAMPLE

DUPLICATE outgoing edges



$$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$$

EXAMPLE

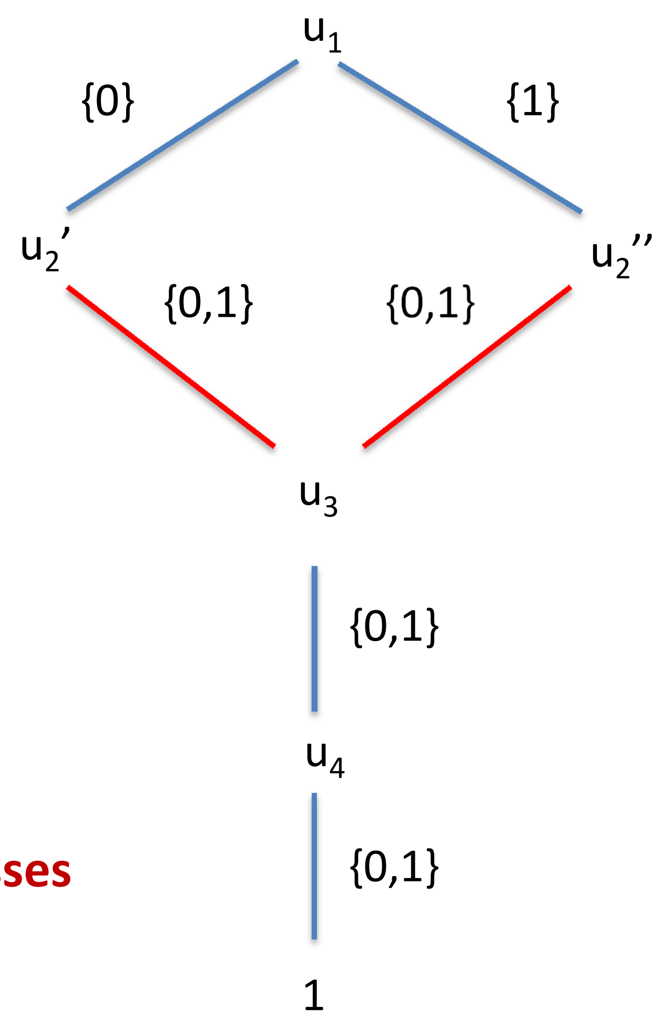


Filter edge domains (u_2',u_3) and (u_2'',u_3)

(No filtering possible)

$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$

EXAMPLE



Split u_3?

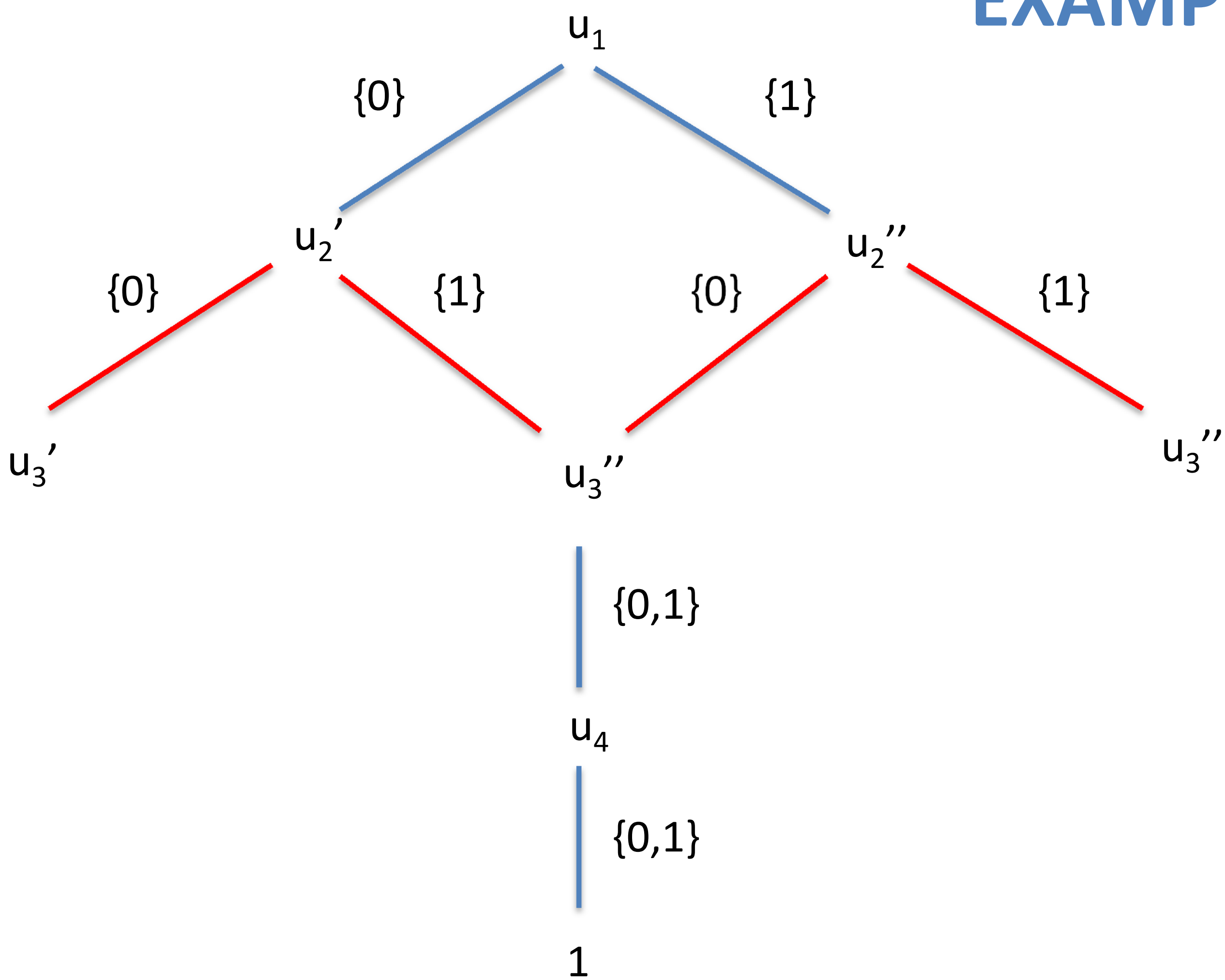
SP $(u_2',u_3,0) = 0$
SP $(u_2',u_3,1) = 1$
SP $(u_2'',u_3,0)= 1$
SP $(u_2'',u_3,1)= 2$

Split u_3 into 3 equivalence classes

$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$

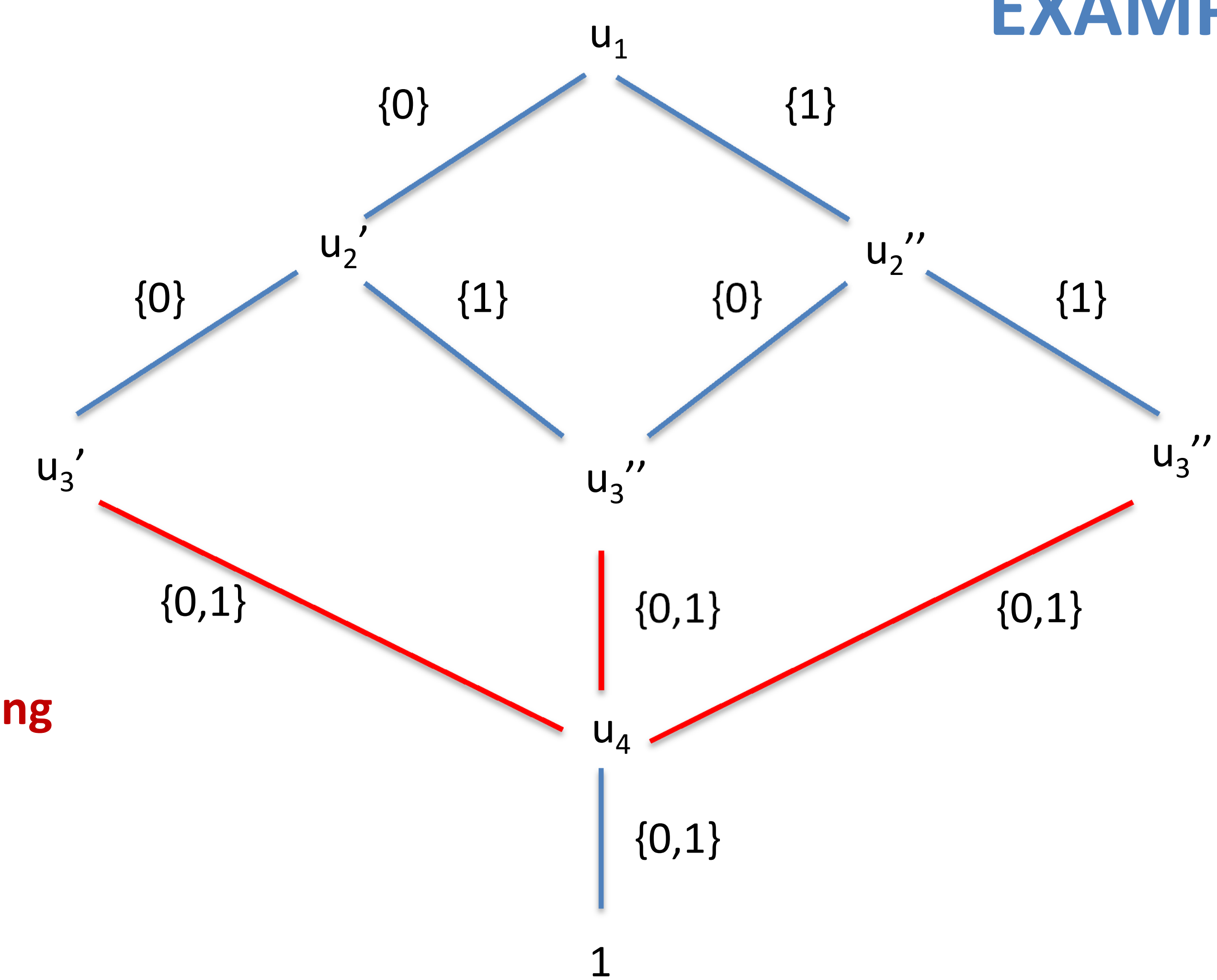
EXAMPLE

Split u_3 into 3 equivalence classes



$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$

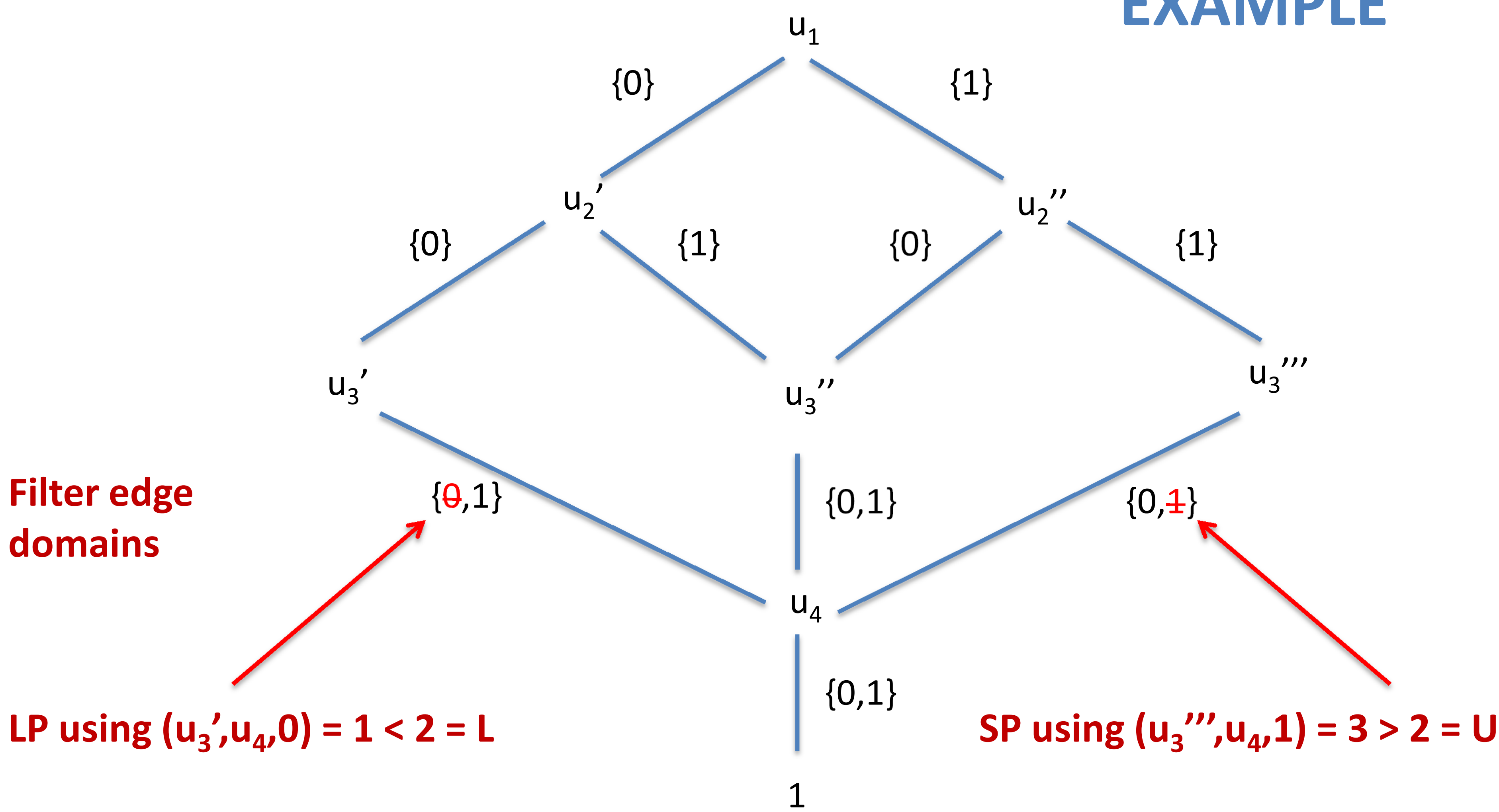
EXAMPLE



Duplicate outgoing edges

$$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$$

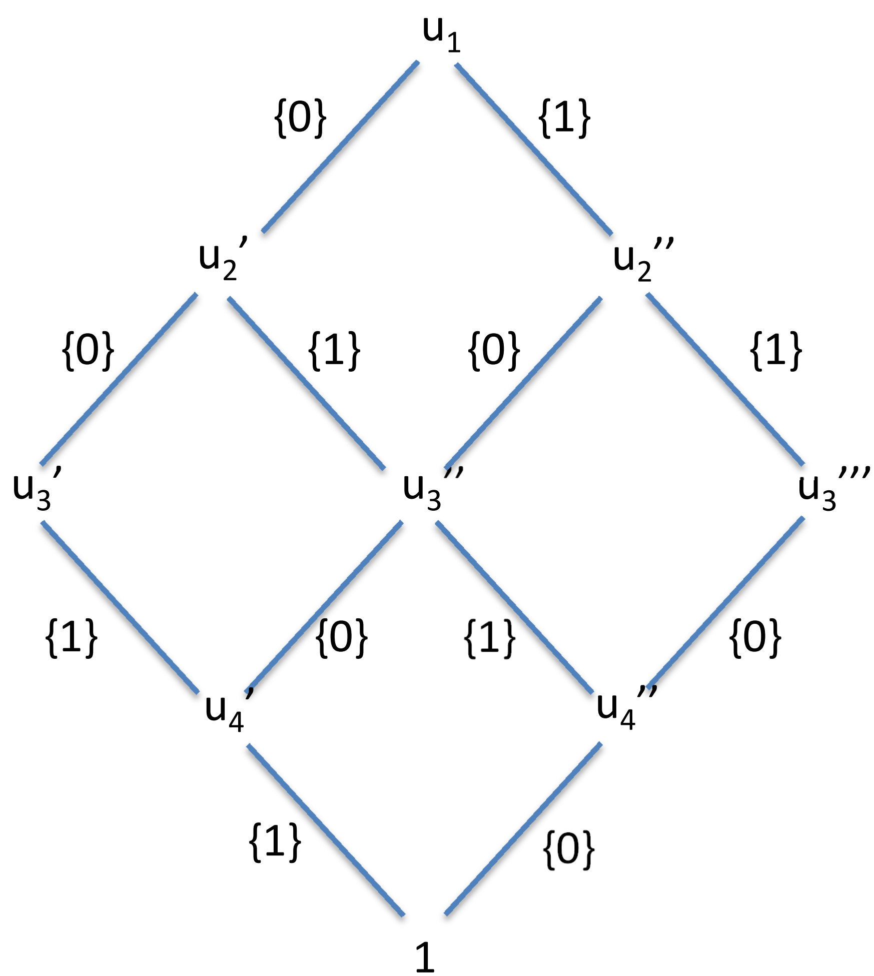
EXAMPLE



$\mathtt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$

EXAMPLE

Continuing…



$\texttt{among}(\{x_1,x_2,x_3,x_4\},\{1\},2,2)$

Approximate equivalence

- Example: edge-value equivalence was **exact**
  - Problem: a few nodes “consume” BDD when processing a constraint
  - Want intra-constraint **diversification**
  - Want inter-constraint **diversification**
- One solution: **approximate** equivalence
  - Edge-value pairs are equivalent if SPs/LPs differ by at most some threshold value

EXPERIMENTS

Problem instances

- Nurse rostering instances (horizon *n* days)
  - **Work 4-5 days per week**
  - **Max A days every B days (Max A/B)**
  - **Min C days every D days (Min A/B)**

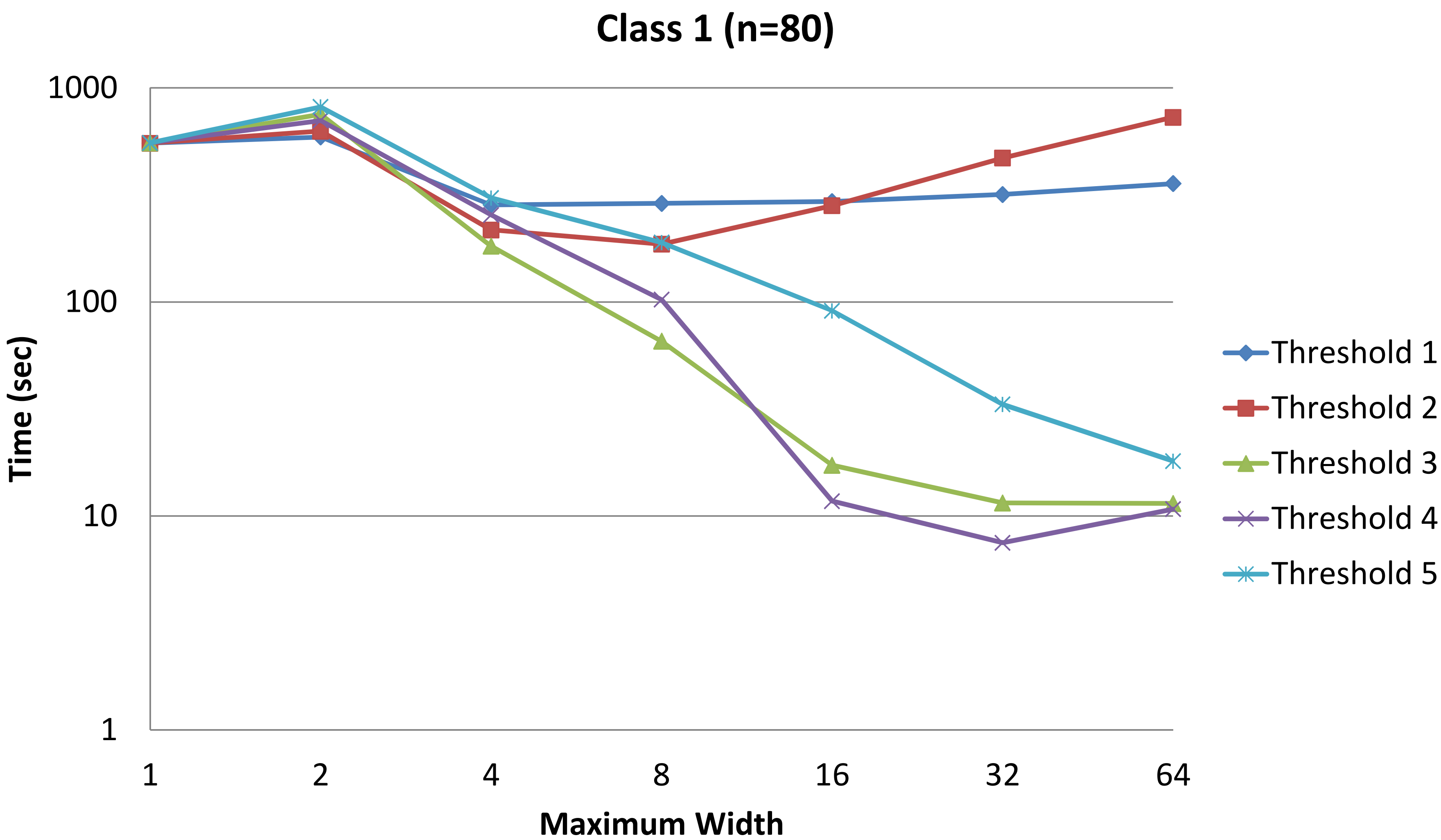
- Class 1 Max 6/8 Min 22/30
- Class 2 Max 6/9 Min 20/30
- Class 3 Max 7/9 Min 22/30

Number of Feasible Solutions

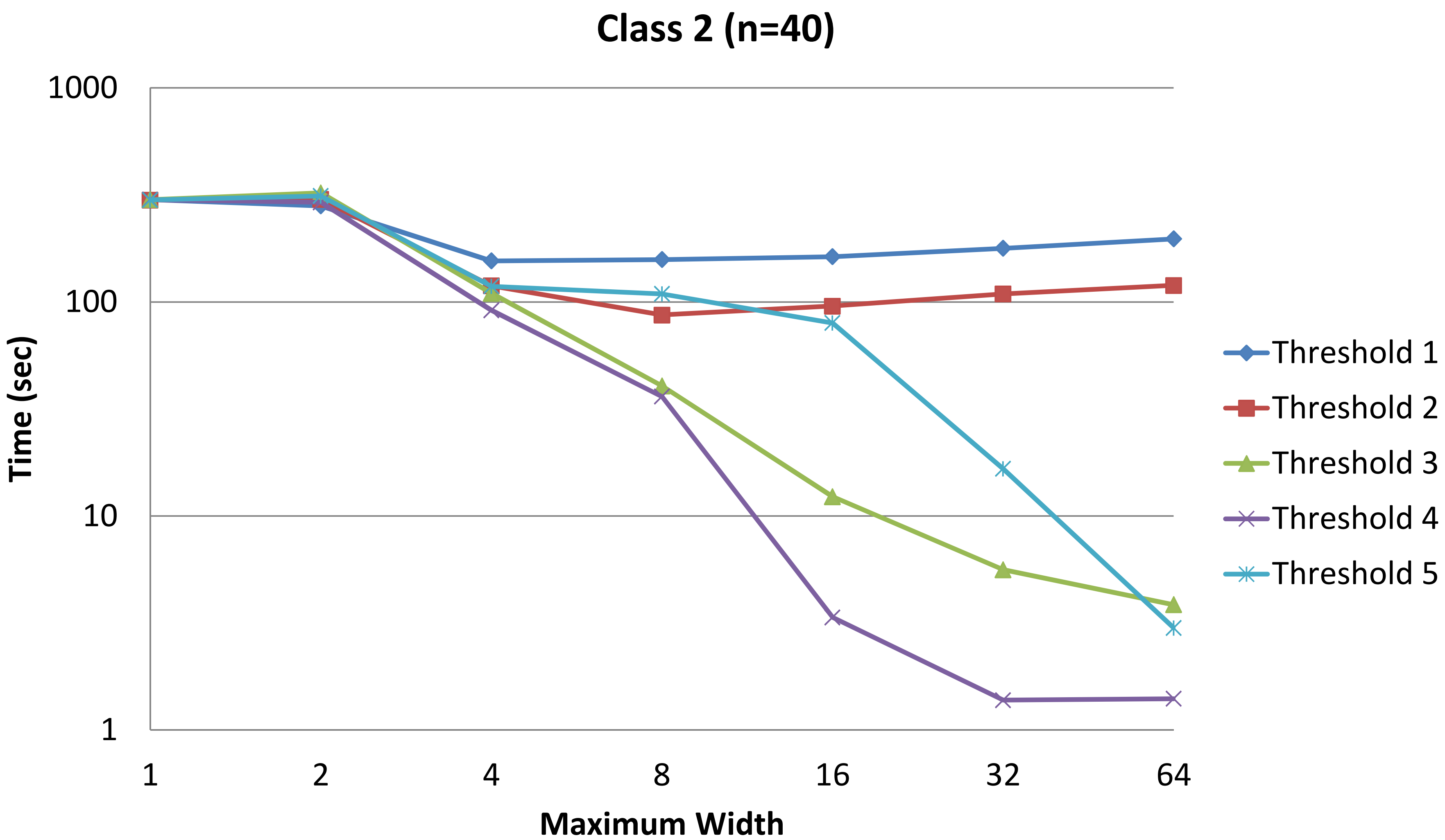
Horizon	40	50	60	70	80
Class 1	2284	4575	6567	2810	730
Class 2	3	3	3	3	3
Class 3	137593	388726	718564	105618	22650

FINDING ALL FEASIBLE SOLUTIONS

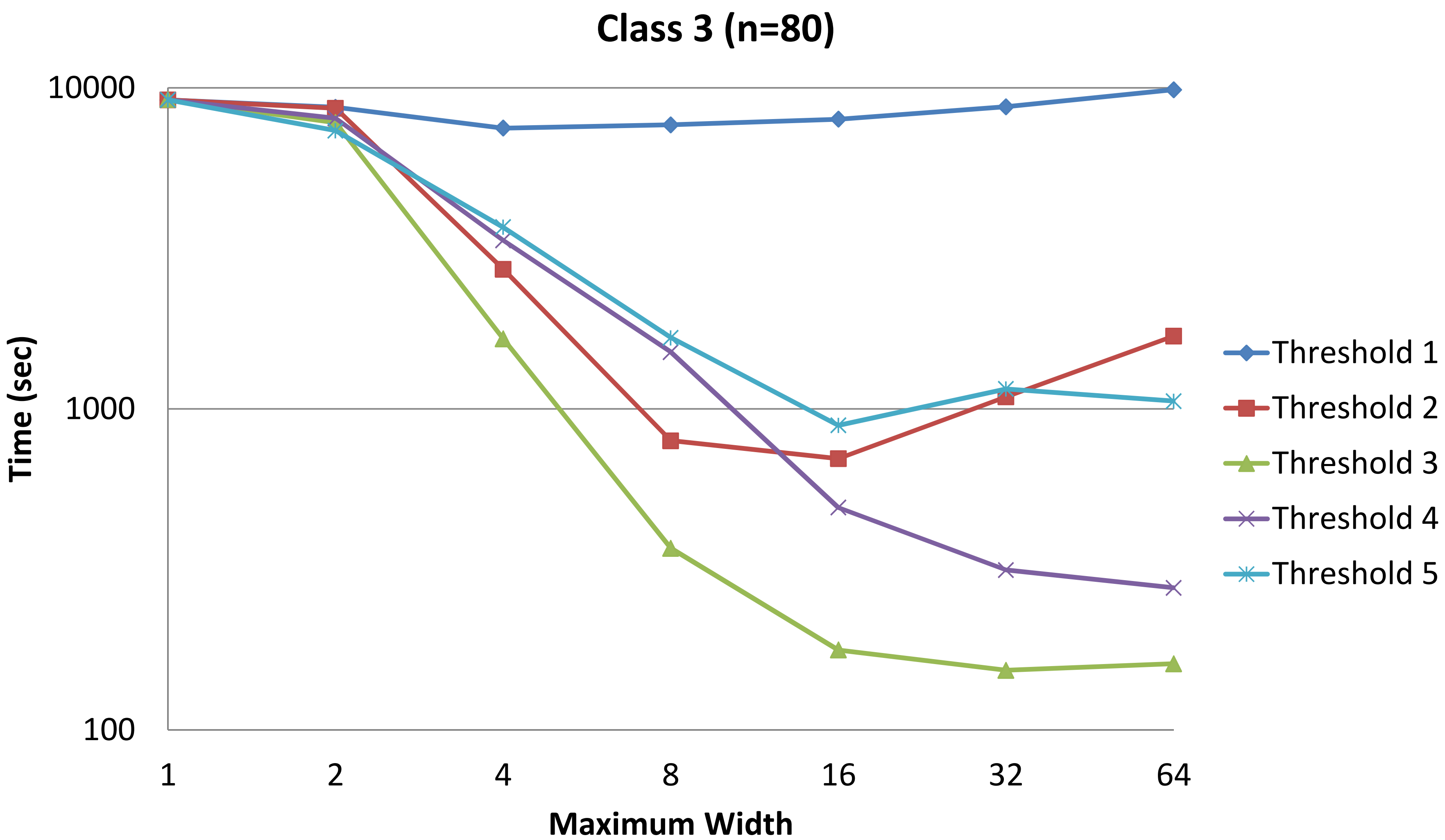
Computation time



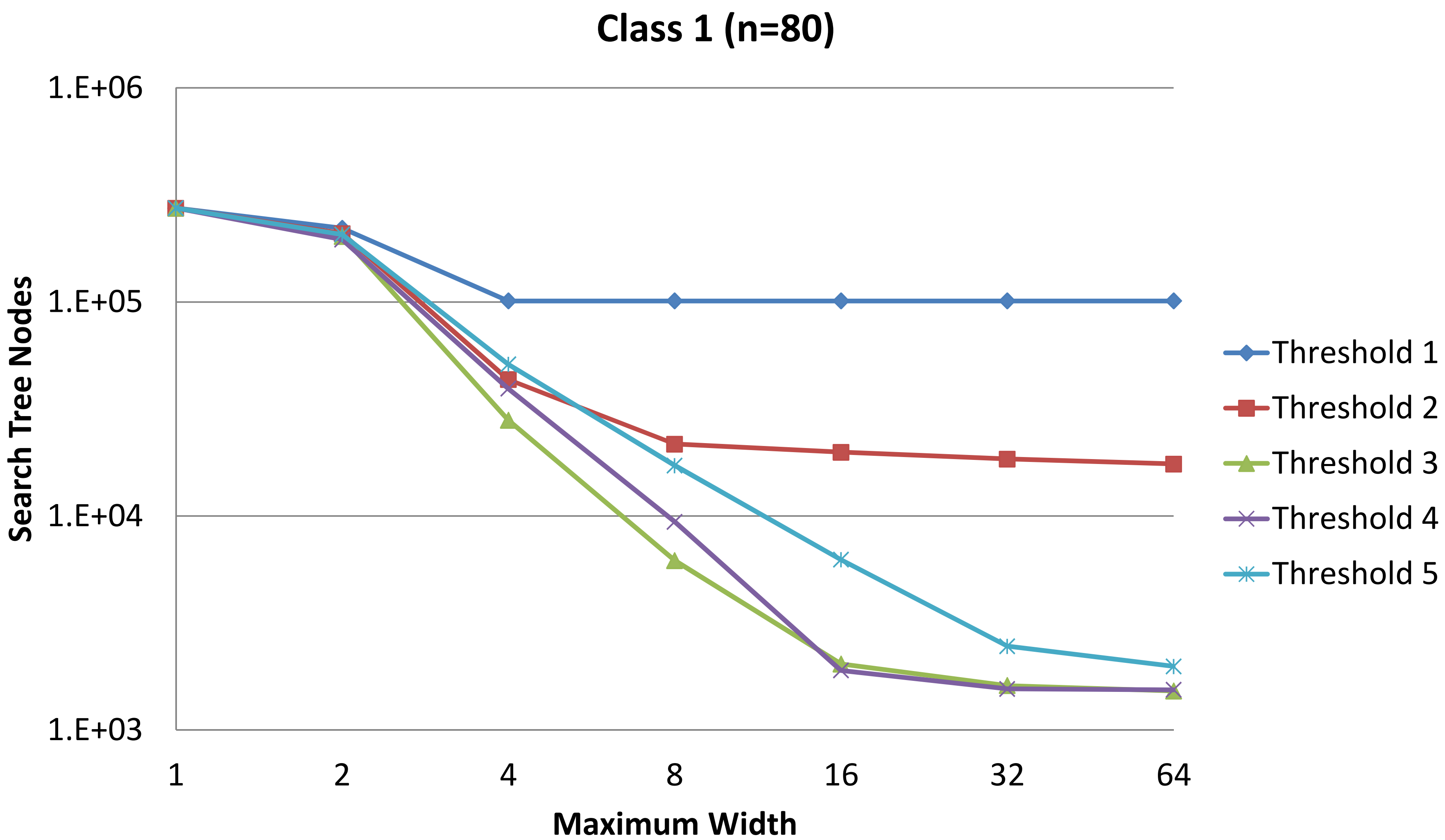
Computation time



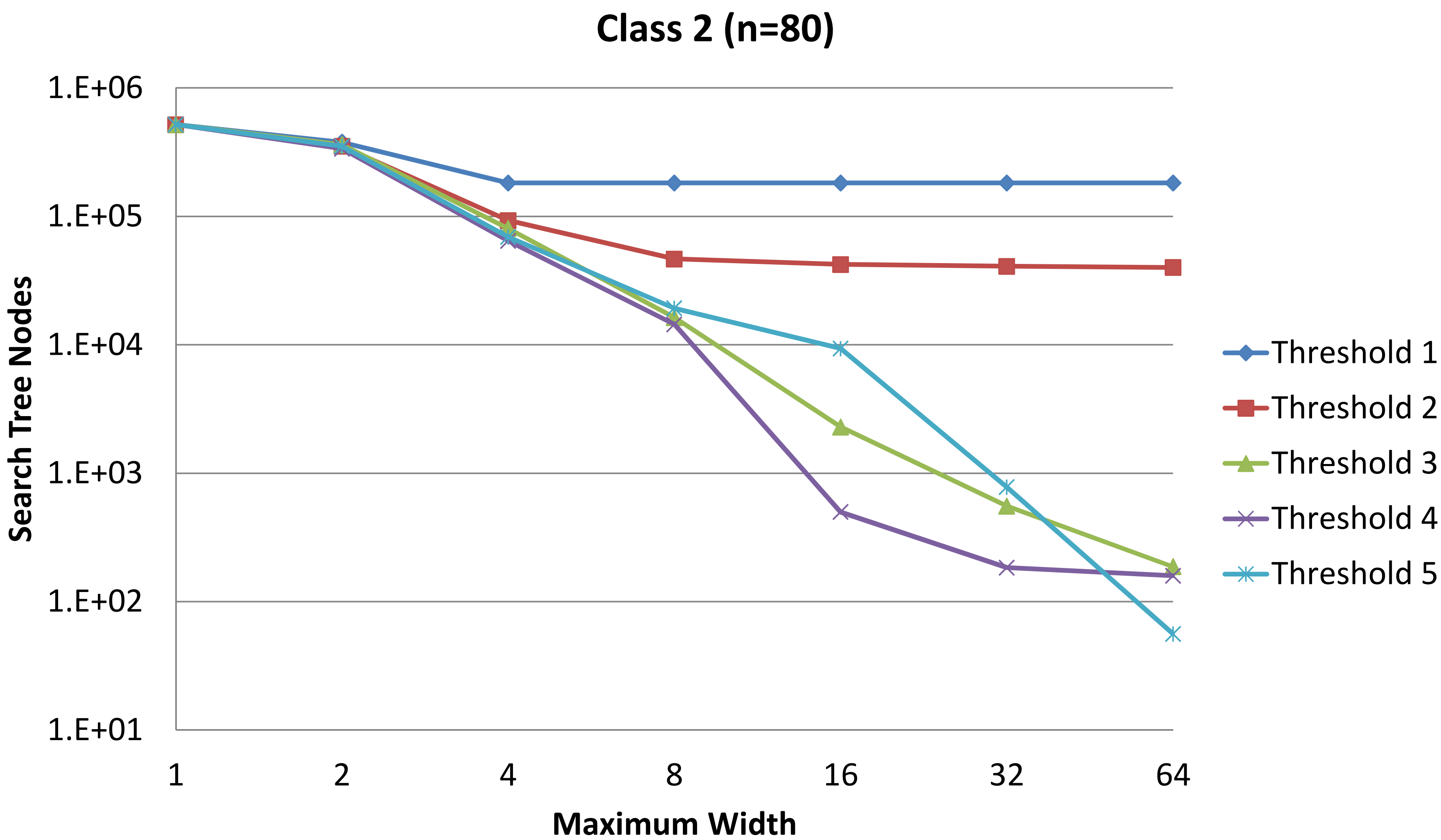
Computation time



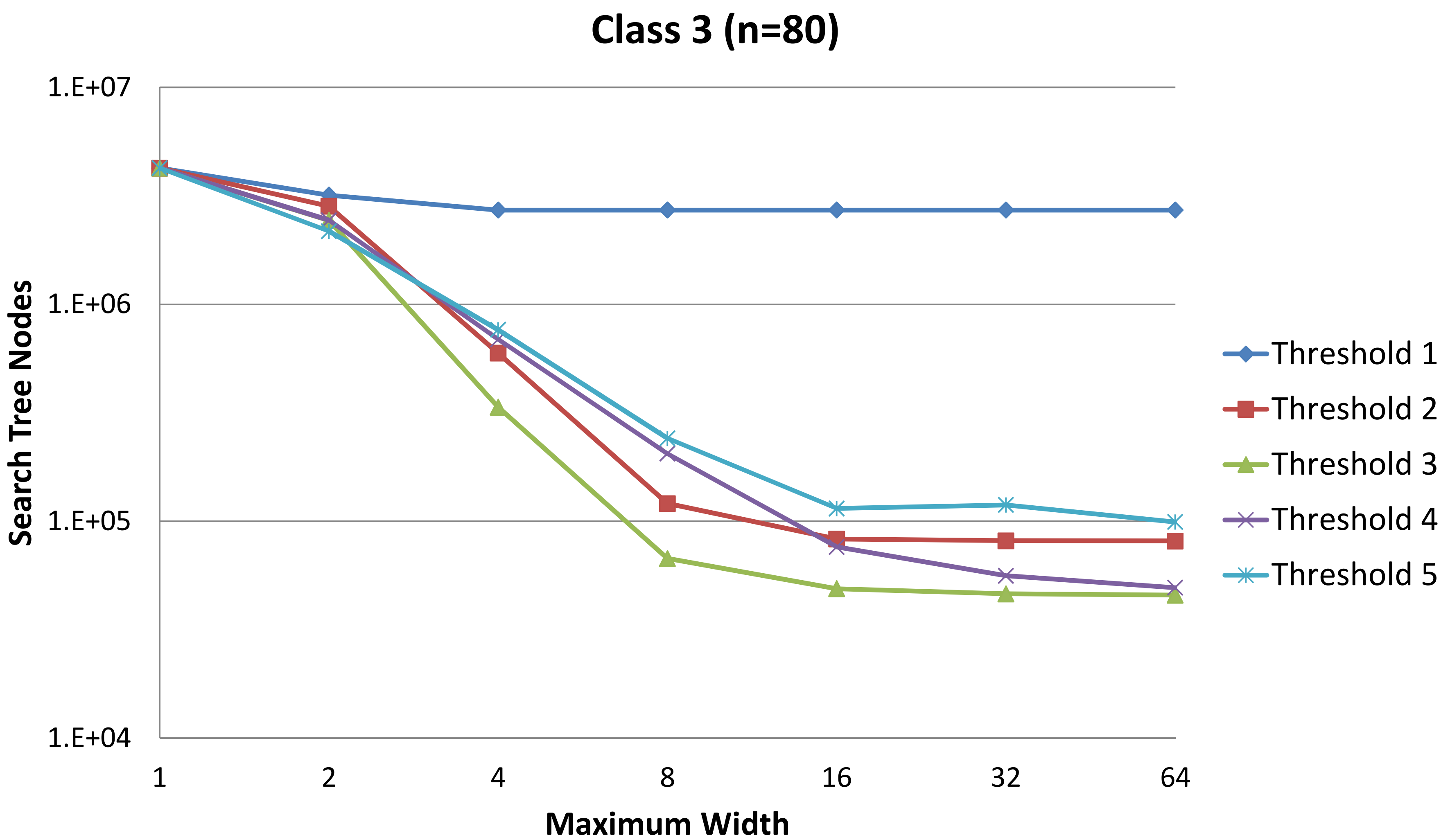
Search tree nodes



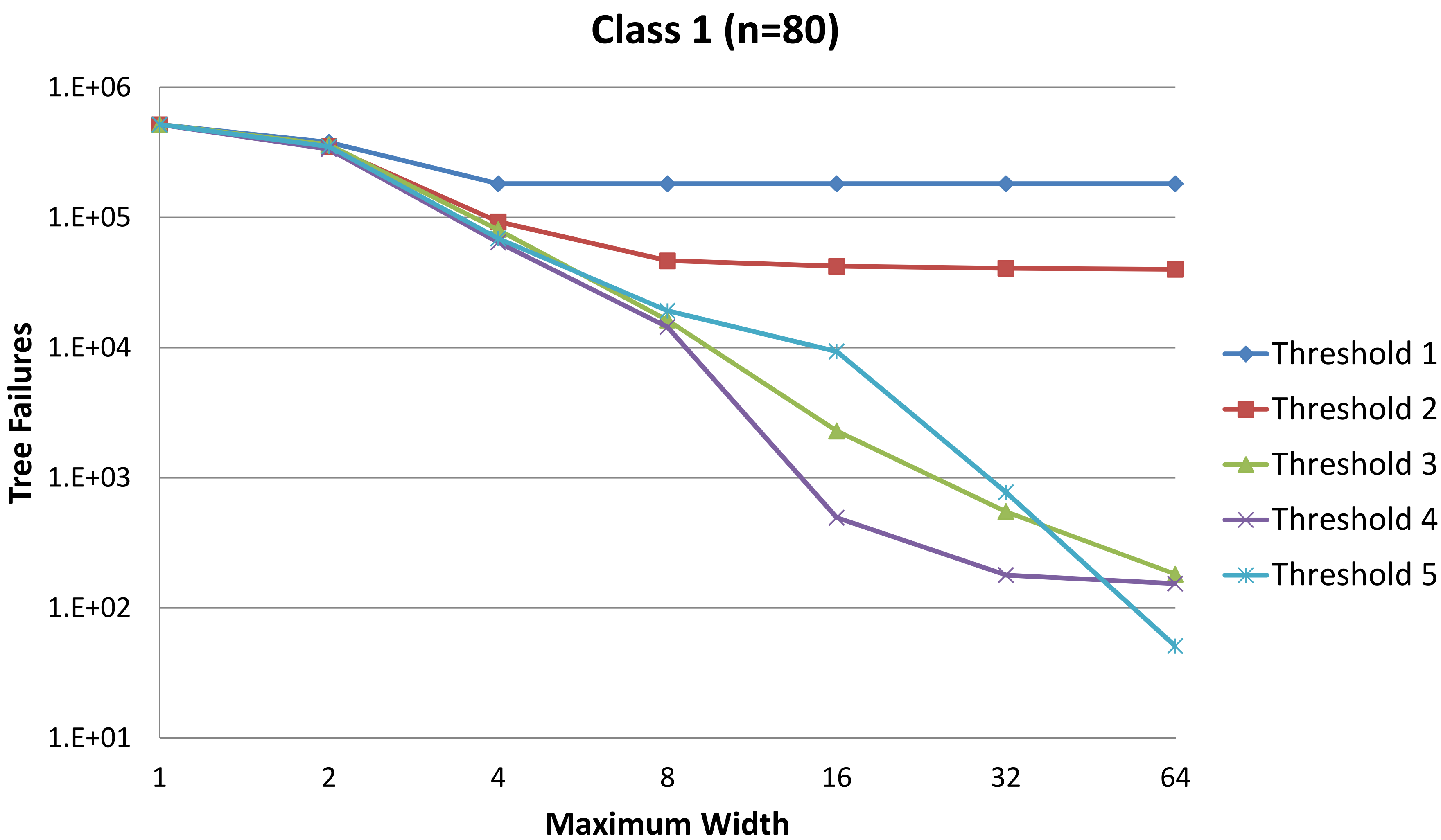
Search tree nodes



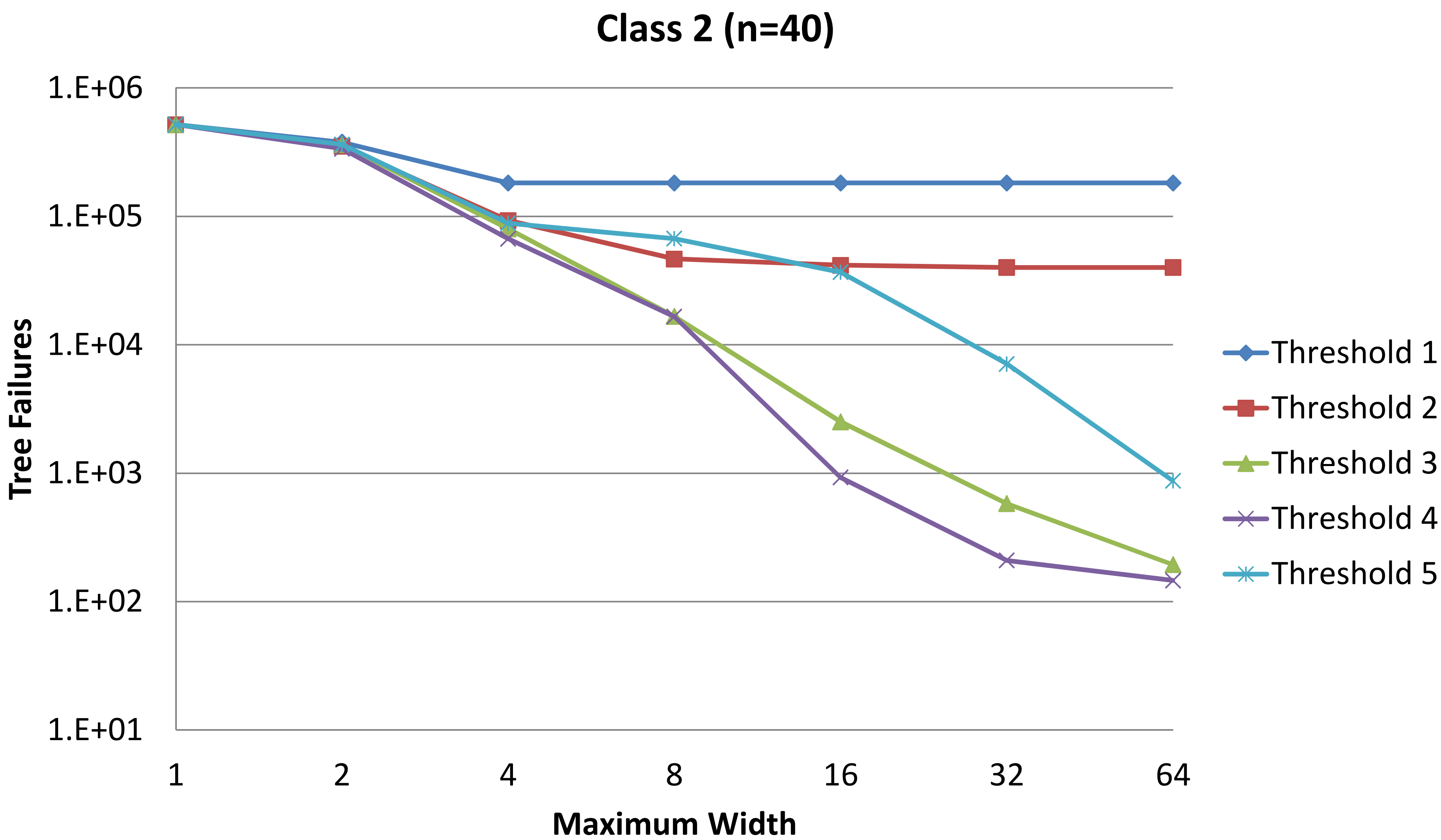
Search tree nodes



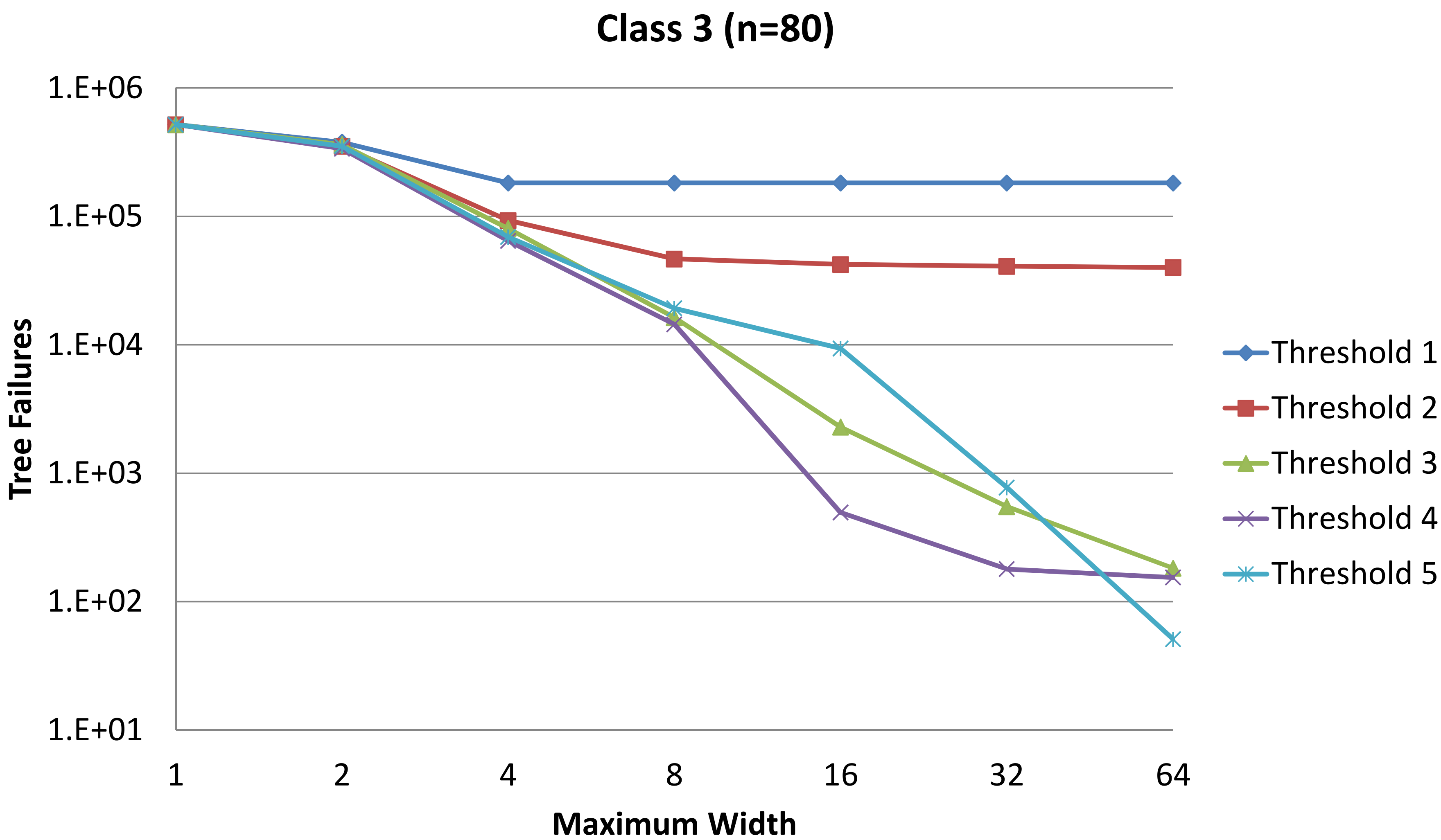
Search tree failures



Search tree failures

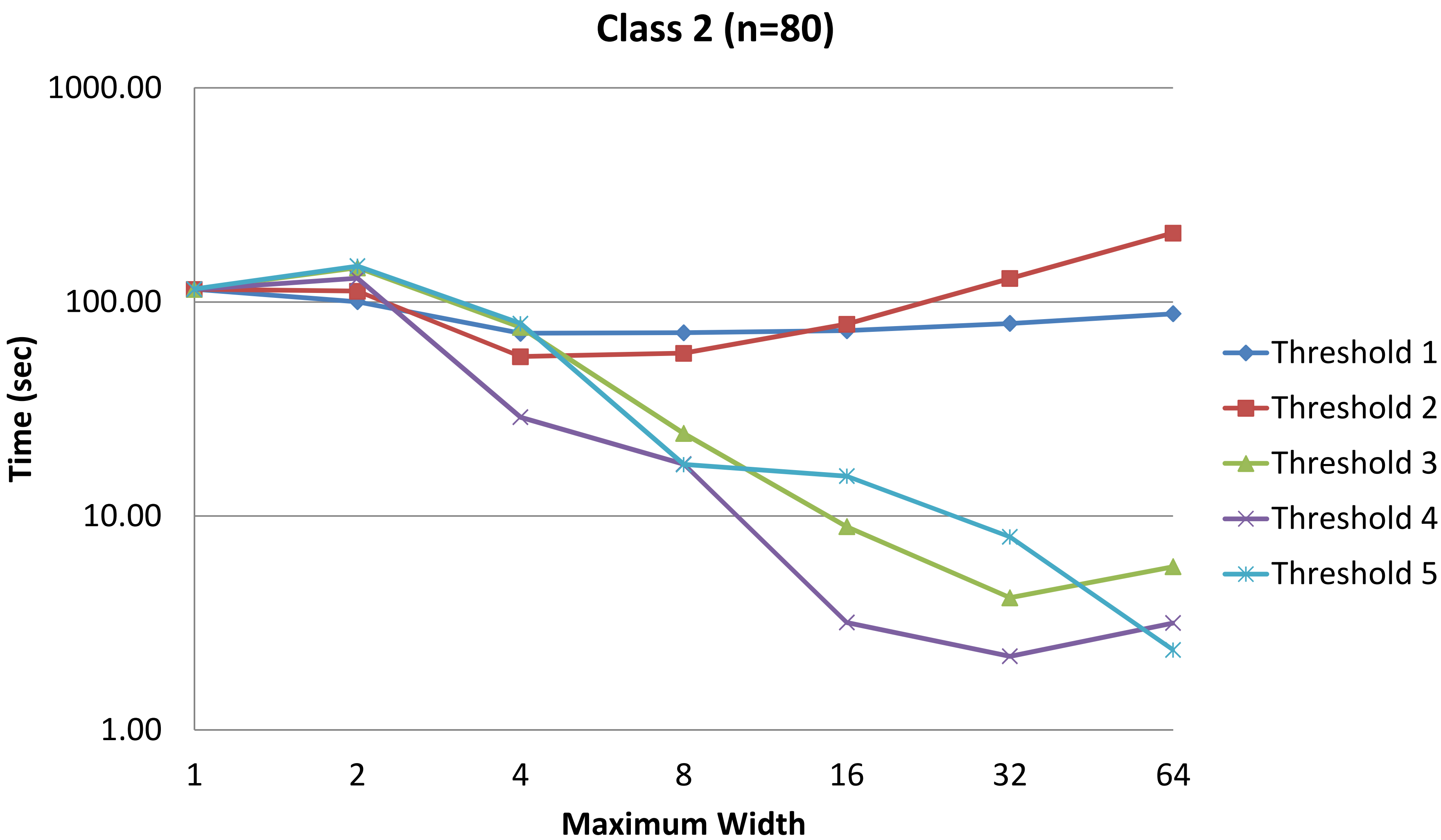


Search tree failures

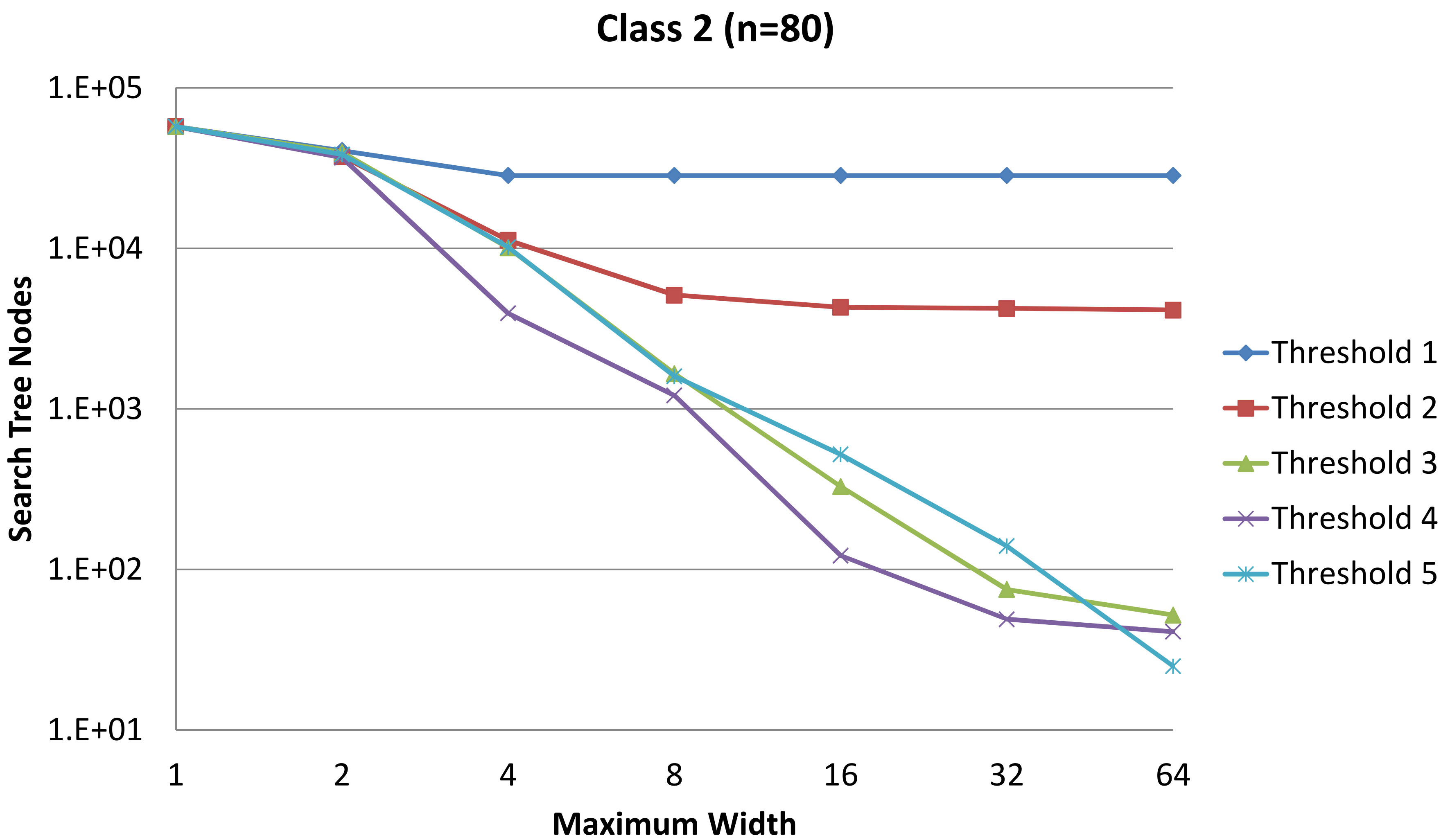


FINDING THE FIRST FEASIBLE

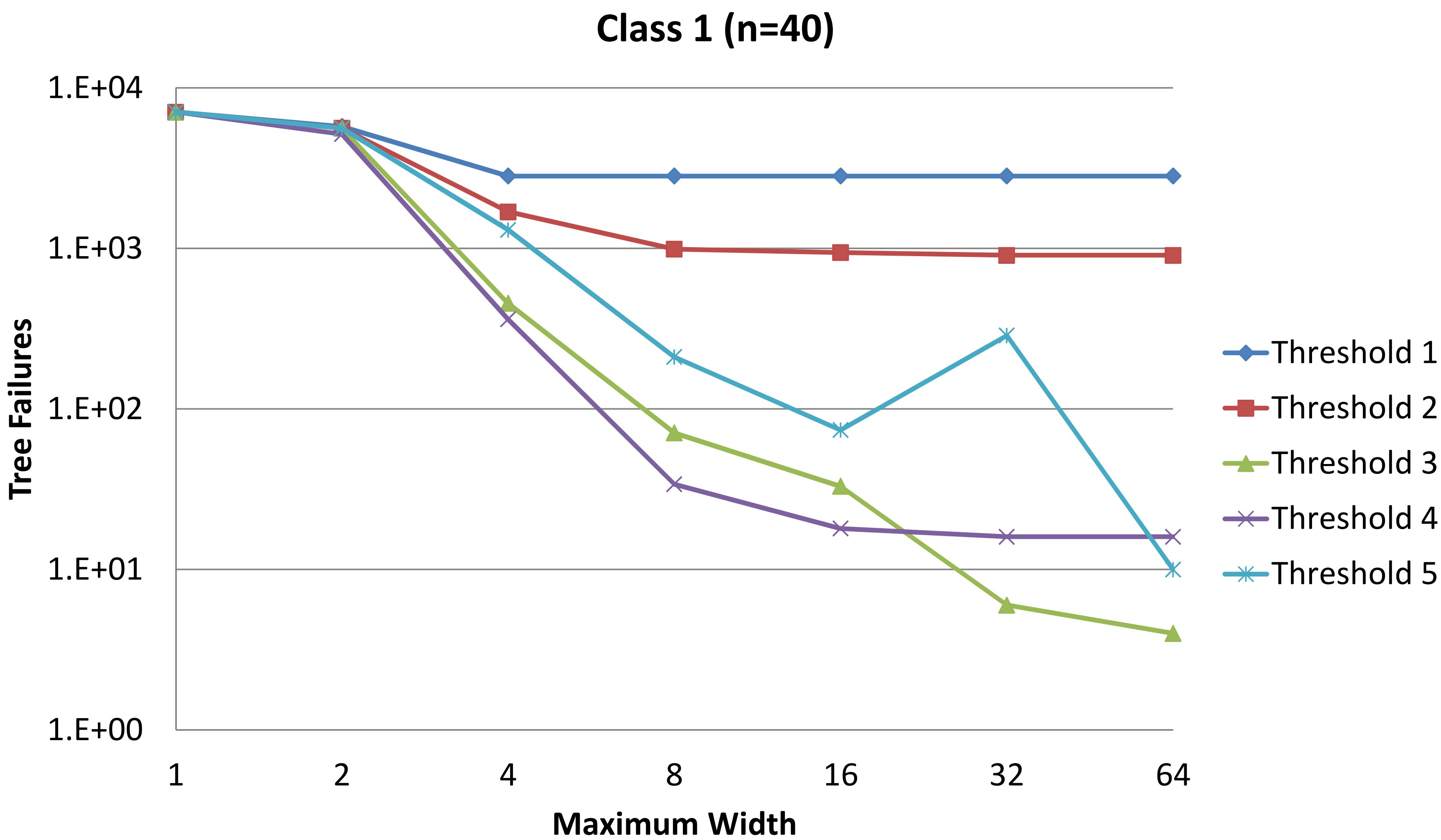
Computation time



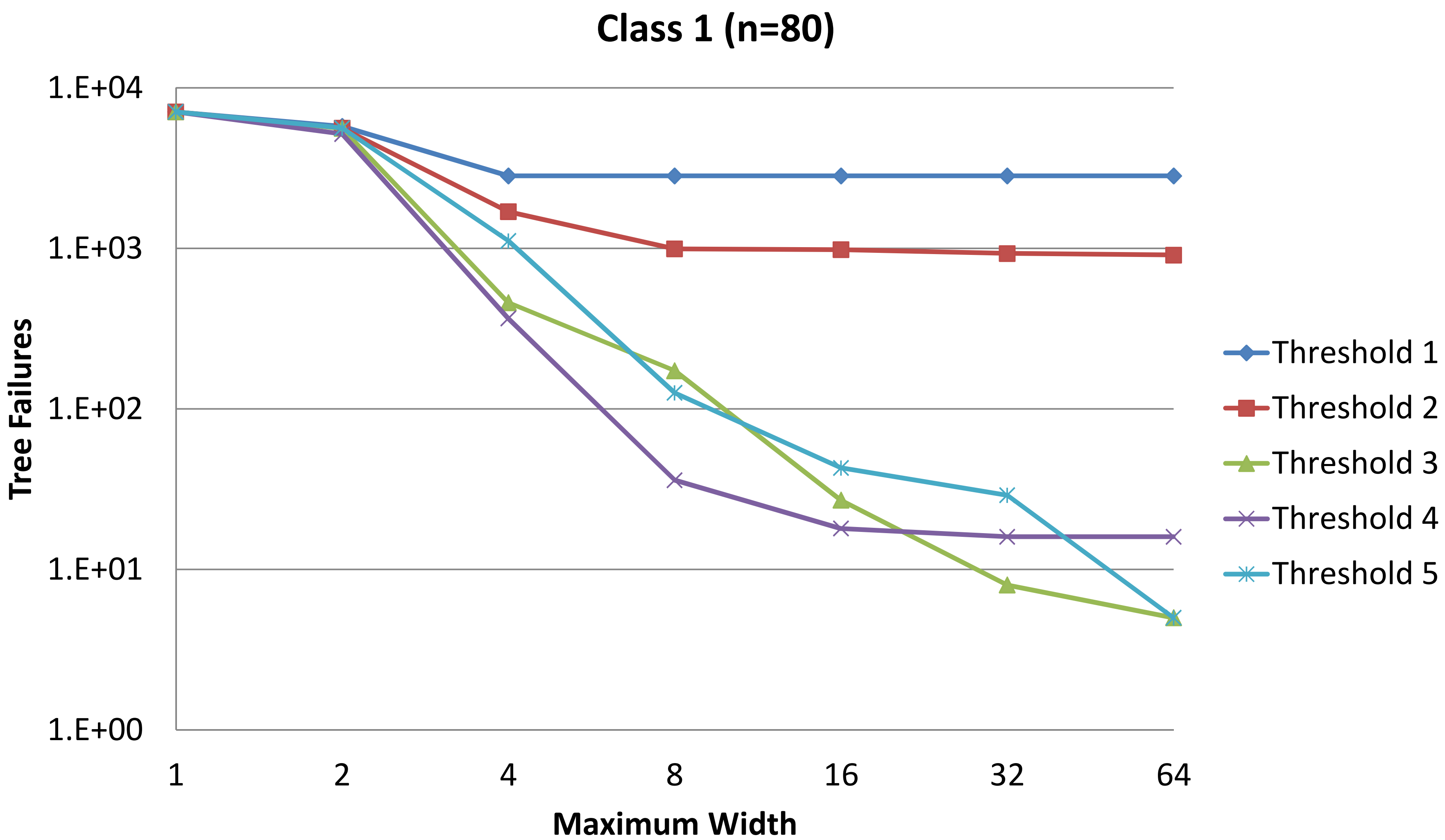
Search tree nodes



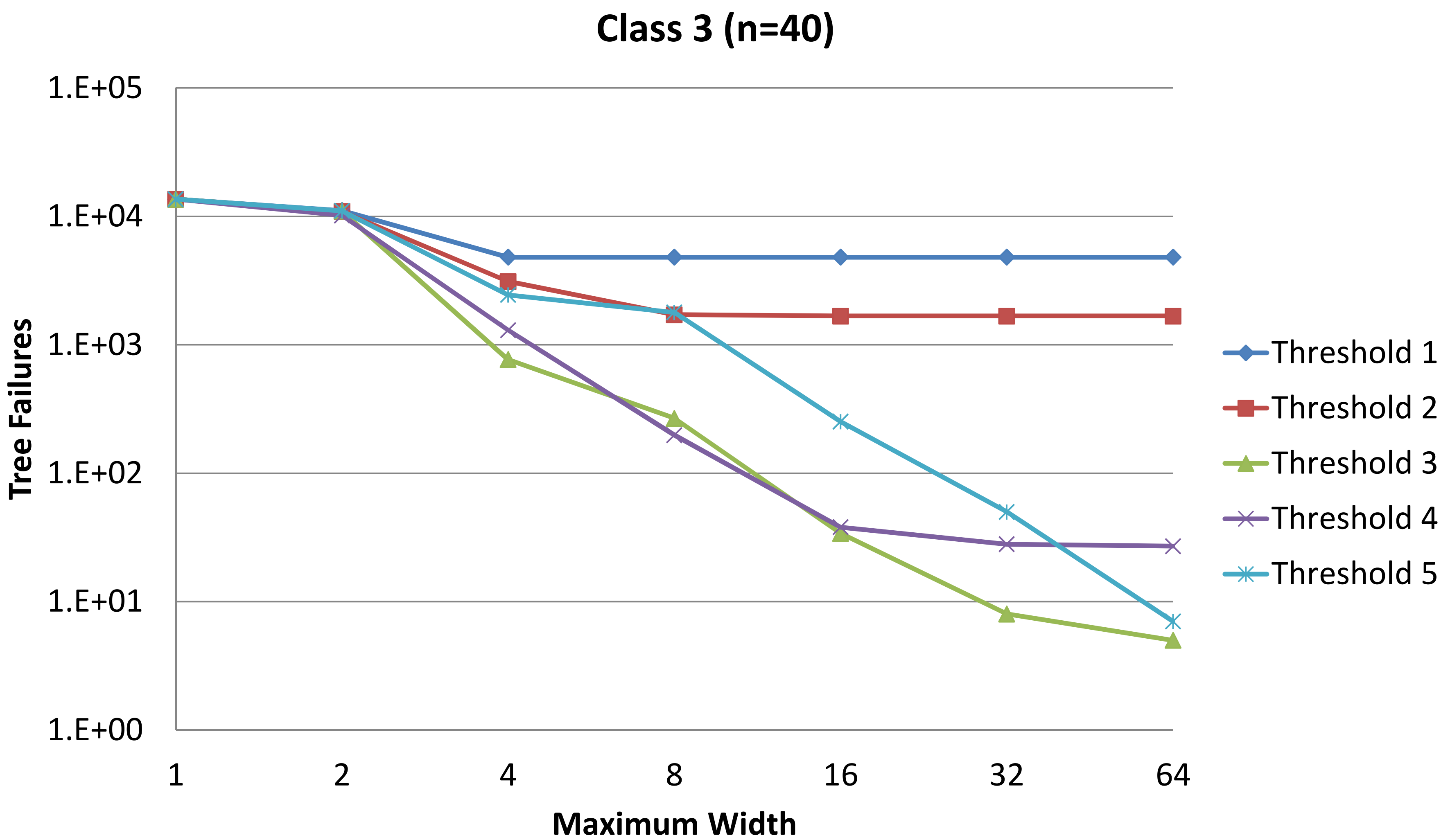
Search tree failures



Search tree failures



Search tree failures



Compared to the state of the art

	Size	gcc+seq BT	gen-seq BT	among BDD store BT	gcc+seq CPU	gen-seq CPU	among BDD store CPU
Class 1	40	185287	0	253	216.49	0.77	3.76
Class 1	80	198091	0	97	1061.62	0.61	7.50
Class 2	40	393748	0	204	390.93	0.01	1.38
Class 2	80	393748	0	51	1786.62	0.05	5.38
Class 3	40	328376	0	510	417.63	34.43	177.37
Class 3	80	1847335	0	295	7457.36	15.41	160.85

CONCLUSION AND RESEARCH ISSUES

Conclusion

- MDD store provides **substantial advantage** over domain store for filtering **multiple among constraints**
  - **Wider** MDDs yield greater speedups
  - **Huge reduction** in the amount of backtracking

- **Intensive processing** at search nodes can pay off when the constraint store is richer

Some research issues

- Adjusting the **width** and **threshold**
  - **Dynamic** adjustment
- Interaction with branching schemes
- How to propagate other constraints?
  - **Regular** constraint
  - **Sequence** constraint