A Scalable Method for Multiagent Constraint Optimization

Adrian Petcu, Boi Faltings
adrian.petcu@epfl.ch; boi.faltings@epfl.ch

Artificial Intelligence Laboratory
Ecole Polytechnique Fédérale de Lausanne (EPFL)
IN-Ecublens, 1015 Lausanne, Switzerland
http://liawww.epfl.ch/Publications/Archive/Petcu2005.pdf

Overview

- *Multiagent Combinatorial Optimization*
- Two solving paradigms: backtracking vs. dynamic programming
- DPOP – complete optimization based on dynamic programming (variable elimination)
- Evaluation
- Conclusions & future work

Example: Multiagent Meeting Scheduling

- A set of agents want to setup meetings
- M1: A1, A2
- M2: A2, A3
- M3: A1, A2, A3
- Time slots: [8AM-9AM,9AM-10AM...]
- Each agent has its preferences
- Goal: best schedule

M1,M2,M3

A1

M1,M3

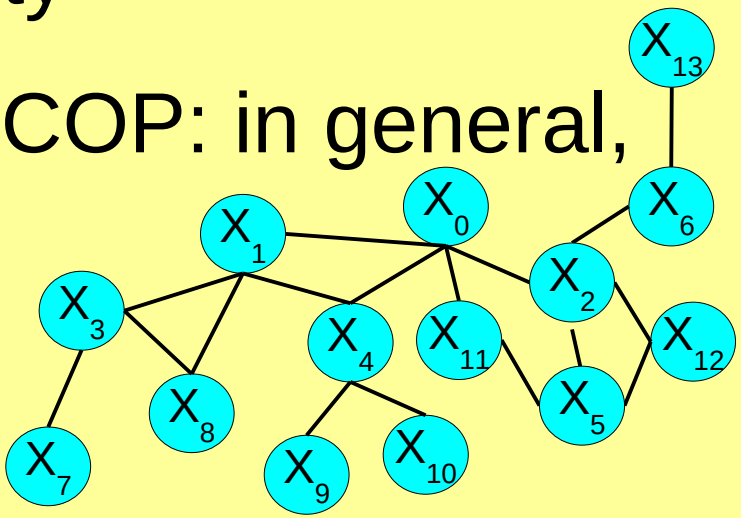
A3

M2,M3

Multi-agent Constraint Optimization (MCOP)

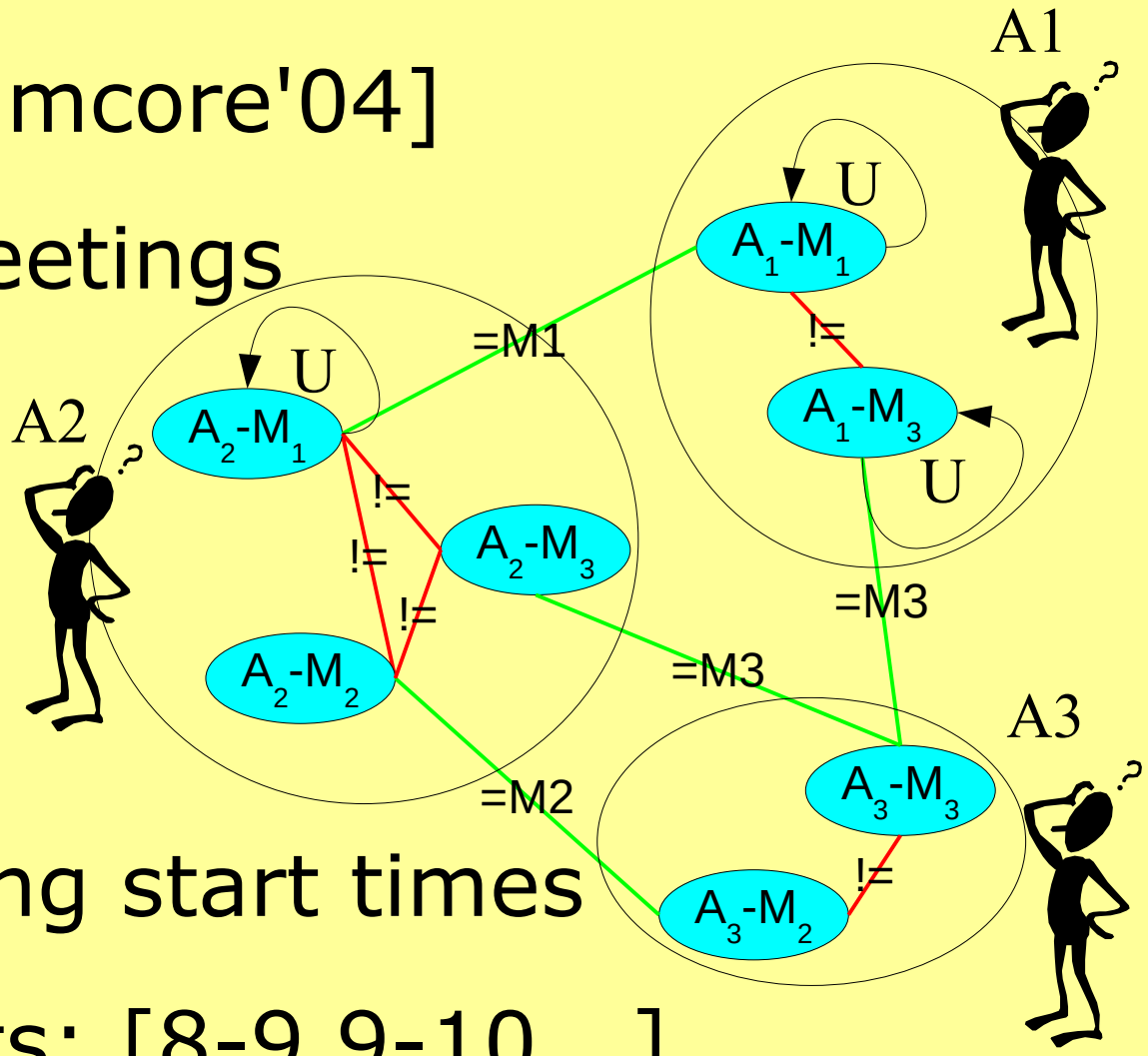
- Constraint Optimization Problems (COP): tuple <X=vars, D=domains, R=relations> (**valued CSPs**: [Schiex et al - IJCAI'95])
- Domains: finite, discrete
- Relations are real-valued utility functions
- Objective: maximize overall utility
- MCOP – multi-agent version of COP: in general, one variable per agent

R(X2,X1)	X_1		
	a	b	c
a	0	2	1
X2 b	3	4	6
c	5	2	5



Example: Multiagent Meeting Scheduling

- PEAV Model [Teamcore'04]
- Goal: setup 3 meetings
- M1: A1, A2
- M2: A2, A3
- M3: A1, A2, A3
- Variables=meeting start times
- Values=time slots: [8-9,9-10...]



Why not CSD™ (centralize, solve, distribute)?

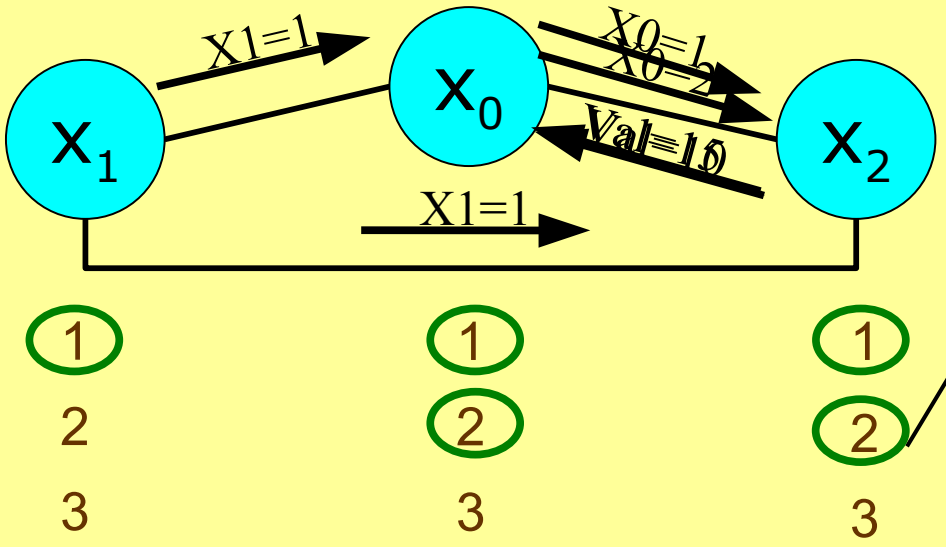
- Centralized:
  - pros: simpler, usually more efficient algorithms
  - cons: performance bottleneck, central point of failure, **privacy loss**, more sensitive to fraud, latency in dynamic systems
- Distributed:
  - pros: (some) resilience to failures, better workload distribution, better privacy
  - cons: complex algorithms, communication overhead, some techniques are not easily distributed

Overview

- Multiagent Combinatorial Optimization
- *Two solving paradigms: backtracking vs. dynamic programming*
- DPOP – a utility propagation algorithm based on dynamic programming
- Experimental evaluation
- Conclusions & future work

Solving paradigm 1: backtracking

- Assumes an ordering of the variables
- Variables are instantiated sequentially
- After (complete) assignment, backtrack
- Examples: ADOPT, ABT, AWC, B&B…



The utility of this combination

Exponential # of messages

(at least one message for a state change)

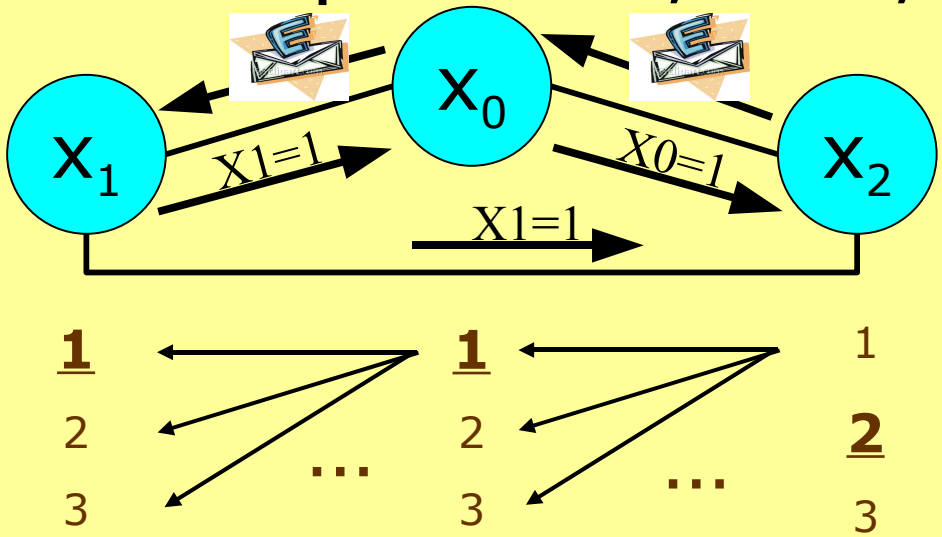
Linear sized messages

Q: What's wrong with many messages?

- X1 performs assignment X1=1
  - Local assignment operation (let's say it's free)
  - Identify recipient (DNS lookup)
  - Open connection with recipient
  - Send data packet (X1's value) – typical packet size: 100's to 1000's bytes (for 1 value: X1=1!!)
  - Acknowledgment
  - Close connection
- Several times over (once for each child)
- A: HUGE overhead!

Solving paradigm 2: dynamic programming

- Assumes an ordering of the variables
- Incrementally compute all partial solutions; when they are complete, pick the best one
- Examples: BE, CTE, BTE, DPOP



The whole set of utilities one for each combination (exponentially many)

Exponential message size

Linear # of messages

Overview

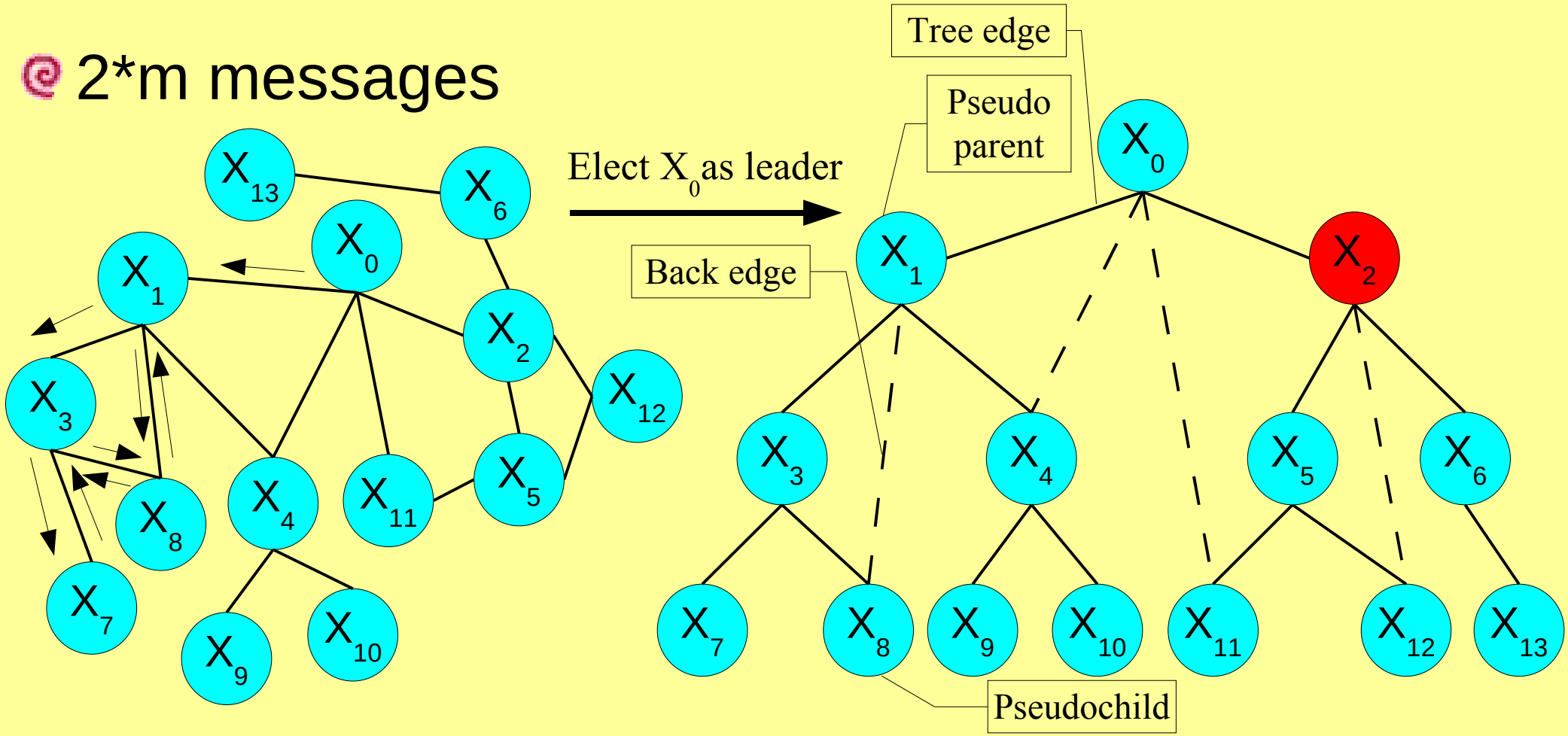
- Multiagent Combinatorial Optimization
- Two solving paradigms: backtracking vs. dynamic programming
- *DPOP – a utility propagation algorithm based on dynamic programming*
- Experimental evaluation
- Conclusions & future work

DPOP – main ideas

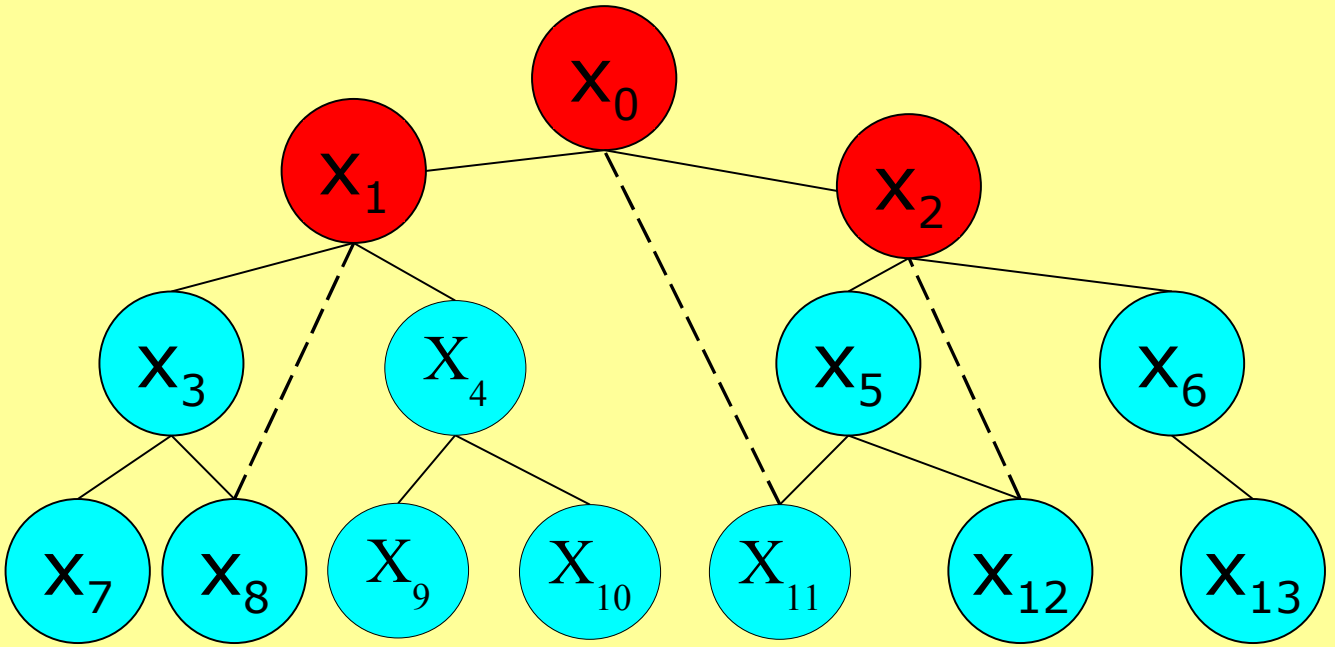
- Distributed utility propagation algorithm (aka distributed bucket elimination on DFS trees)
  - **1**: **DFS:** distributed depth first traversal
  - **2**: **UTIL*ity*** messages propagate bottom-up along the DFS tree
  - **3**: **Optimal VALUE** assignments propagate top-down
- DFS: 2*m messages transmitted
- UTIL*:* n-1 messages transmitted
- VALUE: n-1 messages transmitted

DPOP phase 1: distributed depth first traversal

- *DFS tree*: neighbors lie in the same branch
- *Prop*: subtrees are independent (e.g. no link $X_8 - X_4$)
- 2*m messages



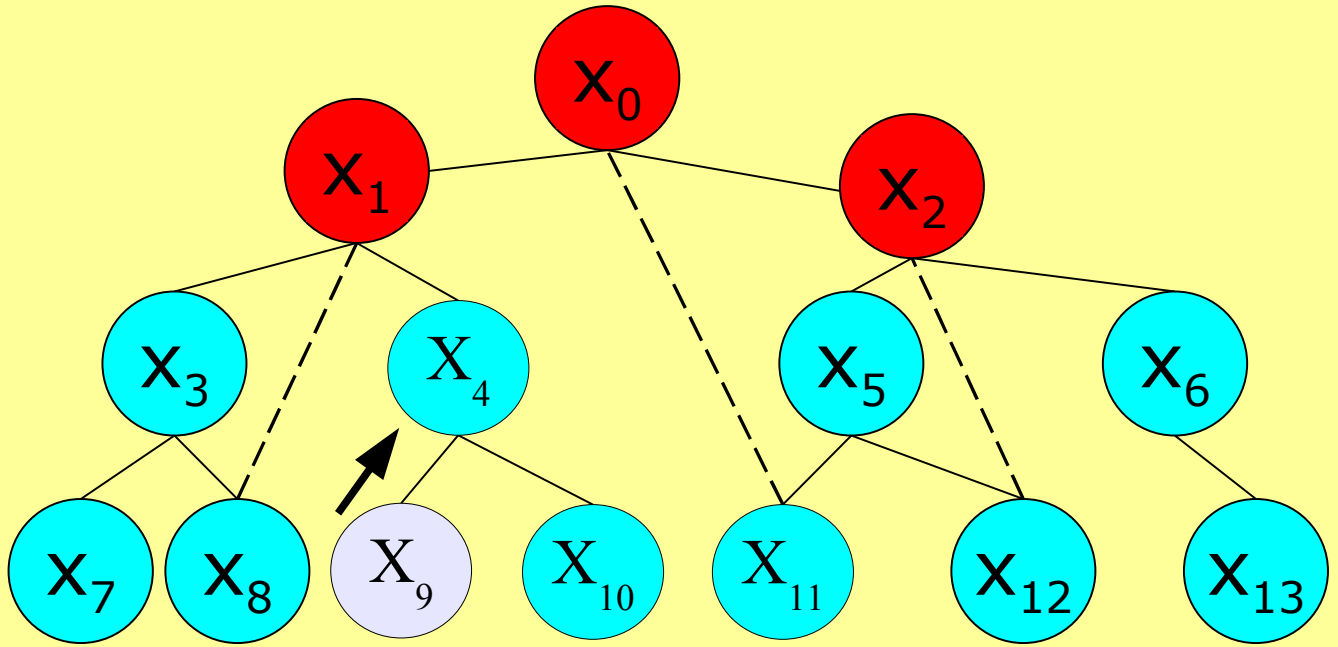
DPOP phase 2: bottom-up *UTIL* propagation



- Process equivalent to **bucket elimination** (Dechter96) on DFS
- propagation starts from leaves and goes up to the root
- each node waits for *UTIL* messages from children, joins them and sends *UTIL* to its parent
- n-1 messages transmitted
- Space exponential in the induced width of the problem

DPOP phase 2: bottom-up *UTIL* propagation

X9→X4	x_4		
	a	b	c
	5	4	6

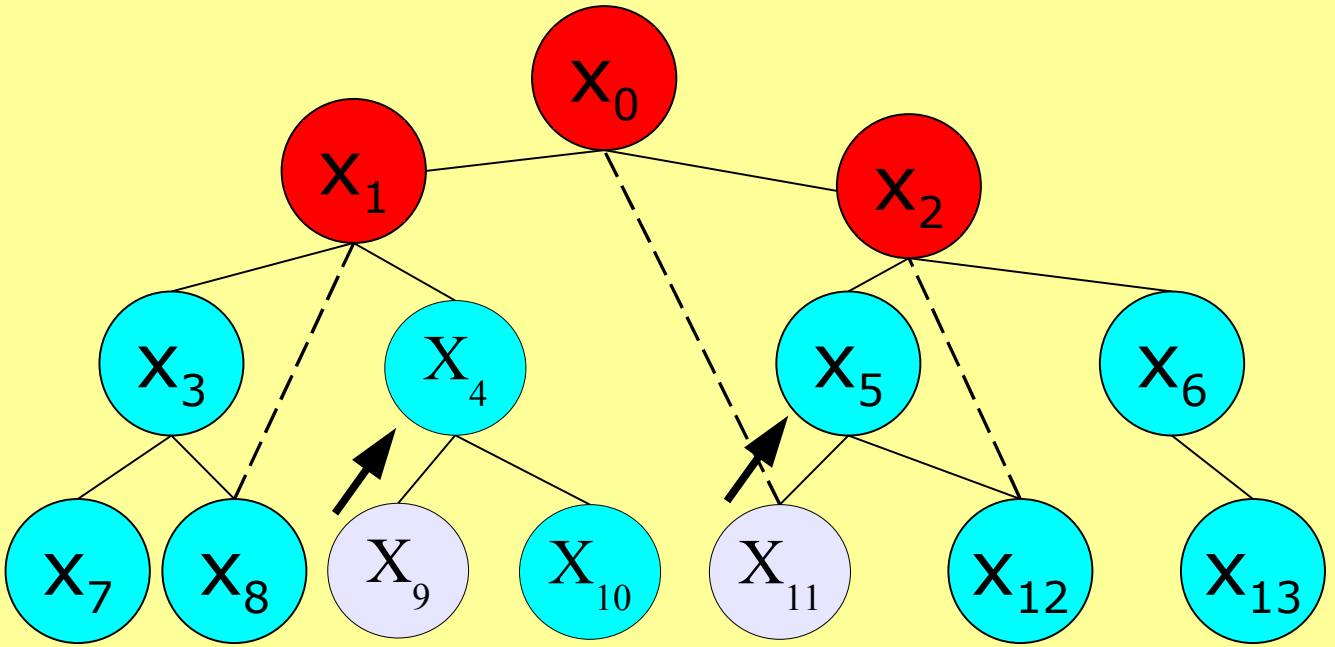


- Process equivalent to **bucket elimination** (Dechter96) on DFS
- propagation starts from leaves and goes up to the root
- each node waits for *UTIL* messages from children, joins them and sends *UTIL* to its parent
- n-1 messages transmitted
- Space exponential in the induced width of the problem

DPOP phase 2: bottom-up *UTIL* propagation

X9→X4	x_4		
	a	b	c
	5	4	6

X11→X5	x_5		
	a	b	c
a	2	2	3
X0 b	5	3	7
c	6	3	1

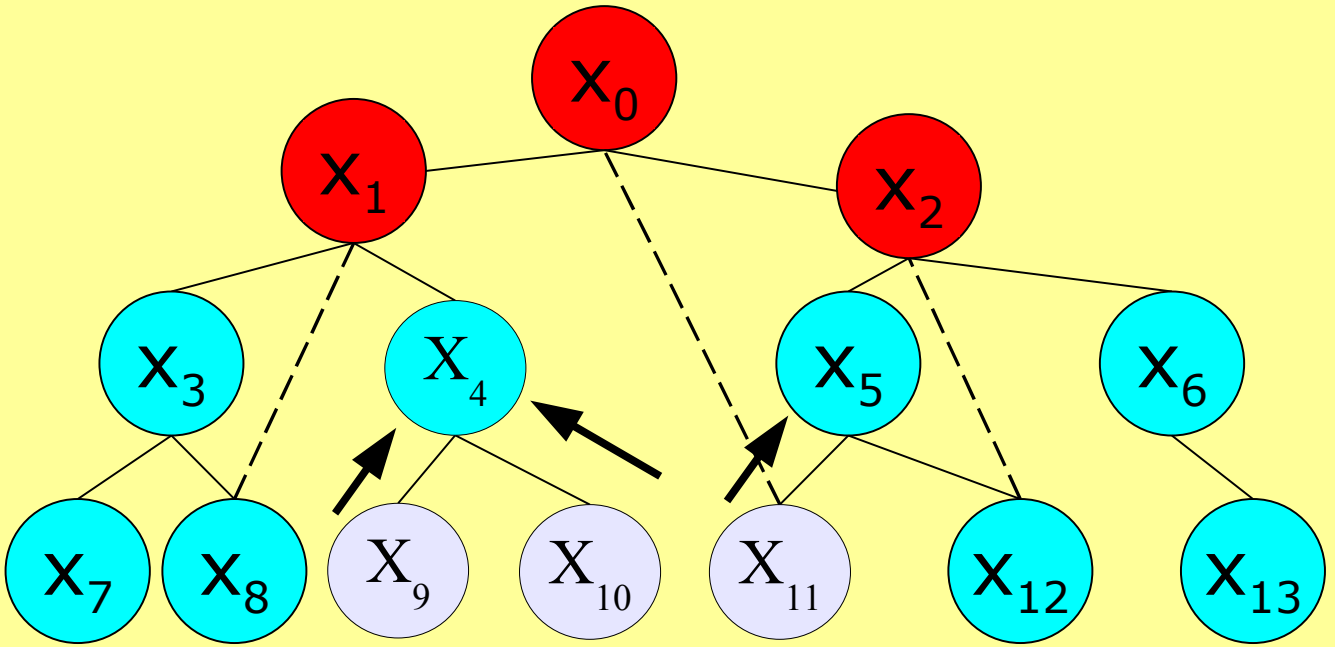


- Process equivalent to **bucket elimination** (Dechter96) on DFS
- propagation starts from leaves and goes up to the root
- each node waits for *UTIL* messages from children, joins them and sends *UTIL* to its parent
- n-1 messages transmitted
- Space exponential in the induced width of the problem

DPOP phase 2: bottom-up *UTIL* propagation

X9→X4	x_4		
	a	b	c
	5	4	6

X11→X5	x_5		
	a	b	c
a	2	2	3
X0 b	5	3	7
c	6	3	1

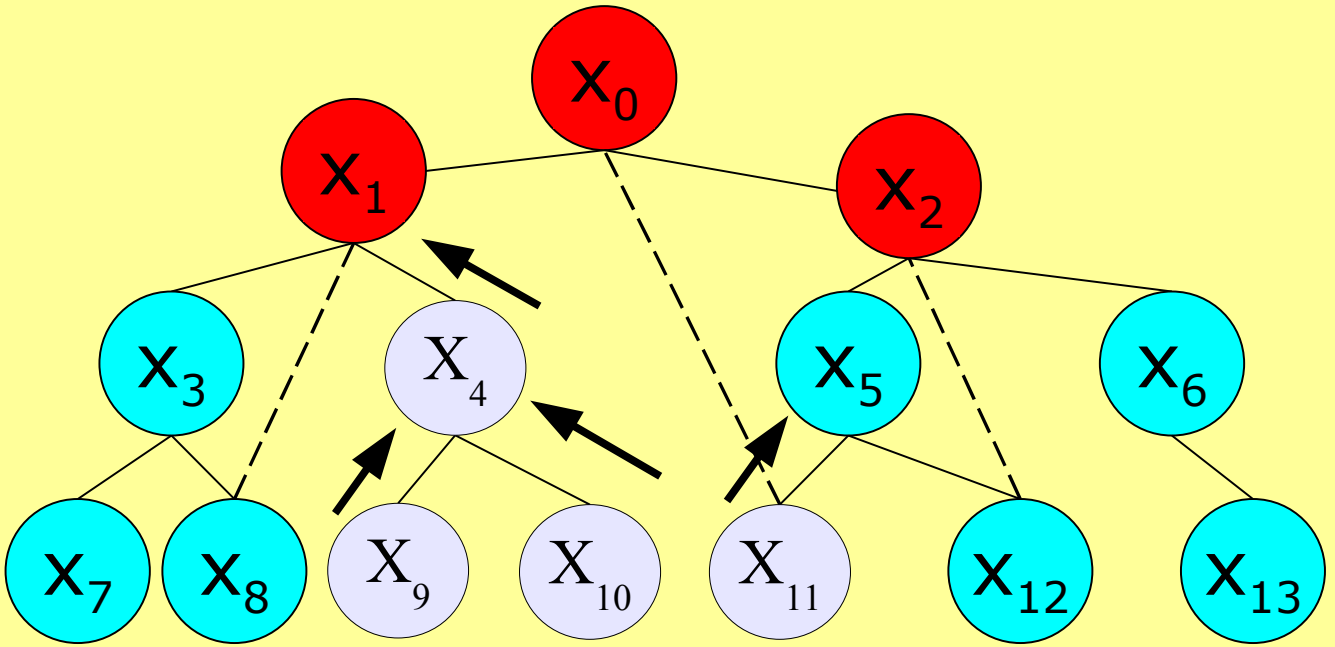


- Process equivalent to **bucket elimination** (Dechter96) on DFS
- propagation starts from leaves and goes up to the root
- each node waits for *UTIL* messages from children, joins them and sends *UTIL* to its parent
- n-1 messages transmitted
- Space exponential in the induced width of the problem

DPOP phase 2: bottom-up *UTIL* propagation

X9→X4	x_4		
	a	b	c
	5	4	6

X11→X5	x_5		
	a	b	c
a	2	2	3
X0 b	5	3	7
c	6	3	1



- Process equivalent to **bucket elimination** (Dechter96) on DFS
- propagation starts from leaves and goes up to the root
- each node waits for *UTIL* messages from children, joins them and sends *UTIL* to its parent
- n-1 messages transmitted
- Space exponential in the induced width of the problem

DPOP – computation details: $X_2 \rightarrow X_1$, $X_3 \rightarrow X_1$ and $X_1 \rightarrow X_0$

X_2

R(X2,X1)

X2 \ X_1	a	b	c
a	0	2	1
b	3	4	6
c	5	2	5

X_3

R(X3,X1)

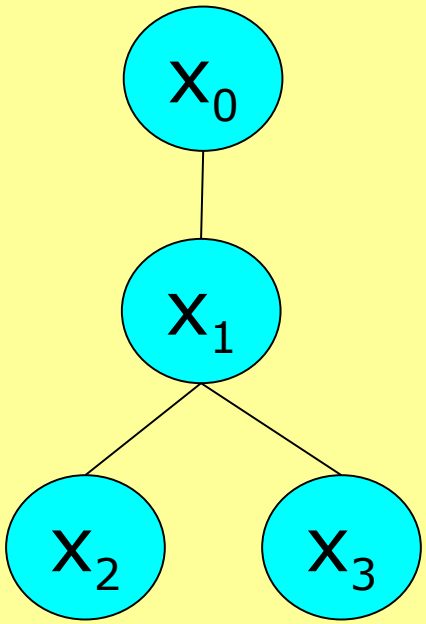
X3 \ X_1	a	b	c
a	6	2	3
b	3	3	2
c	4	4	1

X_1

R(X1,X0)

X1 \ X_0	a	b	c
a	2	2	3
b	5	3	7
c	6	3	1

X_0



DPOP – computation details: $X_2 \rightarrow X_1$, $X_3 \rightarrow X_1$ and $X_1 \rightarrow X_0$

X_2

R(X2,X1)

X2 \ X_1	a	b	c
a	0	2	1
b	3	4	6
c	5	2	5

X_3

R(X3,X1)

X3 \ X_1	a	b	c
a	6	2	3
b	3	3	2
c	4	4	1

X_1

X2→X1

X_1	a	b	c
	5	4	6

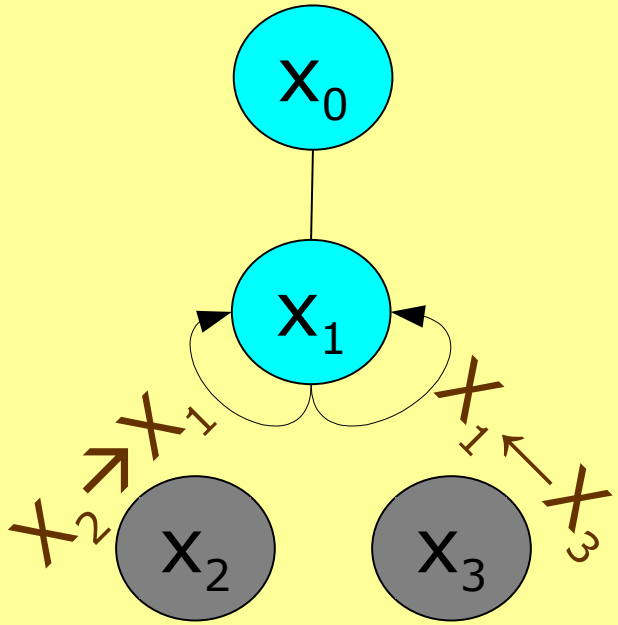
X3→X1

X_1	a	b	c
	6	4	3

R(X1,X0)

X1 \ X_0	a	b	c
a	2	2	3
b	5	3	7
c	6	3	1

X_0



DPOP – computation details: $X_2 \rightarrow X_1$, $X_3 \rightarrow X_1$ and $X_1 \rightarrow X_0$

X_2

R(X2,X1)

X2 \ X_1	a	b	c
a	0	2	1
b	3	4	6
c	5	2	5

X_3

R(X3,X1)

X3 \ X_1	a	b	c
a	6	2	3
b	3	3	2
c	4	4	1

X_1

X2→X1

X_1	a	b	c
	5	4	6

+

X3→X1

X_1	a	b	c
	6	4	3

+

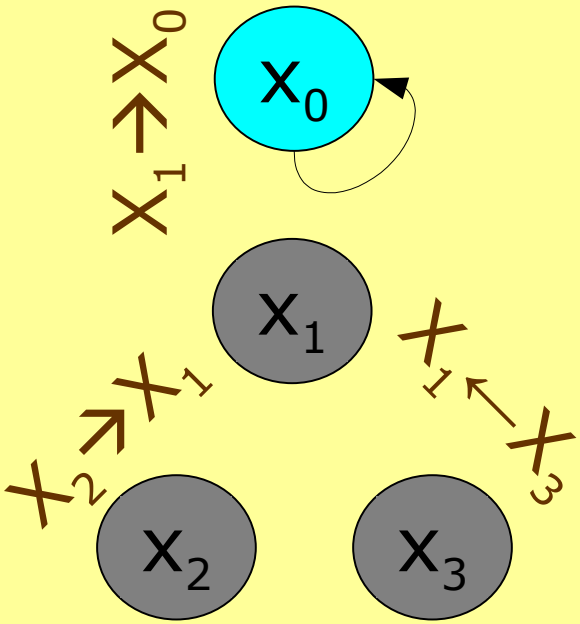
R(X1,X0)

X1 \ X_0	a	b	c
a	2	2	3
b	5	3	7
c	6	3	1

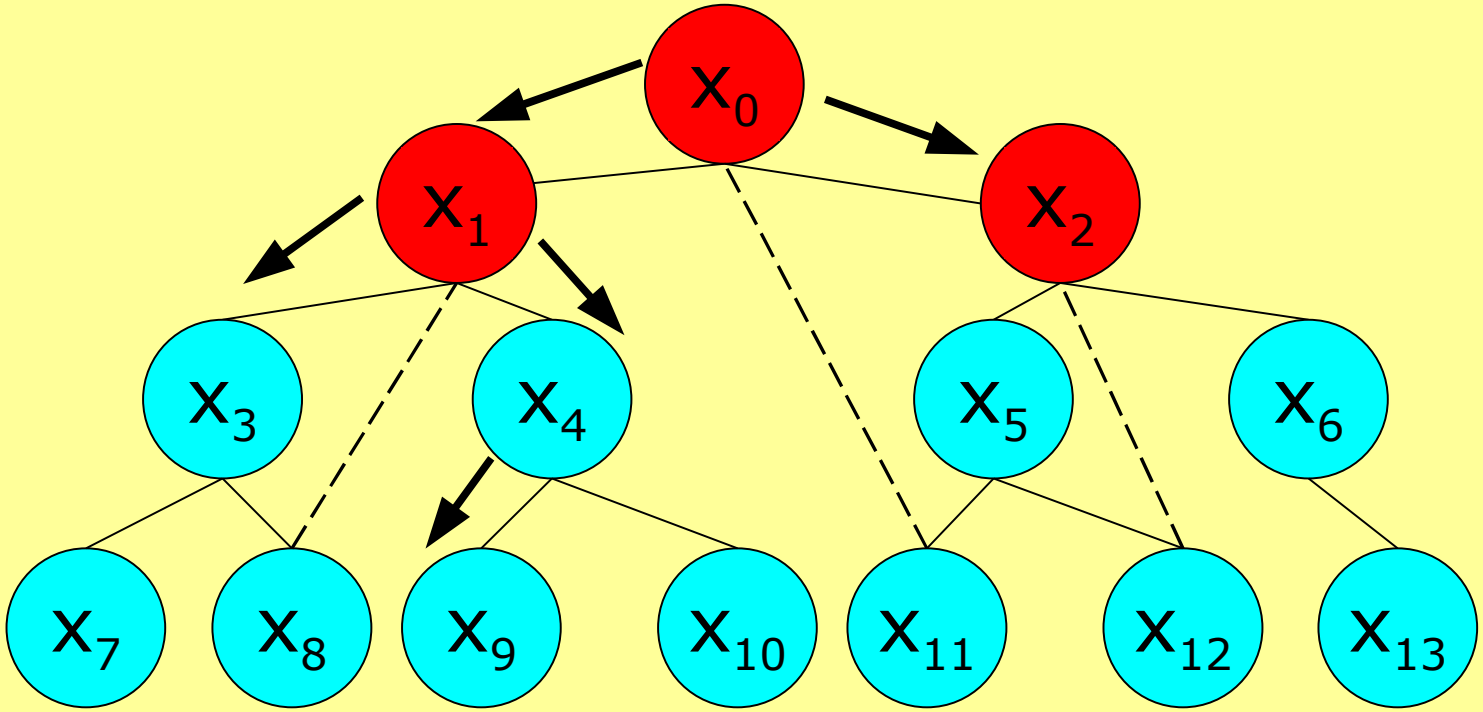
X_0

X1→X0

X_0	a	b	c
	15	13	15



DPOP phase 3: top-down *VALUE* propagation



- Process equivalent to **solution reconstruction** in BE (Dechter96)
- The root has global *UTIL* information; it finds its best value, and sends *VALUE* message to x_1 and x_2.
- Intermediary nodes (x_1 and x_2) subsequently do the same
- The process continues top-down to the leaves.

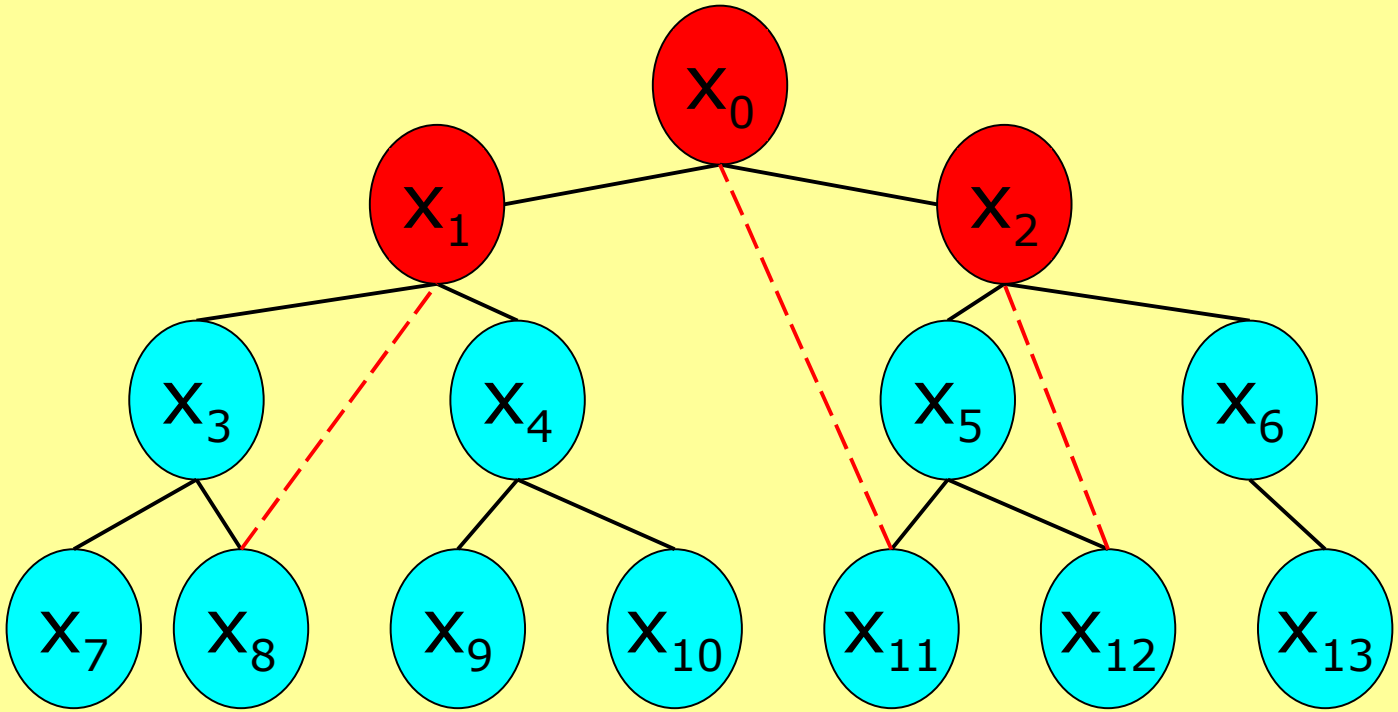
DPOP: distributed optimization method

- Quick recap: complete method, 3 phases:
  - 1: **DFS** generation protocol → correct DFS
  - 2: **UTIL*ity*** messages bottom-up
  - 3: **Optimal VALUE** assignments top-down
- All 3 phases produce a linear number of messages
- At the end of phase 3, all nodes are assigned their optimal values

DPOP – complexity is exponential in induced width

X9→X4	X_4		
	a	b	c
	5	4	6

X11→X5	X_5		
	a	b	c
a	2	2	3
X0 b	5	3	7
c	6	3	1



- **Linear # of messages!** largest one is exponential in induced width
- Each back edge has its associated tree path
- Overlaps between tree paths increase complexity
- **Space complexity** = maximum no of overlaps = **induced width**

Overview

- Multiagent Combinatorial Optimization
- Two solving paradigms: backtracking vs. dynamic programming
- DPOP – a utility propagation algorithm based on dynamic programming
- *Experimental evaluation*
- Conclusions & future work

Experimental results on meeting scheduling

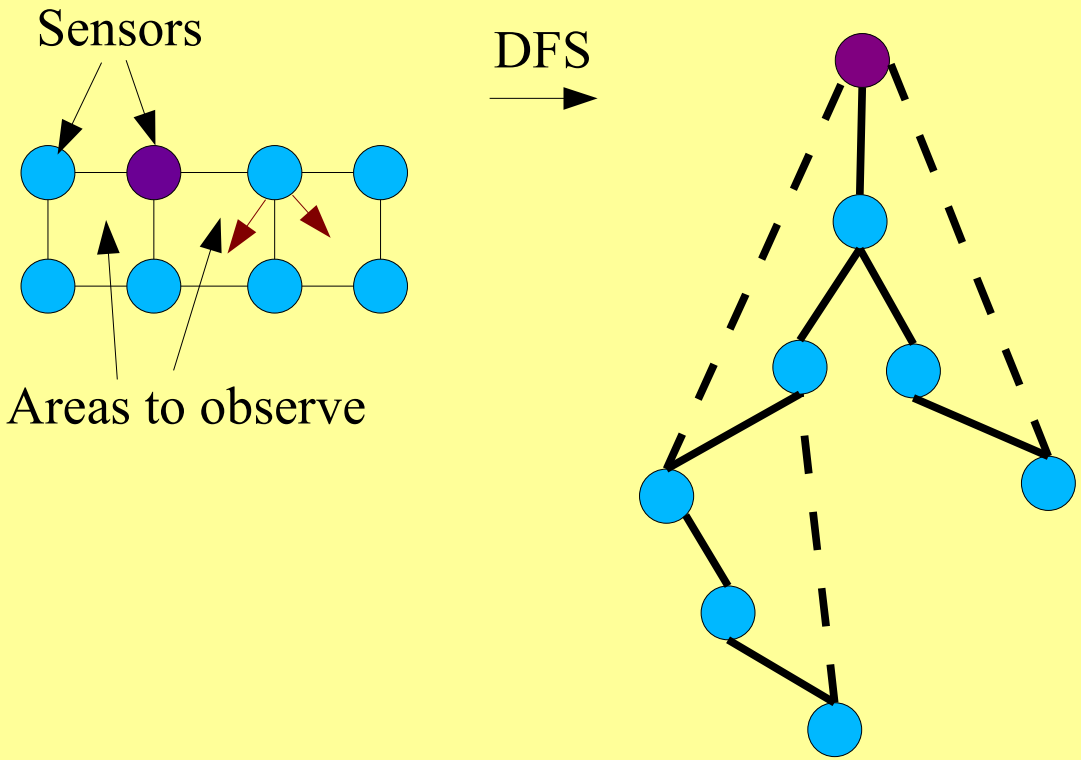
- Model of *Maheswaran et al. 2004*: find the optimal schedule for *m* meetings between *n* agents, each having his own preferences
- Largest previous experiments: 33 agents, 12 meetings, 47 variables, 123 constraints, solved with ADOPT (backtracking)
- Scales well for loose problems!!

Adopt →

Agents	Meetings	Vars	Constr	Msgs	Max size	Time(s)	Cycles
30	14	44	52	95	512	2.1	30
40	15	50	60	109	4096	7.2	32
70	34	112	156	267	32K	21.6	70
100	50	160	214	373	256K	43.4	86
200	101	270	341	610	256K	72.3	96

Sensor networks (Maheswaran et al '04)

- Exclusively allocate sensors to areas -NP
- Width=2 -> DPOP scales indefinitely
- DFS depth increases ->ADOPT does not scale



Overview

- Multiagent Combinatorial Optimization
- Two solving paradigms: backtracking vs. dynamic programming
- DPOP – a utility propagation algorithm based on dynamic programming
- Experimental evaluation
- *Conclusions & future work*

The beauty of this beast (aka advantages)

- Reduction in complexity: dom^{n} (BKT)>> dom^{height} (ADOPT, etc) >= dom^{width} (DPOP) ($height>=width$)
- Takes advantage of problem structure
- Works well for large but loose problems
- Linear number of messages, thus low overhead
- Easy to spot difficult parts of the problem; can *priori* compute the required network traffic
- Easy to generate the elimination order (DFS)

The beast within this beauty (aka disadvantages)

- Message size and memory are exponential in the induced width (things can get BIG…); however, likely to work well for loose problems
- Choice of the DFS is relevant, as the induced width varies with it;
- Finding the min-width DFS is NP-hard

Conclusions, future work
aka living together, happily ever after

- Distributed optimization with a linear # of messages
- Works well for large but loose problems
- Various extensions possible:
  - Bounded propagation: accuracy vs. size, anytime
  - Local search/inference hybrid for dense problems
  - Self stabilizing version for dynamic optimization
  - Optimal solution stability in dynamic optimization
  - Distributed computation of VCG taxes for IC
- How relevant is the the DFS? Better heuristics?

References

- F. Kschischang, B.Frey, and H. Löliger: *Factor graphs and the sum-product algorithm*. IEEE Transactions on Information Theory, 2001.
- Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
- Rina Dechter: *Constraint Networks*, Encyclopedia of Artificial Intelligence, Second Edition, pages 276-285, 1992.
- Modi, Shen, Tambe and Yokoo. ADOPT: *An asynchronous complete method for distributed constraint optimization*.
- Yokoo et al. ABT: Asynchronous backtracking
- Yokoo et al. AWC: Asynchronous weak commitment search
- Maheswaran, Tambe, Bowring, Pearce, Varakantham: *Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling*

DPOP phase 1: distributed depth first traversal

- *Pseudotree*: neighbors lie in the same branch
- *Prop*: subtrees are independent (e.g. no link $X_8 - X_4$)
- DFS is pseudo tree!