Autonomous Agents and Multiagent Systems

Distributed Constraint Optimization

Francesco Amigoni

Many of the following slides are taken from:

- the “lecture slides provided by authors”, Chapter 12, available at:
  http://www.the-mas-book.info/index.html
- the “Tutorial on Multi-agent Distributed Constrained Optimization” at AAMAS2019, available at:
  https://www2.isye.gatech.edu/~fferdinando3/cfp/AAMAS19/

Chapter 12: Distributed Constraint Handling and Optimization

A. Farinelli[1] M. Vinyals[1] A. Rogers[2] N.R. Jennings[2]

[1]Computer Science Department
University of Verona, Italy

[2]Agents, Interaction and Complexity Group
School of Electronics and Computer Science
University of Southampton,UK

Multi-Agent Systems. MIT Press, 2012 (2nd edition), edited by Gerhard Weiss. http://www.the-mas-book.info

Constraints

- Pervade our everyday lives
- Are usually perceived as elements that limit solutions to the problems we face

Constraints

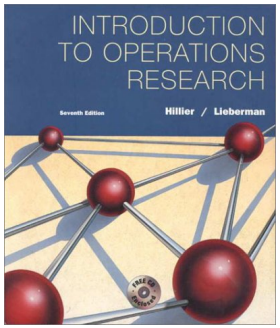
From a computational point of view, they:

- Reduce the space of possible solutions
- Encode knowledge about the problem at hand
- Are key components for efficiently solving hard problems

Constraint Processing

Many different disciplines deal with hard computational problems that can be made tractable by carefully considering the constraints that define the structure of the problem.

Planning
Scheduling



Operational
Research

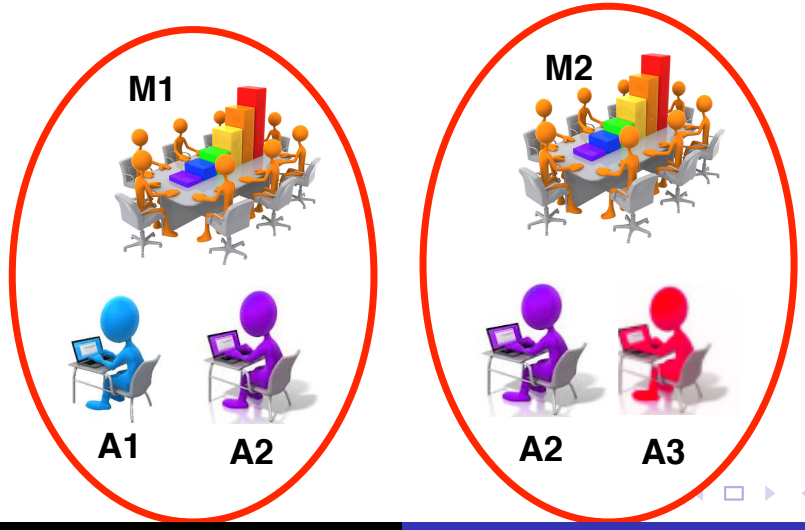
Automated Reasoning
Decision Theory

Computer
Vision

Constraint Processing in Multi-Agent Systems

Focus on how constraint processing can be used to address optimization problems in Multi-Agent Systems (MAS) where:

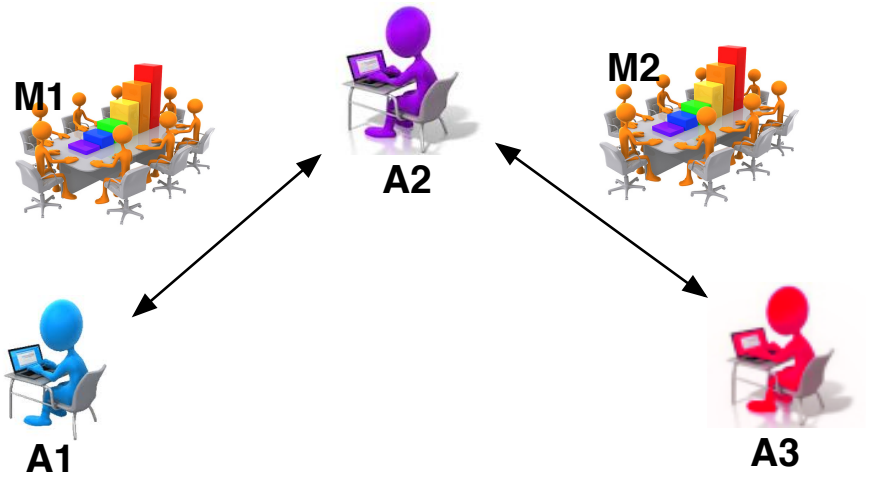
A set of agents must come to some agreement, typically via some form of negotiation, about which action each agent should take in order to jointly obtain the best solution for the whole system.



Distributed Constraint Optimization Problems (DCOPs)

We will consider Distributed Constraint Optimization Problems (DCOP) where:

Each agent negotiates locally with just a subset of other agents (usually called neighbors) that are those that can directly influence his/her behavior.



Constraint Networks

A constraint network $\mathcal{N}$ is formally defined as a tuple $\langle X, D, C \rangle$ where:

- $X = \{x_1, \ldots, x_n\}$ is a set of discrete variables;
- $D = \{D_1, \ldots, D_n\}$ is a set of variable domains, which enumerate all possible values of the corresponding variables; and
- $C = \{C_1, \ldots, C_m\}$ is a set of constraints; where a constraint C_i is defined on a subset of variables $S_i \subseteq X$ which comprise the scope of the constraint
  - $r = |S_i|$ is the arity of the constraint
  - Two types: hard or soft

Hard constraints

- A hard constraint C_i^h is a relation R_i that enumerates all the valid joint assignments of all variables in the scope of the constraint.

$$R_i \subseteq D_{i_1} \times \ldots \times D_{i_r}$$

$D_j = D_k = \{0, 1\}$

R_i	x_j	x_k
	0	1
	1	0

Soft constraints

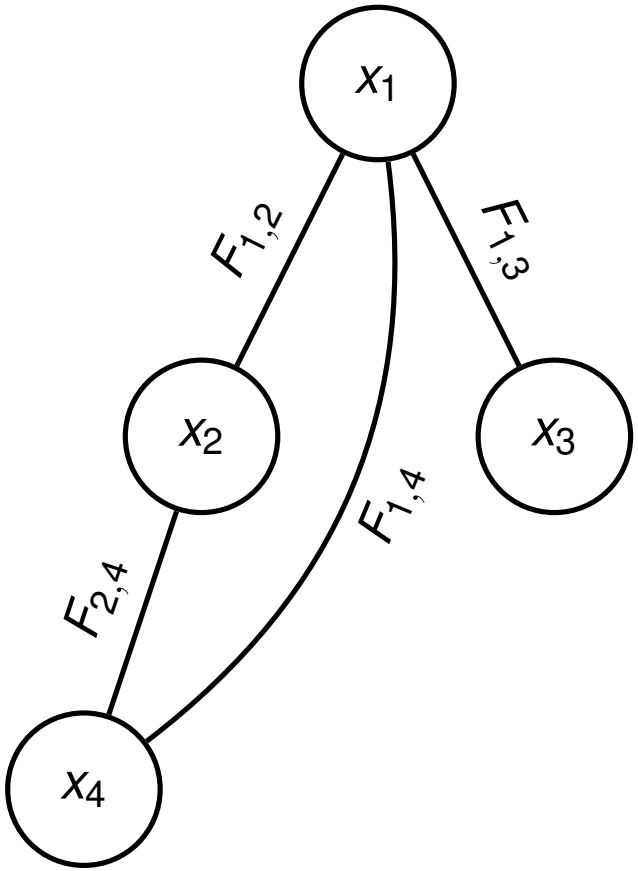
- A soft constraint C_i^s is a function F_i that maps every possible joint assignment of all variables in the scope to a real value.

$$F_i : D_{i_1} \times \ldots \times D_{i_r} \rightarrow \Re$$

F_i	x_j	x_k
2	0	0
0	0	1
0	1	0
1	1	1

Binary Constraint Networks

- Binary constraint networks are those where:
  - Each constraint (soft or hard) is defined over two variables.
- Every constraint network can be mapped to a binary constraint network
  - requires the addition of variables and constraints
  - may add complexity to the model
- They can be represented by a constraint graph



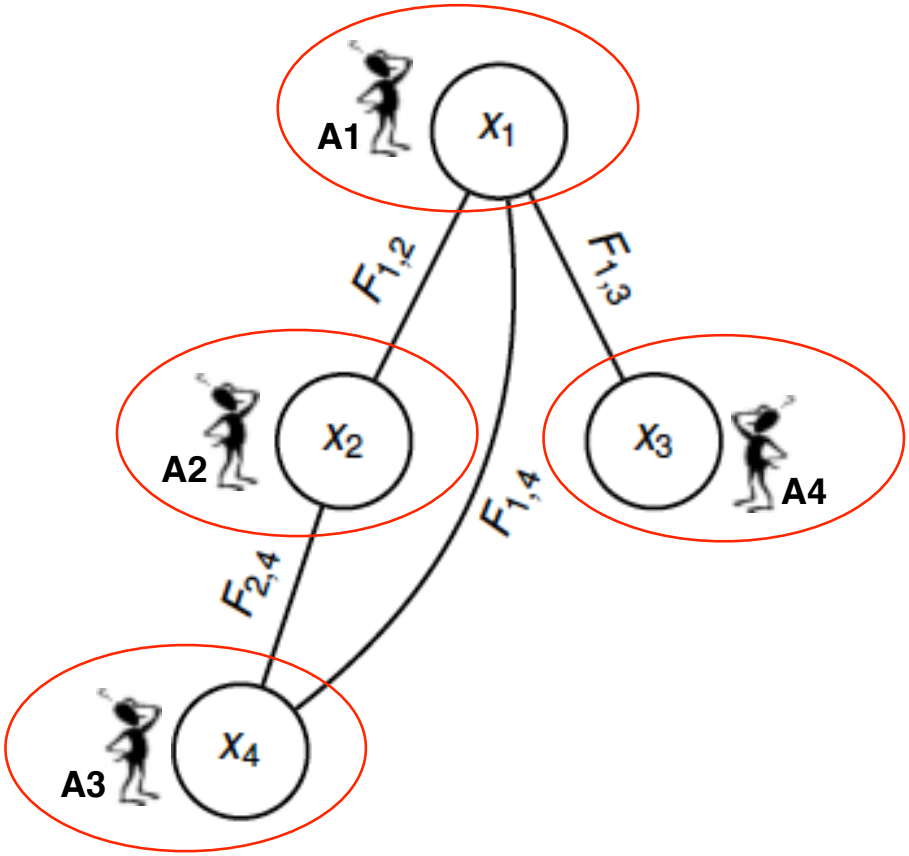
Different objectives, different problems

- **Constraint Satisfaction Problem (CSP)**
  - Objective: find an assignment for all the variables in the network that satisfies all constraints.
- **Constraint Optimization Problem (COP)**
  - Objective: find an assignment for all the variables in the network that satisfies all constraints and optimizes a global function.
  - Global function = aggregation (typically sum) of local functions. $F(x) = \sum_i F_i(x_i)$

Distributed Constraint Reasoning

When operating in a decentralized context:

- a set of agents control variables
- agents interact to find a solution to the constraint network



Distributed Constraint Reasoning

Two types of decentralized problems:

- **distributed CSP (DCSP)**
- **distributed COP (DCOP)**

Here, we focus on DCOPs.

Distributed Constraint Optimization Problem (DCOP)

A DCOP consists of a constraint network $\mathcal{N} = \langle X, D, C \rangle$ and a set of agents $A = \{A_1, \ldots, A_k\}$ where each agent:

- controls a subset of the variables $X_i \subseteq X$
- is only aware of constraints that involve variable it controls
- communicates only with its neighbours

Distributed Constraint Optimization Problem (DCOP)

- Agents are assumed to be fully cooperative
  - Goal: find the assignment that optimizes the global function, not their local local utilities.
- Solving a COP is NP-Hard and DCOP is as hard as COP.

Motivation

Why distribute?

- Privacy
- Robustness
- Scalability

Real World Applications

Many standard benchmark problems in computer science can be modeled using the DCOP framework:

- graph coloring

As can many real world applications:

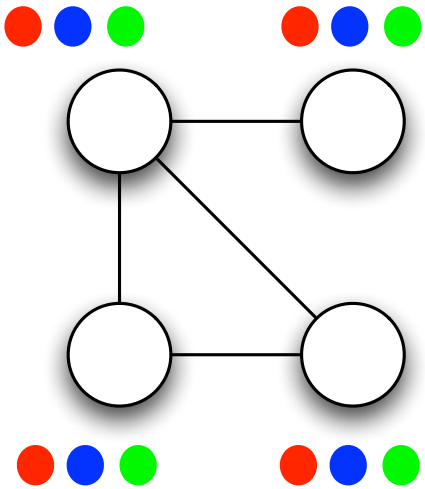
- human-agent organizations (e.g. meeting scheduling)
- sensor networks and robotics (e.g. target tracking)

Graph coloring

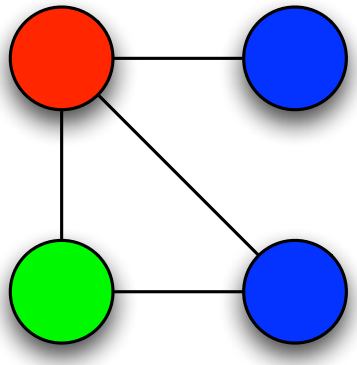
- Popular benchmark
- Simple formulation
- Complexity controlled with few parameters:
  - Number of available colors
  - Number of nodes
  - Density ($\#nodes/\#constraints$)
- Many versions of the problem:
  - CSP, MaxCSP, COP

Graph coloring - CSP

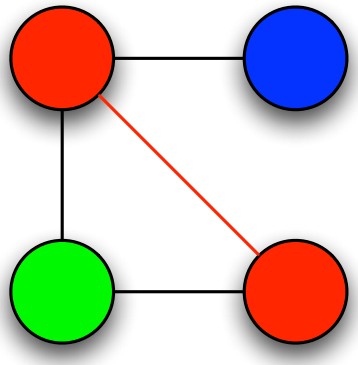
- Nodes can take k colors
- Any two adjacent nodes should have different colors
  - If it happens this is a conflict



Yes!

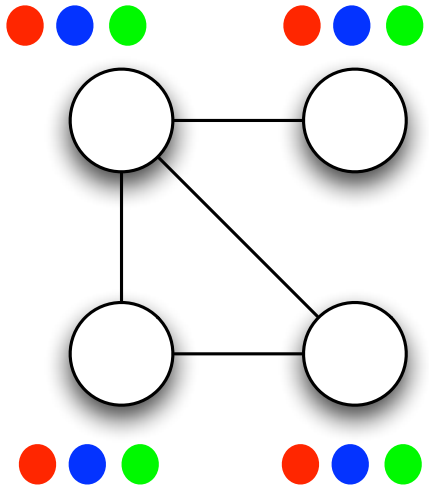


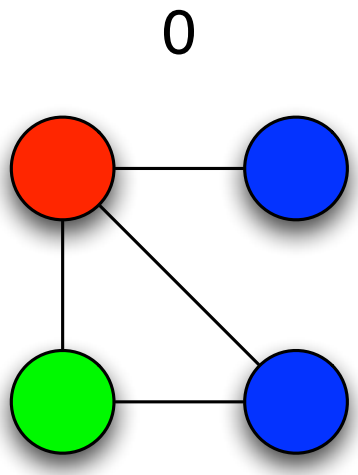
No!

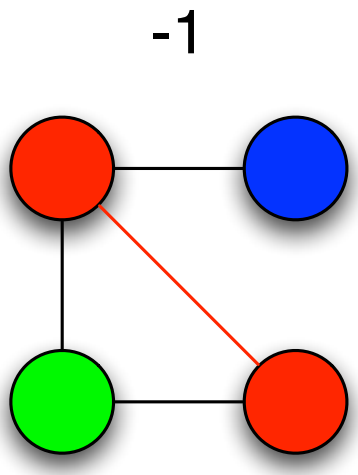


Graph coloring - Max-CSP

- Minimize the number of conflicts

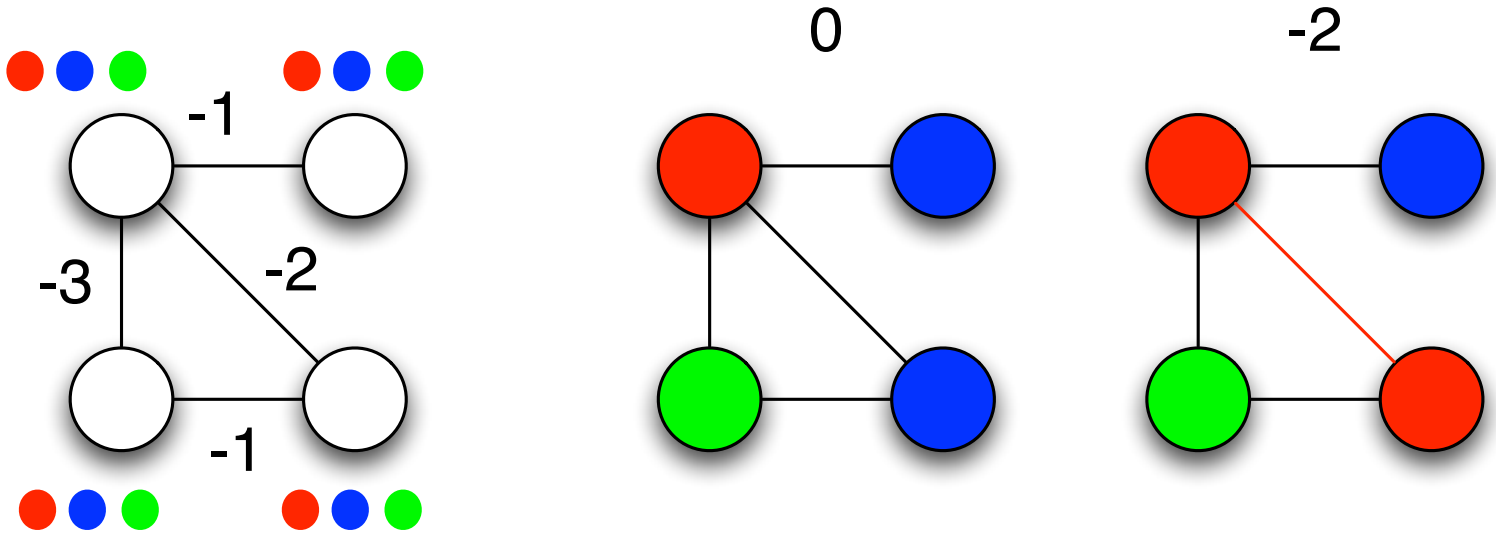






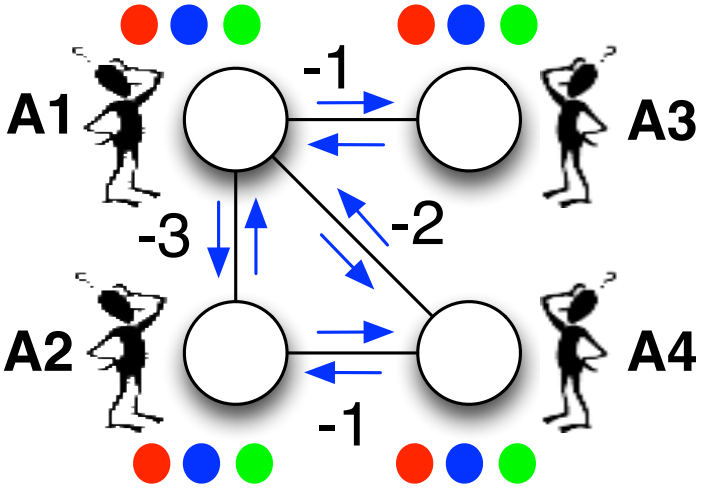
Graph coloring - COP

- Different weights to violated constraints
- Preferences for different colors



Graph coloring - DCOP

- Each node:
  - controlled by one agent
- Each agent:
  - Preferences for different colors
  - Communicates with its direct neighbours in the graph



- A1 and A2 exchange preferences and conflicts
- A3 and A4 do not communicate

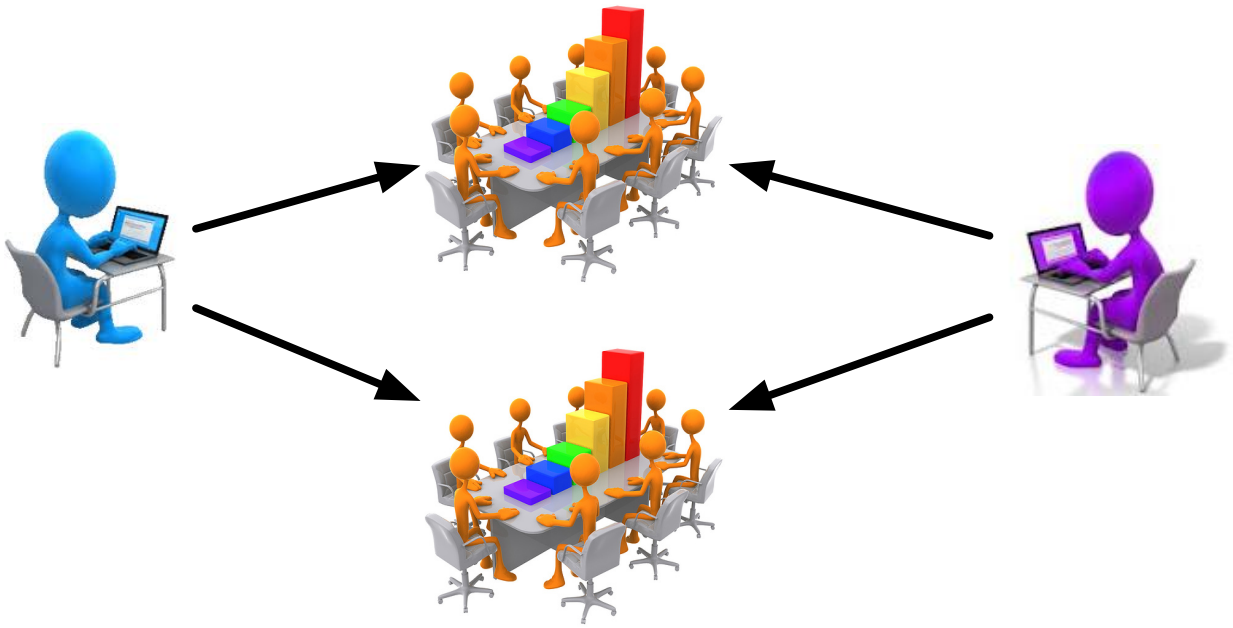
Meeting Scheduling

Motivation:

- Privacy
- Robustness
- Scalability

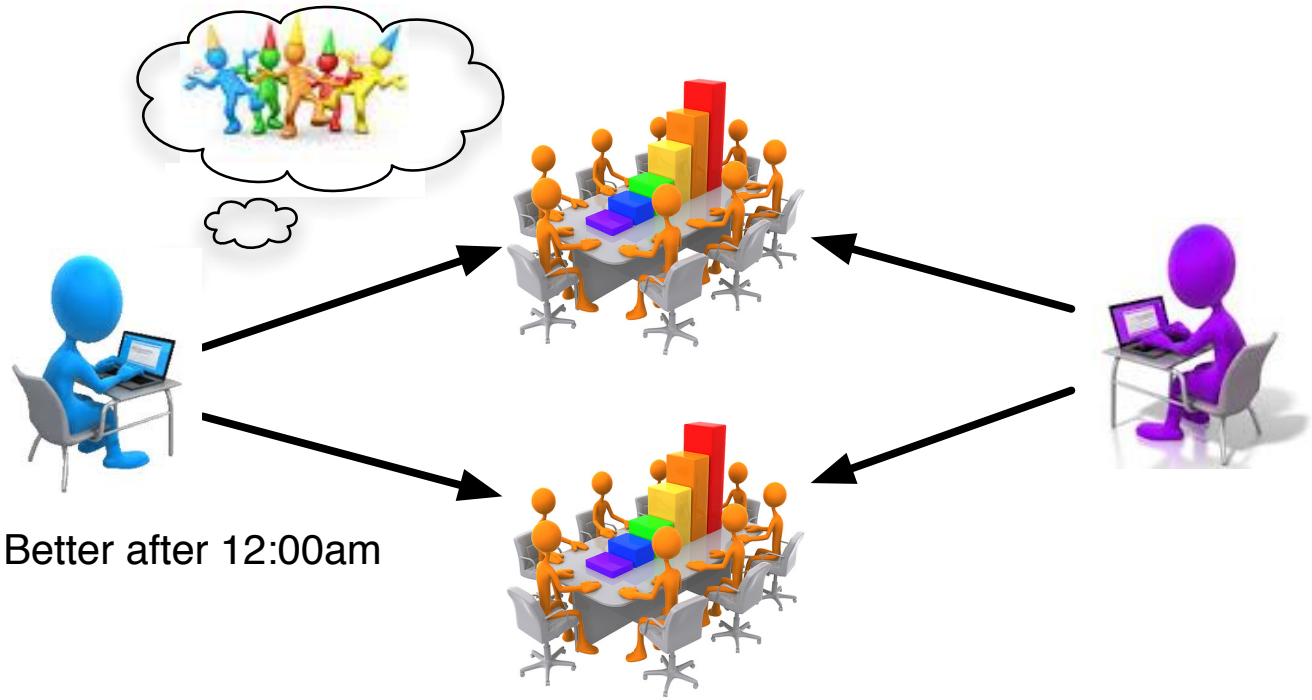
Meeting Scheduling

In large organizations many people, possibly working in different departments, are involved in a number of work meetings.



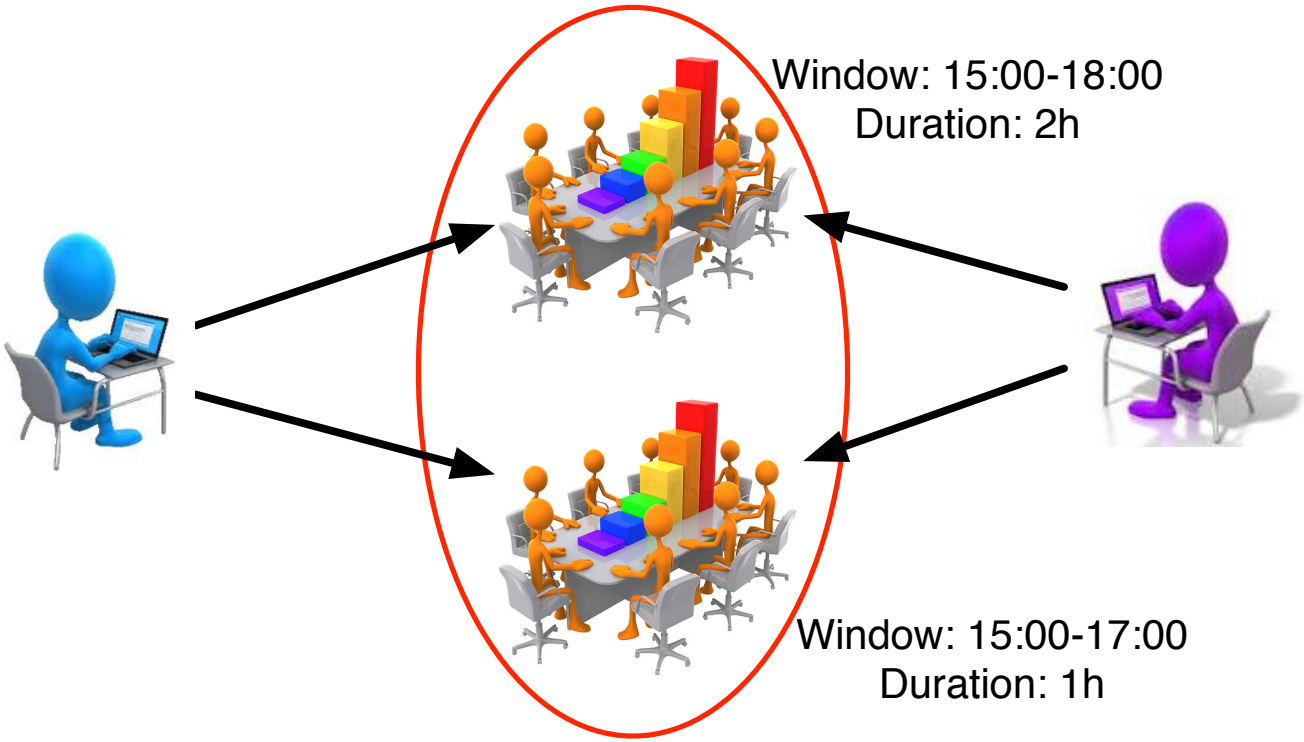
Meeting Scheduling

People might have various private preferences on meeting start times



Meeting Scheduling

Two meetings that share a participant cannot overlap



DCOP formalization for the meeting scheduling problem

- A set of agents representing participants
- A set of variables representing meeting starting times according to a participant.
- Hard Constraints:
  - Starting meeting times across different agents are equal
  - Meetings for the same agent are non-overlapping.
- Soft Constraints:
  - Represent agent preferences on meeting starting times.

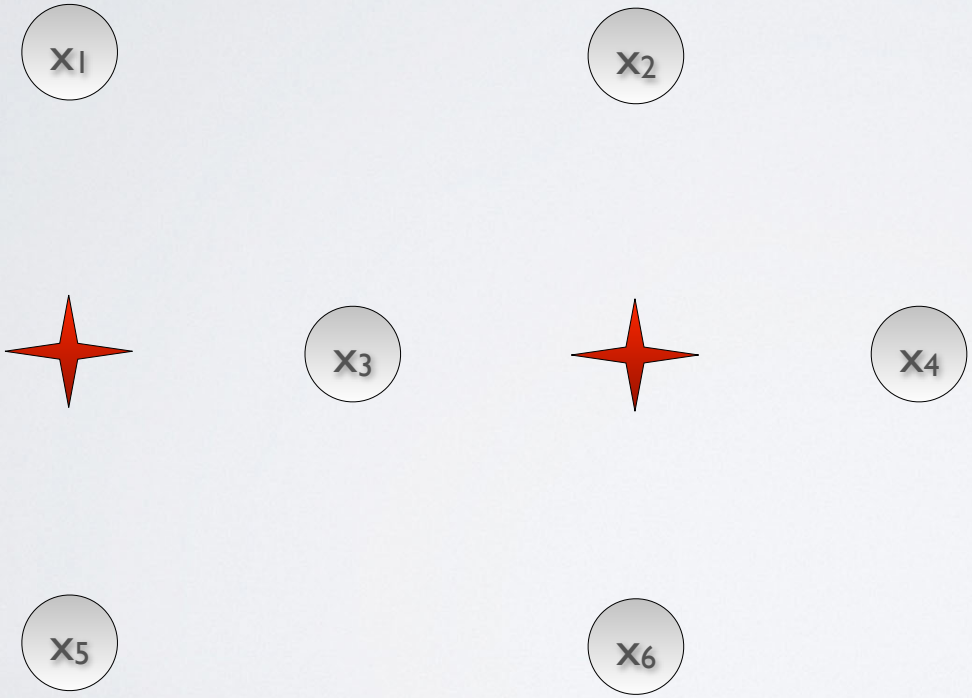
Objective: find a valid schedule for the meeting while maximizing the sum of individuals' preferences.

Target Tracking

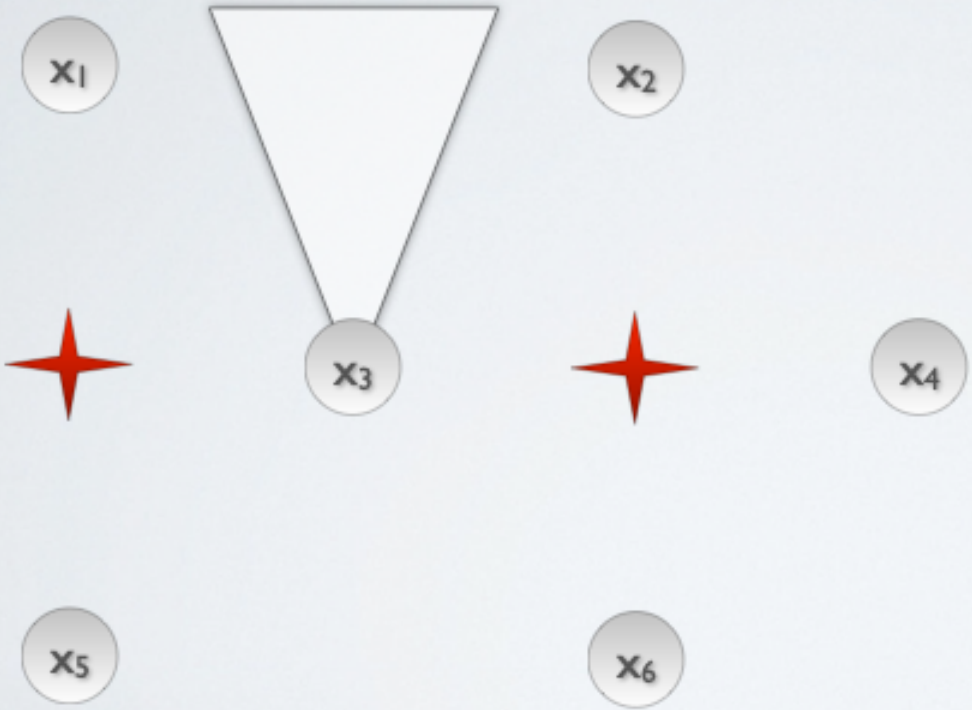
Motivation:

- Privacy
- Robustness
- Scalability

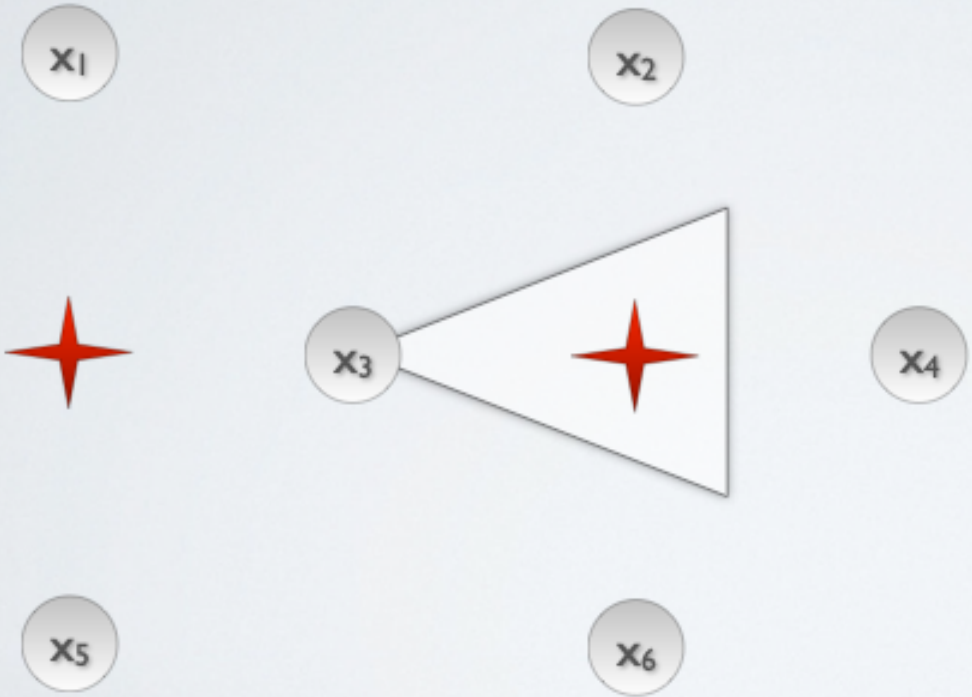
MOTIVATING DOMAIN: SENSOR NETWORK



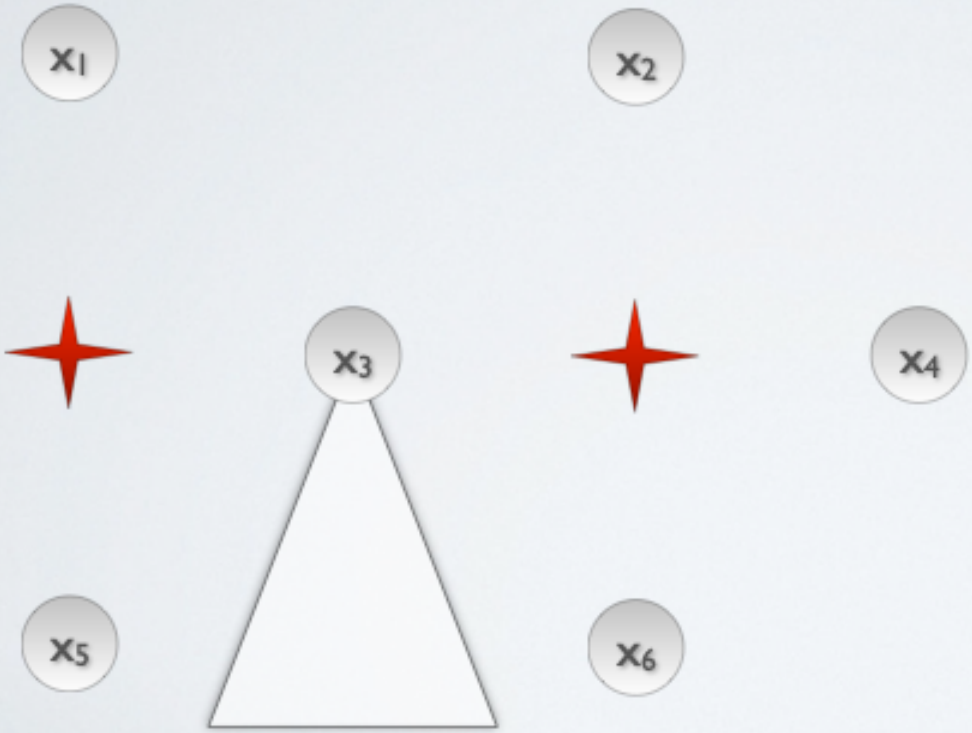
MOTIVATING DOMAIN: SENSOR NETWORK



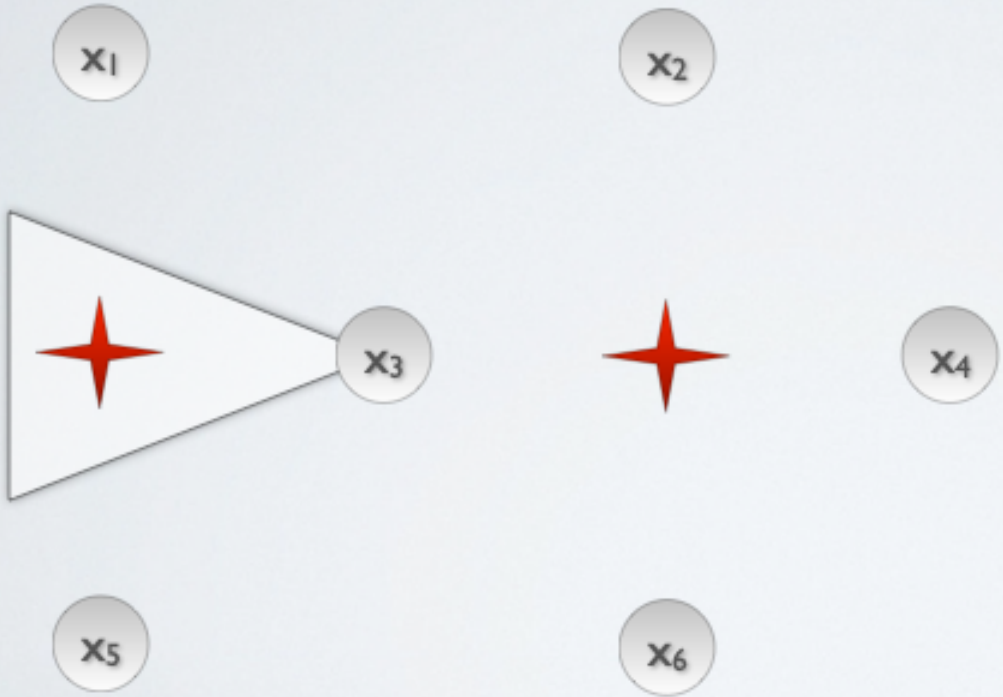
MOTIVATING DOMAIN: SENSOR NETWORK



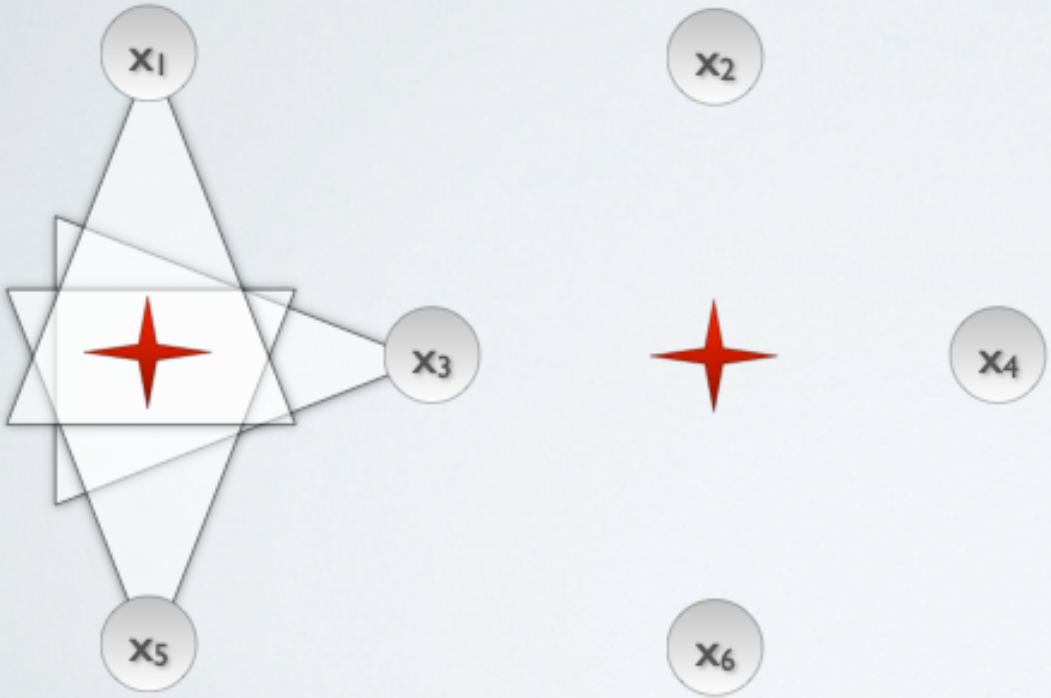
MOTIVATING DOMAIN: SENSOR NETWORK



MOTIVATING DOMAIN: SENSOR NETWORK



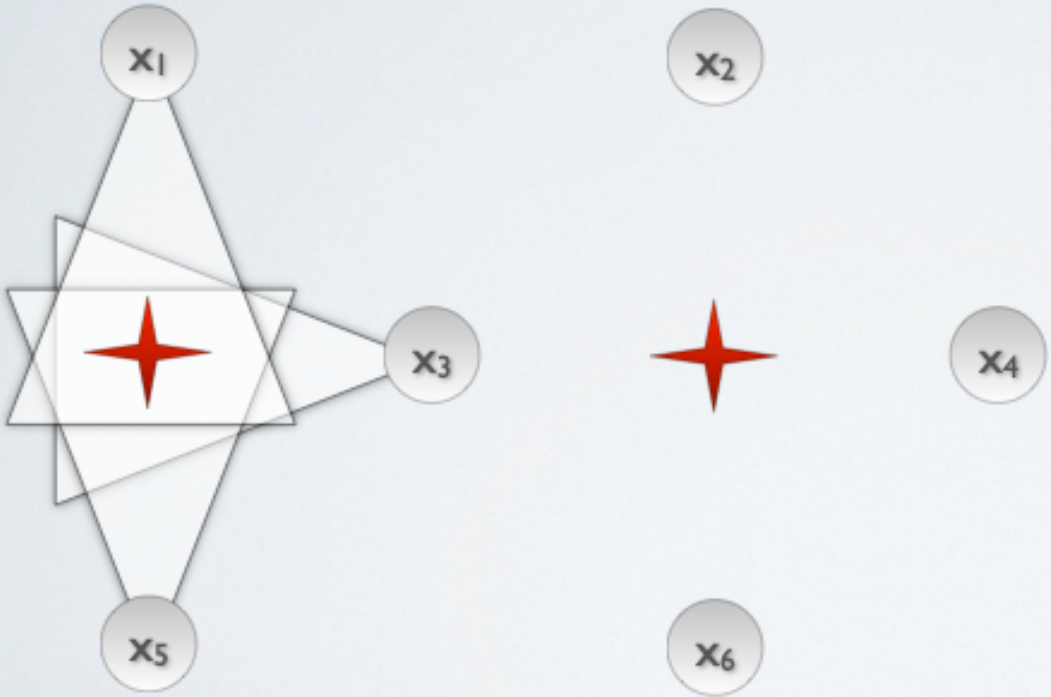
MOTIVATING DOMAIN: SENSOR NETWORK



x_1	x_3	x_5	Sat?
N	N	N	X
N	N	E	X
...			X
S	**W**	**N**	✓
...			X
W	W	W	X

Model the problem as a CSP

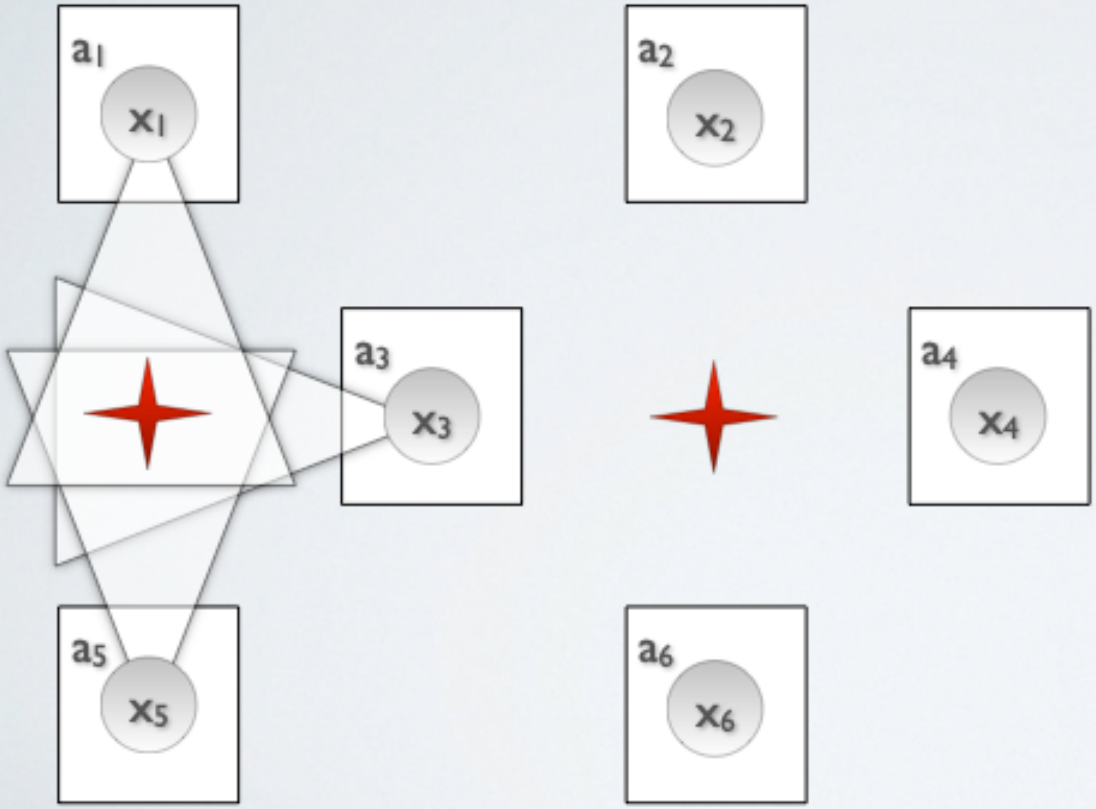
DCOP
DISTRIBUTED CONSTRAINT OPTIMIZATION



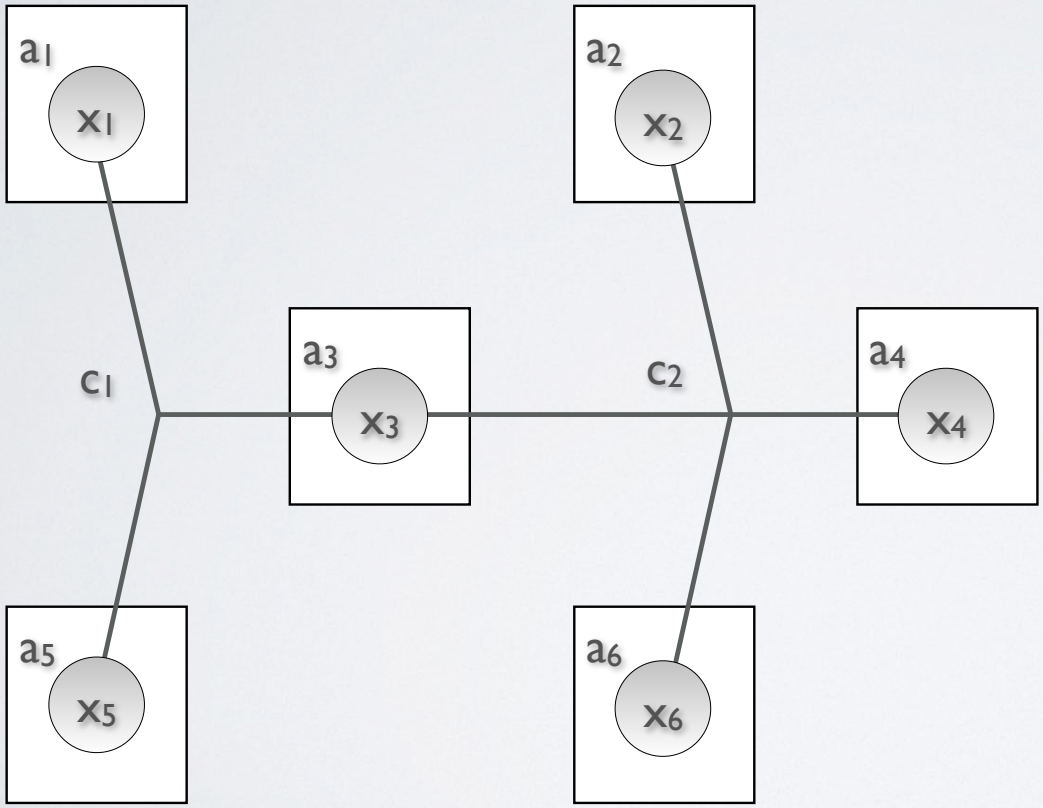
Imagine that each sensor is an autonomous agent.

How should this problem be modeled and solved in a decentralized manner?

DCOP
DISTRIBUTED CONSTRAINT OPTIMIZATION



DCOP
DISTRIBUTED CONSTRAINT OPTIMIZATION



Complete Algorithms


Always find an optimal solution


Exhibit an exponentially increasing coordination overhead


Very limited scalability on general problems.

Complete Algorithms

- **Completely decentralised**
  - Search-based.
    - Synchronous: SyncBB, AND/OR search
    - Asynchronous: ADOPT, NCBB and AFB.
  - Dynamic programming.
- **Partially decentralised**
  - OptAPO

Next, we focus on completely decentralised algorithms

Decentralised Complete Algorithms

Search-based

- Uses distributed search
- Exchange individual values
- Small messages but
  ...exponentially many

Representative: ADOPT [Modi et al., 2005]

Dynamic programming

- Uses distributed inference
- Exchange constraints
- Few messages but
  ...exponentially large

Representative: DPOP [Petcu and Faltings, 2005]

We will discuss DPOP in detail

DPOP

DPOP (Dynamic Programming Optimization Protocol) [Petcu and Faltings, 2005]:

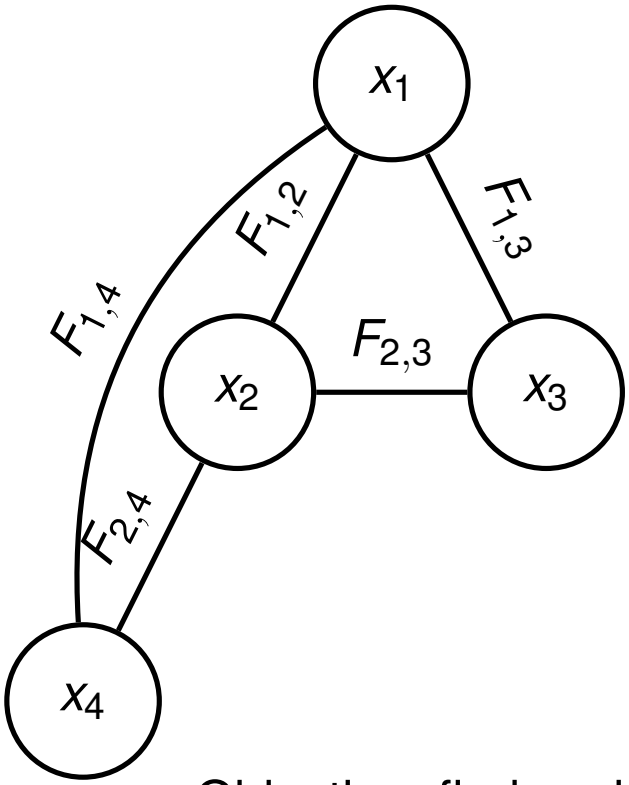
- Based on the dynamic programming paradigm.
- Special case of Bucket Tree Elimination Algorithm (BTE) [Dechter, 2003].

DFS arrangement

The same happens for DPOP!

- Before executing ADOPT, agents must be arranged in a depth first search (DFS) tree.
- DFS trees have been frequently used in optimization because they have two interesting properties:
  - Agents in different branches of the tree do not share any constraints;
  - Every constraint network admits a DFS tree.

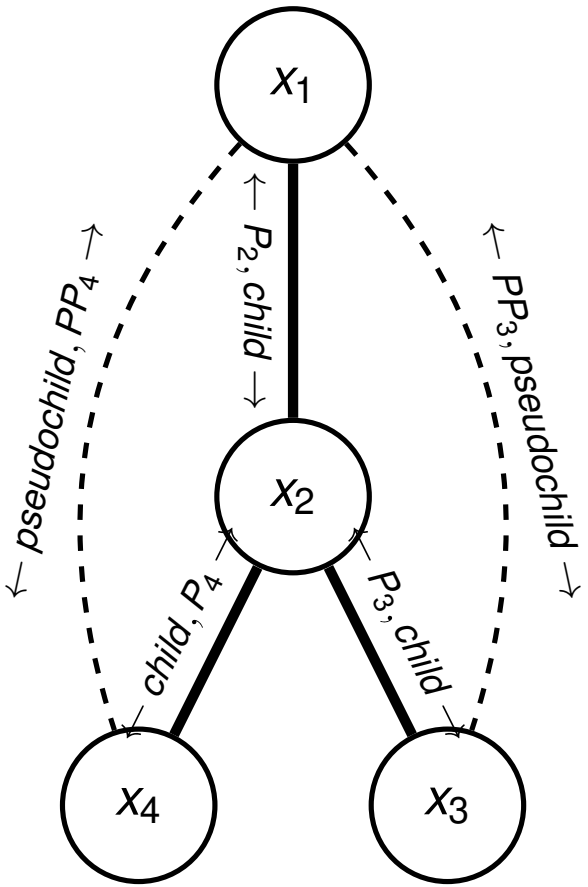
DPOP by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

$=>$

DFS arrangement



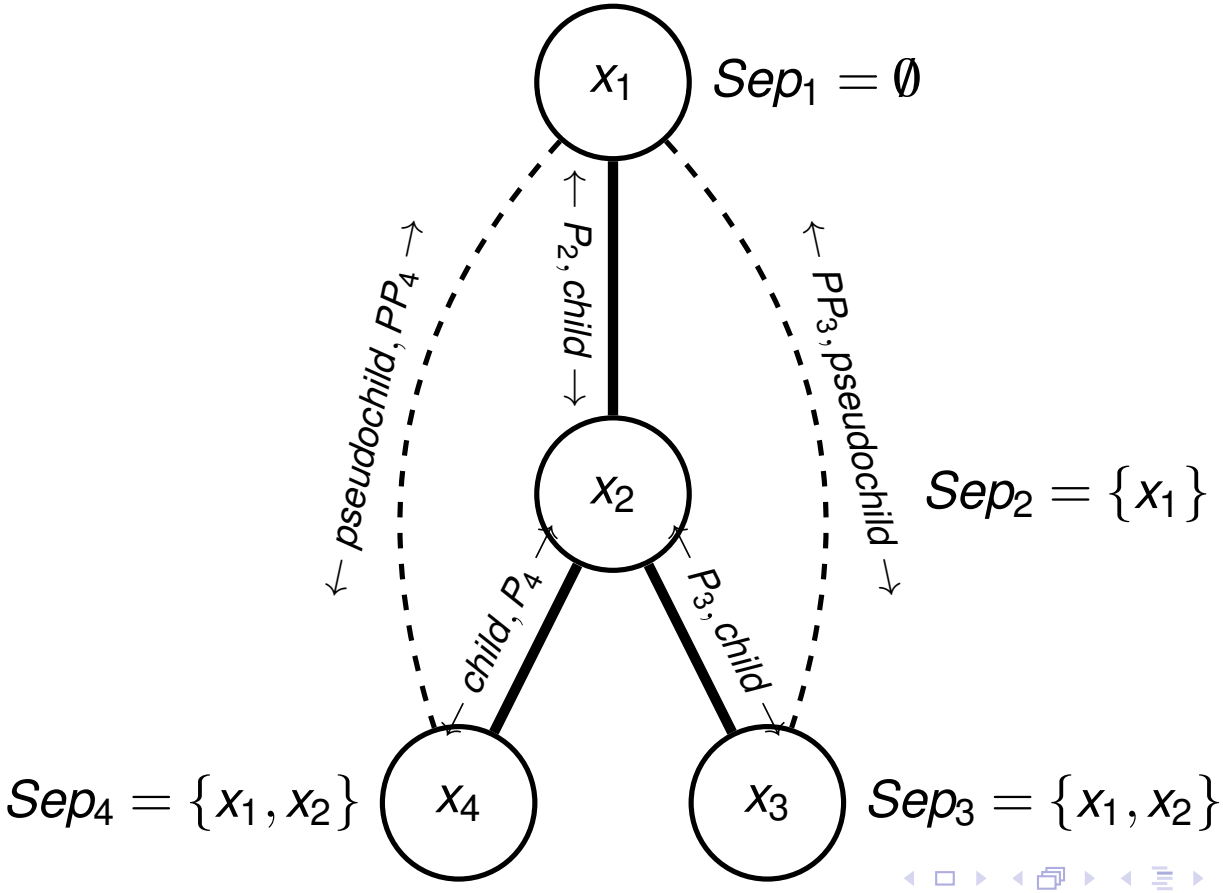
Objective: find assignment with maximal value

DPOP phases

Given a DFS tree structure, DPOP runs in two phases:

- *Util* propagation: agents exchange *util messages* up the tree.
  - Aim: aggregate all info so that root agent can choose optimal value
- *Value* propagation: agents exchange value messages down the tree.
  - Aim: propagate info so that all agents can make their choice given choices of ancestors

Sep_i: set of agents preceding A_i in the pseudo-tree order that are connected with A_i or with a descendant of A_i.



Util message

The *Util* message $U_{i \to j}$ that agent A_i sends to its parent A_j can be computed as:

$$U_{i \to j}(Sep_i) = \max_{x_i} \left(\bigotimes_{A_k \in C_i} U_{k \to i} \otimes \bigotimes_{A_p \in P_i \cup PP_i} F_{i,p} \right)$$

Size exponential in Sep_i

All incoming messages from children

Shared constraints with parents/pseudoparents

The $\otimes$ operator is a join operator that sums up functions with different but overlapping scores consistently.

Join operator

$F_{2,4}$	x_2	x_4
2	0	0
0	0	1
0	1	0
1	1	1

$F_{1,4}$	x_1	x_4
2	0	0
0	0	1
0	1	0
1	1	1

Add

$F_{1,4} \otimes F_{2,4}$

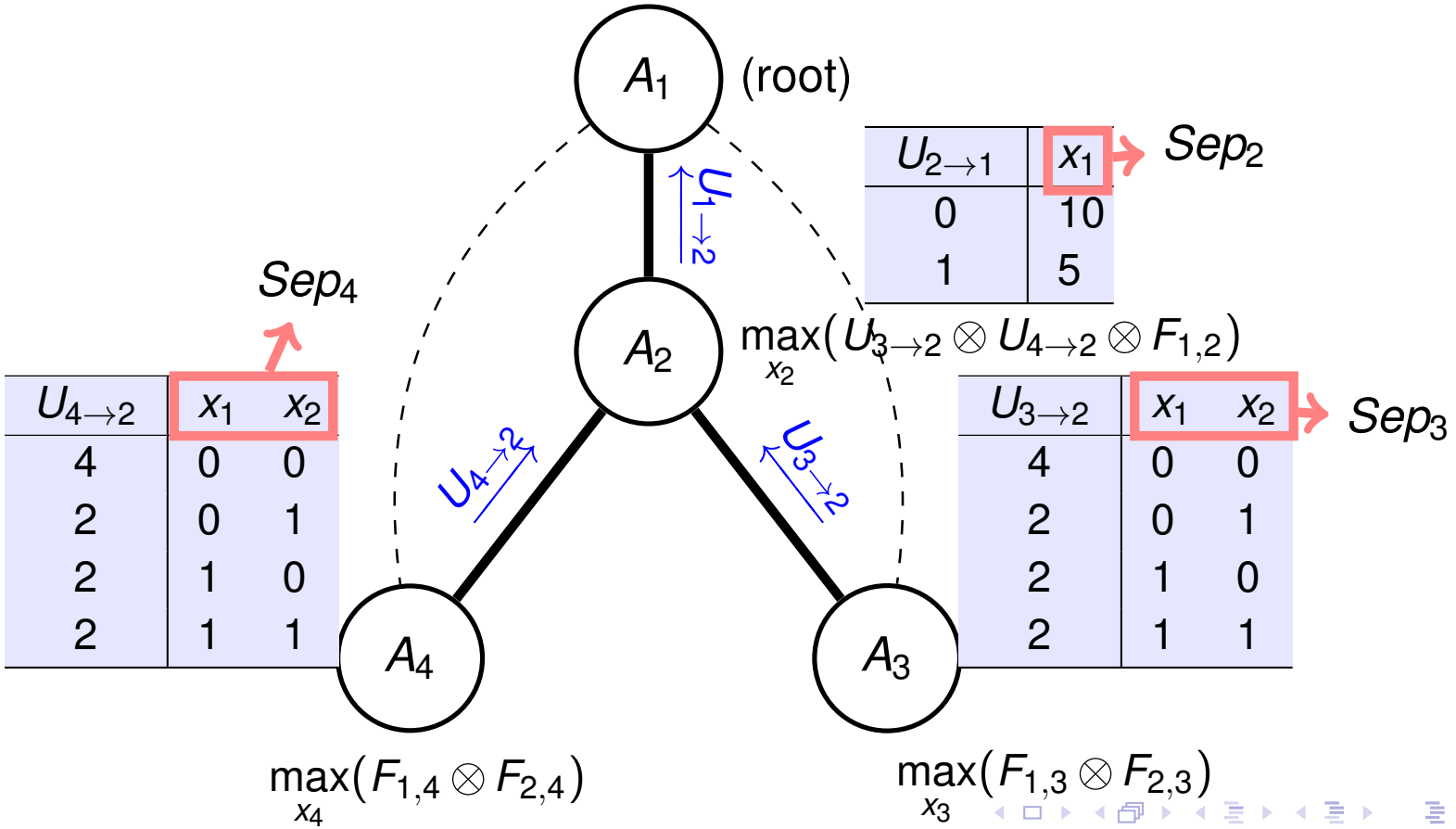
	x_1	x_2	x_4
4	0	0	0
0	0	0	1
2	0	1	0
1	0	1	1
2	1	0	0
2	1	0	1
0	1	1	0
2	1	1	1

Project out x_4

$\max_{\{x_4\}}(F_{1,4} \otimes F_{2,4})$

	x_1	x_2	x_4
max(4,0)	0	0	0
	0	0	1
max(2,1)	0	1	0
	0	1	1
max(2,2)	1	0	0
	1	0	1
max(0,2)	1	1	0
	1	1	1

Complexity exponential to the largest Sep_i.
Largest Sep_i = induced width of the DFS tree ordering used.



$U_{2\to 1}$	x_1
0	10
1	5

$U_{4\to 2}$	x_1	x_2
4	0	0
2	0	1
2	1	0
2	1	1

$U_{3\to 2}$	x_1	x_2
4	0	0
2	0	1
2	1	0
2	1	1

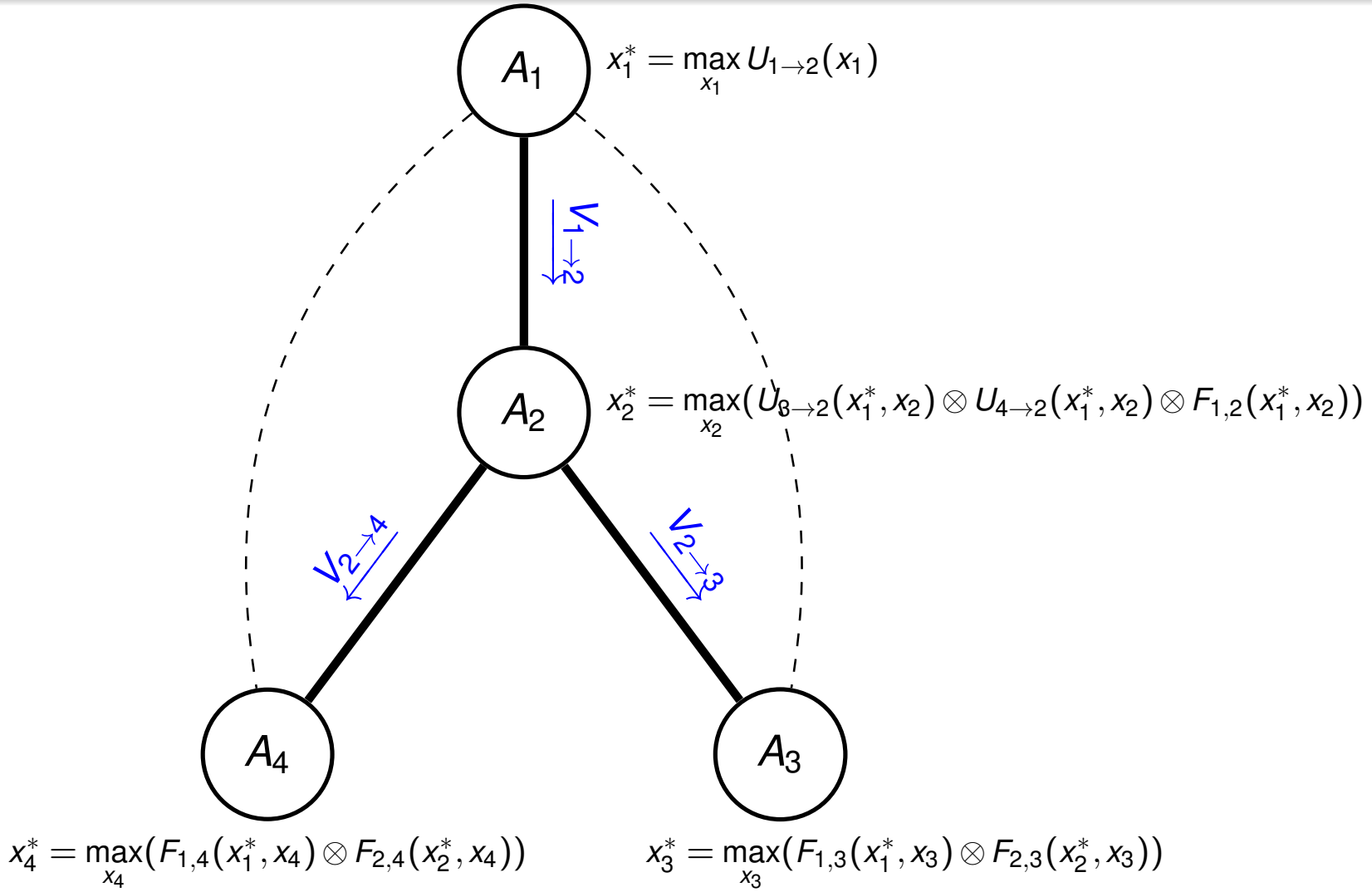
Value message

Keeping fixed the value of parent/pseudoparents, finds the value that maximizes the computed cost function in the util phase:

$$x_i^* = arg\max_{x_i} \left(\sum_{A_j \in C_i} U_{j \to i}(x_i, x_p^*) + \sum_{A_j \in P_i \cup PP_i} F_{i,j}(x_i, x_j^*) \right)$$

where $x_p^* = \bigcup_{A_j \in P_i \cup PP_i} \{x_j^*\}$ is the set of optimal values for A_i's parent and pseudoparents received from A_i's parent.
Propagates this value through children down the tree:

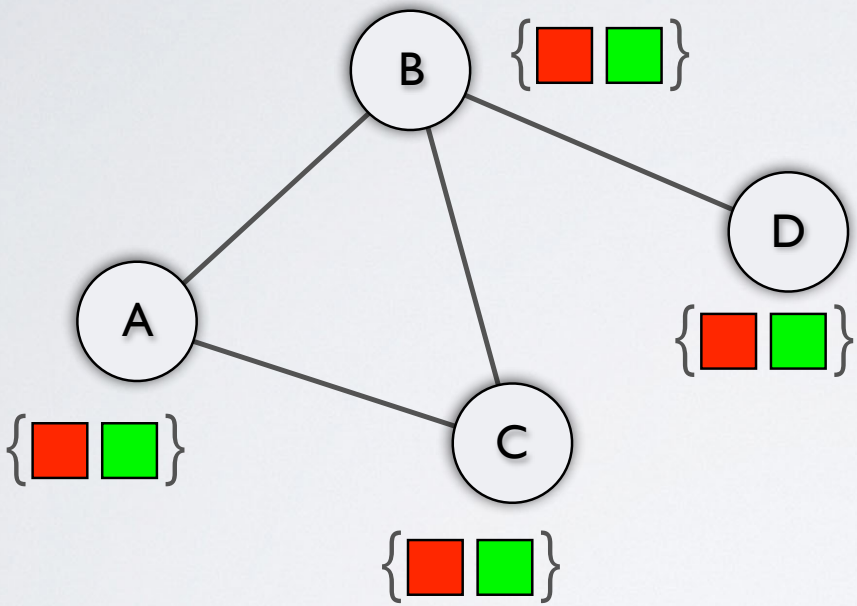
$$V_{i \to j} = \{x_i = x_i^*\} \cup \bigcup_{x_s \in Sep_i \cap Sep_j} \{x_s = x_s^*\}$$


A_1
$x_1^* = \max_{x_1} U_{1\to 2}(x_1)$
$V_{1\to 2}$
A_2
$x_2^* = \max_{x_2}(U_{3\to 2}(x_1^*, x_2) \otimes U_{4\to 2}(x_1^*, x_2) \otimes F_{1,2}(x_1^*, x_2))$
$V_{2\to 4}$
$V_{2\to 3}$
A_4
A_3
$x_4^* = \max_{x_4}(F_{1,4}(x_1^*, x_4) \otimes F_{2,4}(x_2^*, x_4))$
$x_3^* = \max_{x_3}(F_{1,3}(x_1^*, x_3) \otimes F_{2,3}(x_2^*, x_3))$

DPOP extensions

- MB-DPOP [Petcu and Faltings, 2007] trades-off message size against the number of messages.
- A-DPOP trades-off message size against solution quality [Petcu and Faltings, 2005(2)].

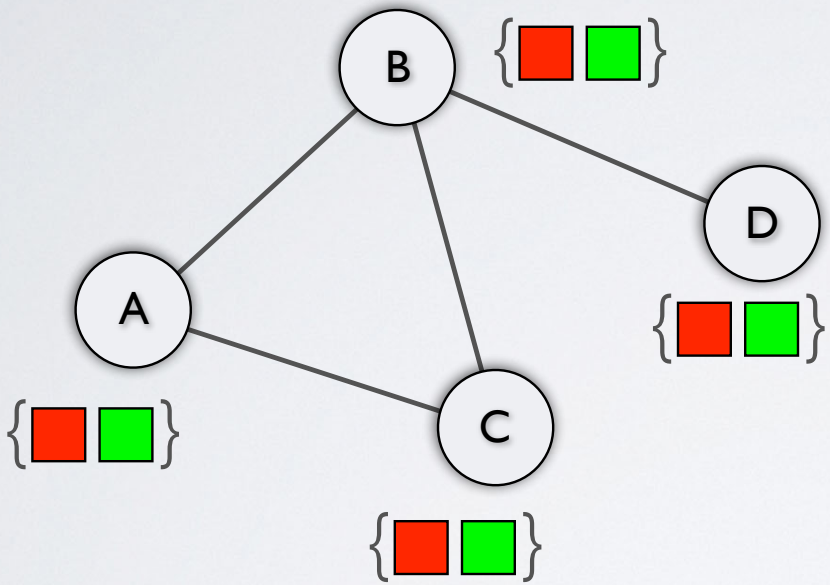
Example run of DPOP

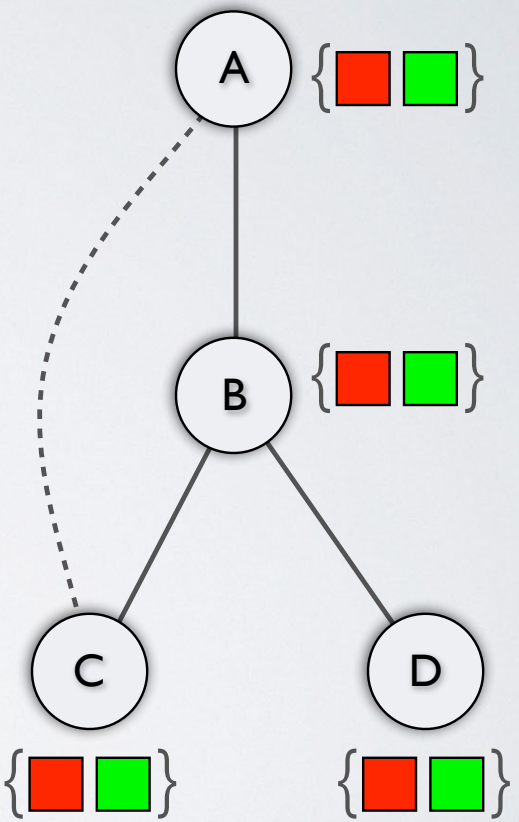


x_i	x_j	Cost (A,B)	Cost (A,C)	Cost (B,C)	Cost (B,D)
		5	5	5	3
		8	10	4	8
		20	20	3	10
		3	3	3	3

Katsutoshi Hirayama, Makoto Yokoo: Distributed Partial Constraint Satisfaction Problem. CP 1997: 222-236

PSEUDO-TREE

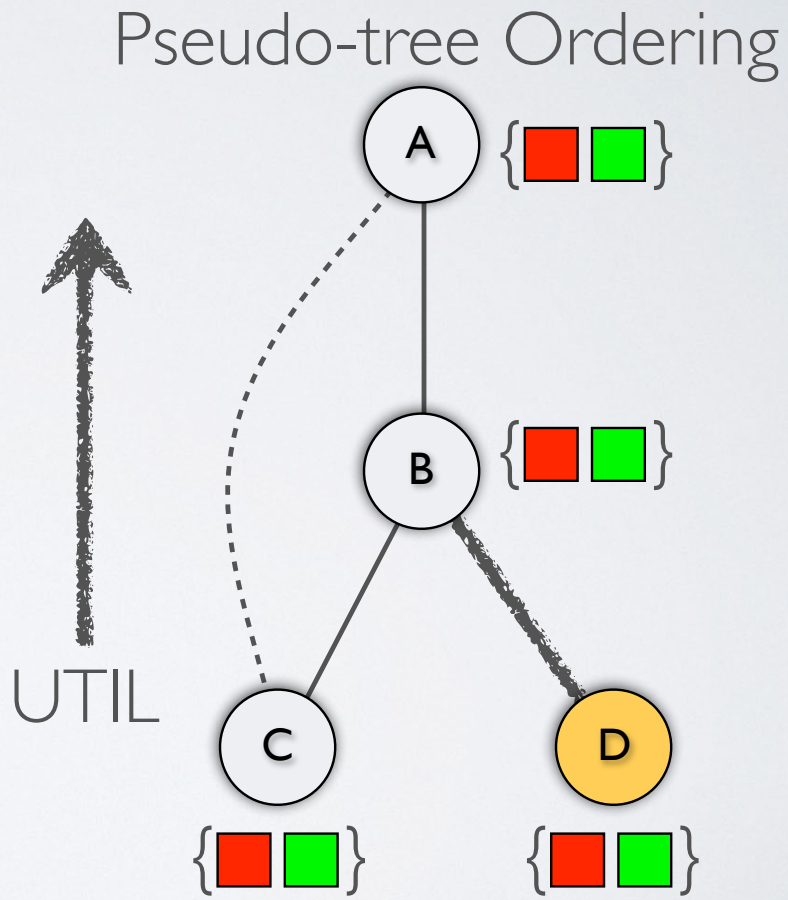




Definition: *A spanning tree of the constraint graph such that no two nodes in sibling subtrees share a constraint in the constraint graph*

DPOP

B	D	(B,D)
r	r	3
r	g	8
g	r	10
g	g	3



DPOP

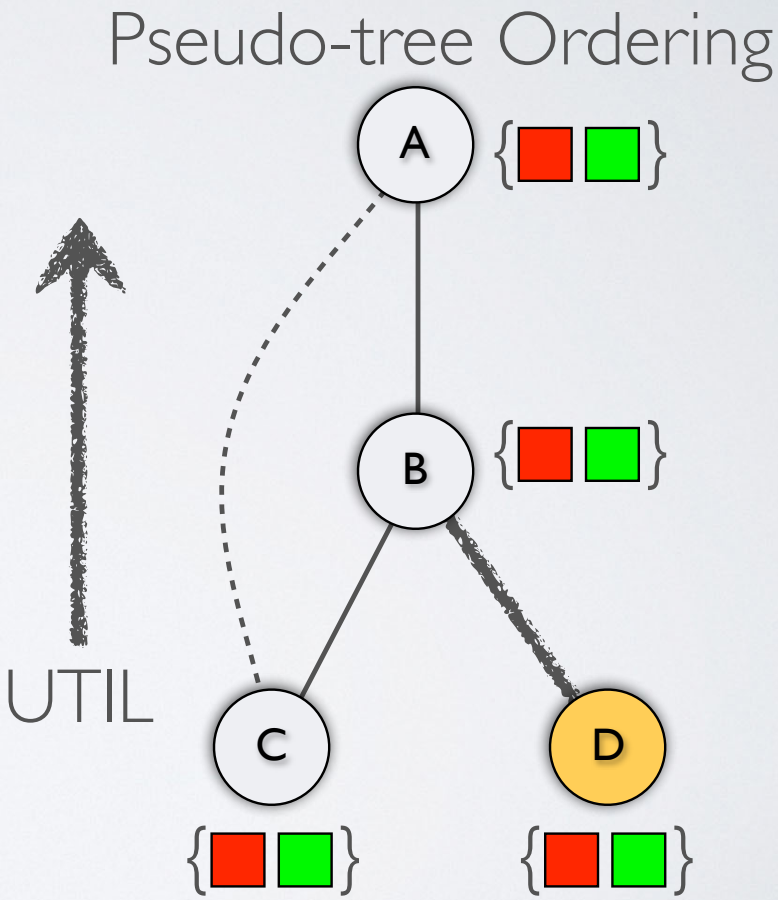
B	D	(B,D)
r	r	3
r	g	8
g	r	10
g	g	3

$\min\{3, 8\} = 3$

$\min\{10, 3\} = 3$

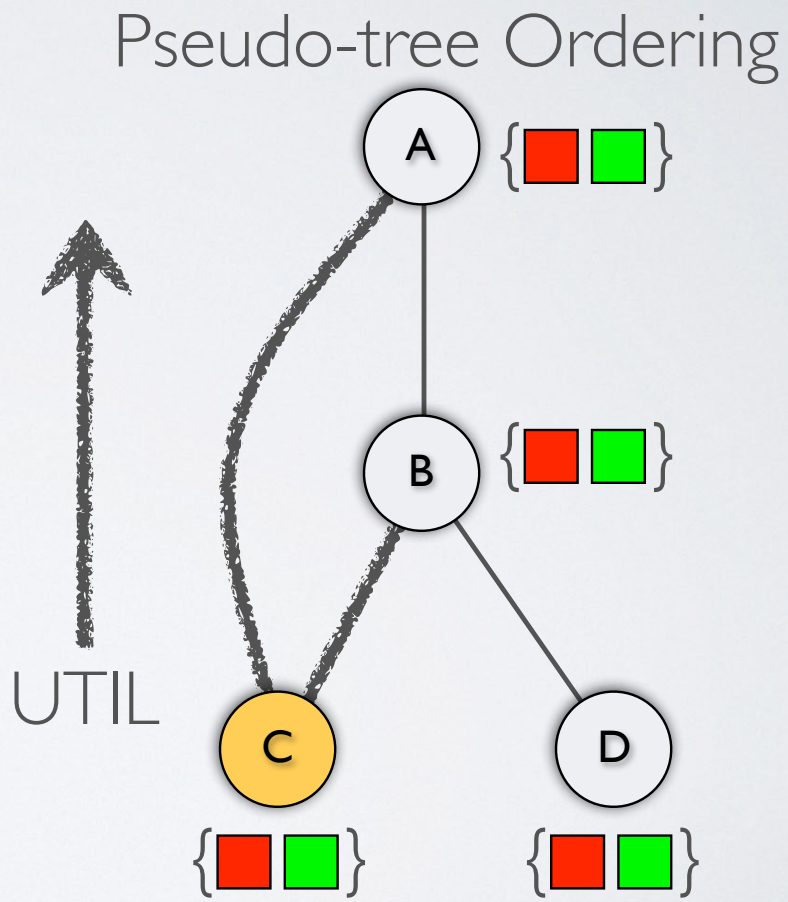
MSG to B

B	cost
r	3
g	3



DPOP

A	B	C	(B,C)	(A,C)	
r	r	r	5	5	10
r	r	g	4	8	12
r	g	r	3	5	8
r	g	g	3	8	11
g	r	r	5	10	15
g	r	g	4	3	7
g	g	r	3	10	13
g	g	g	3	3	6



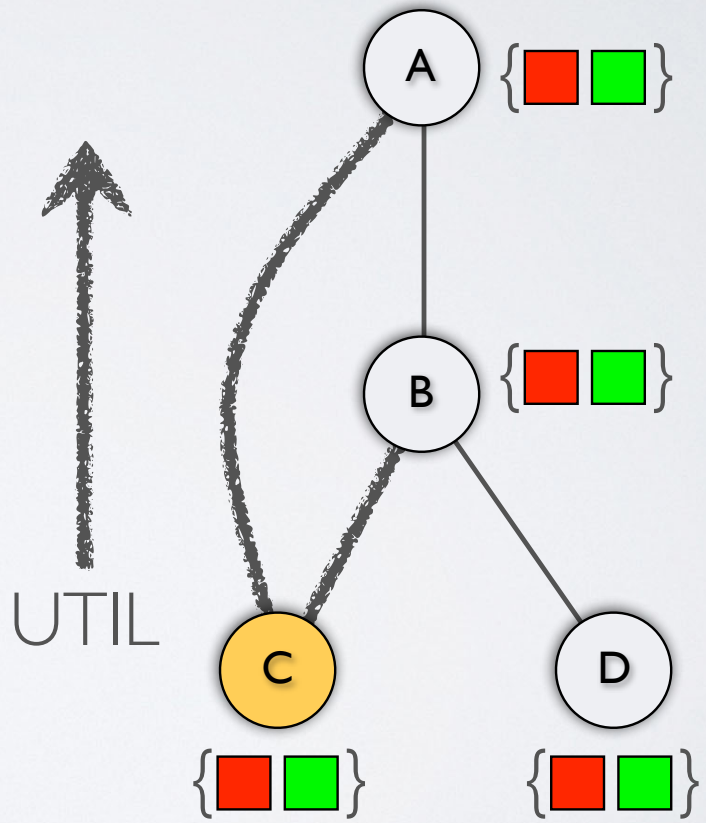
DPOP

A	B	C	(B,C)	(A,C)	
r	r	r	5	5	10
r	r	g	4	8	12
r	g	r	3	5	8
r	g	g	3	8	11
g	r	r	5	10	15
g	r	g	4	3	7
g	g	r	3	10	13
g	g	g	3	3	6

MSG to B

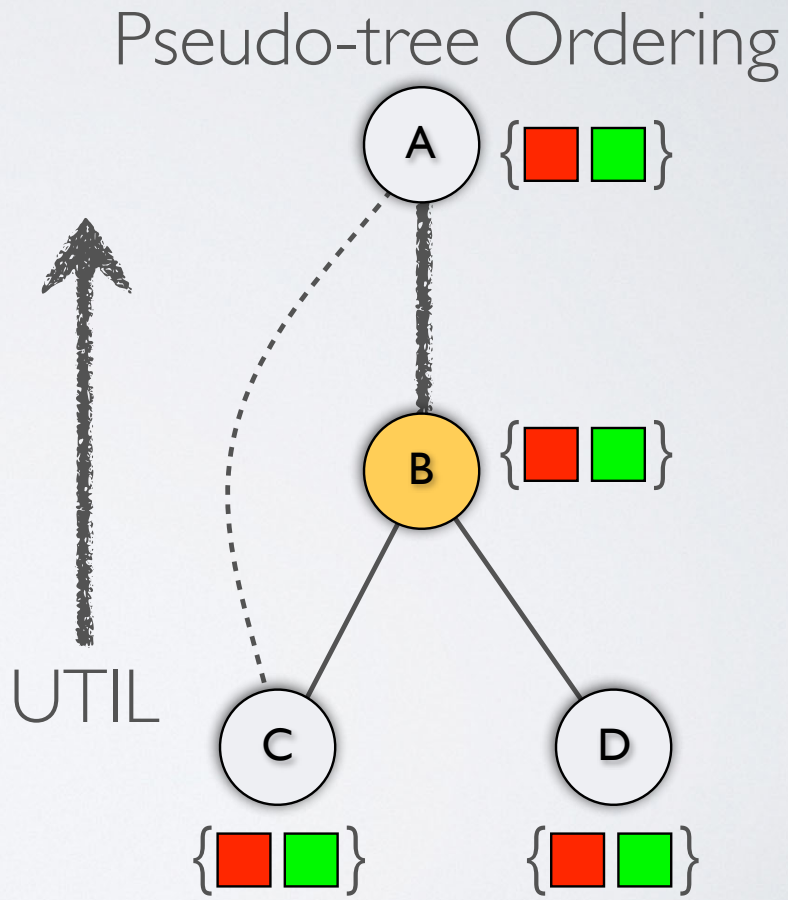
A	B	
r	r	10
r	g	8
g	r	7
g	g	6

Pseudo-tree Ordering



DPOP

A	B	(A,B)	Util C	Util D	
r	r	5	10	3	18
r	g	8	8	3	19
g	r	20	7	3	30
g	g	3	6	3	12



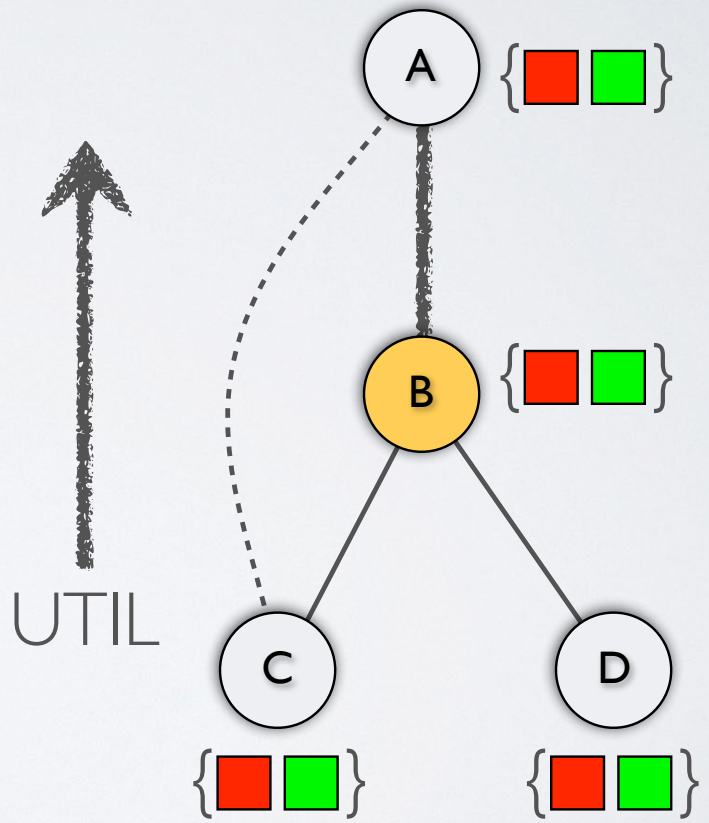
DPOP

A	B	(A,B)	Util C	Util D	
r	r	5	10	3	18
r	g	8	8	3	19
g	r	20	7	3	30
g	g	3	6	3	12

MSG to A

A	cost
r	18
g	12

Pseudo-tree Ordering

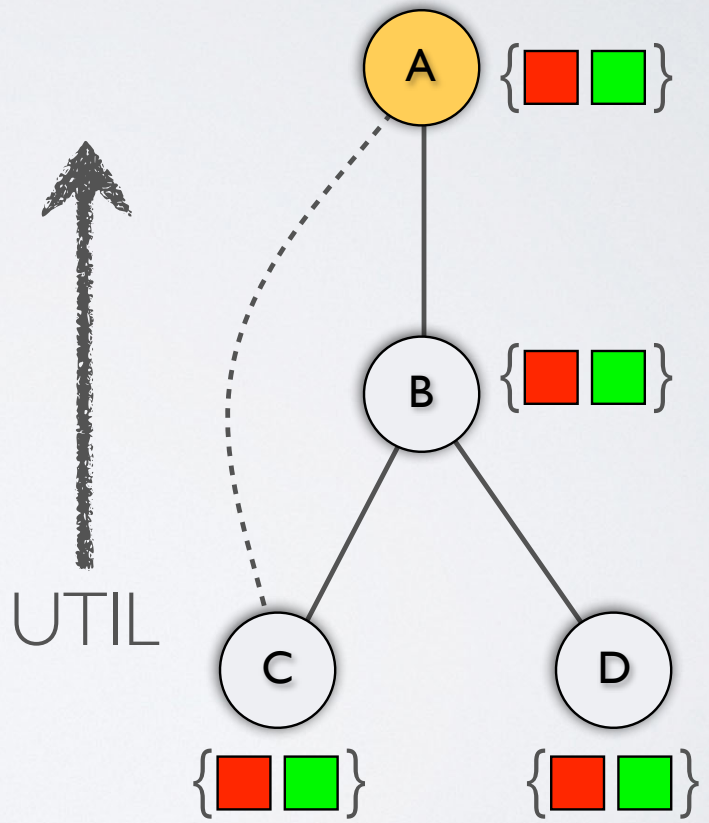


DPOP

A	cost
r	18
g	12

optimal cost = 12

Pseudo-tree Ordering

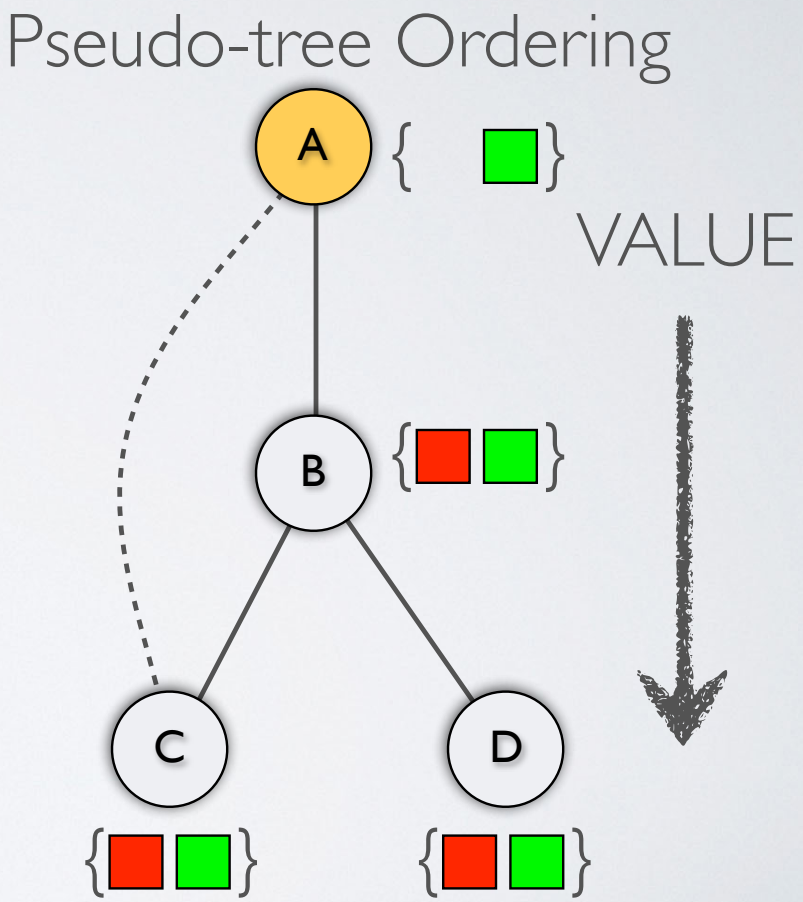


DPOP

A	cost
r	18
g	12

- Select value for A = 'g'
- Send MSG A = 'g' to agents B and C

Alternative version in which value messages $x_i = x_i^*$ are sent to children and pseudo-children

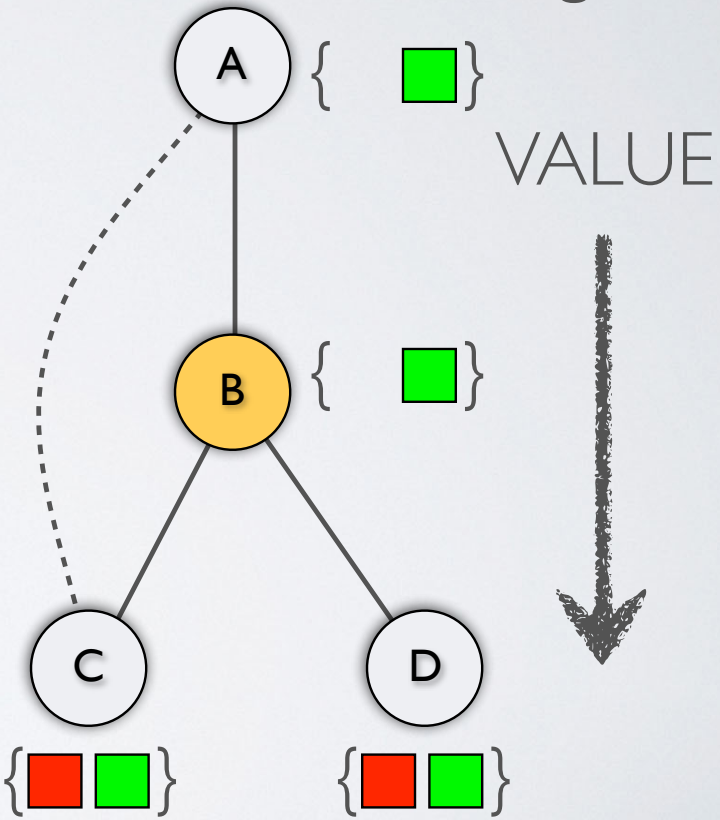


DPOP

A	B	(A,B)	Util C	Util D	
r	r	5	10	3	18
r	g	8	8	3	19
g	r	20	7	3	30
g	**g**	3	6	3	12

- Select value for B = 'g'
- Send MSG B = 'g' to agents C and D

Pseudo-tree Ordering

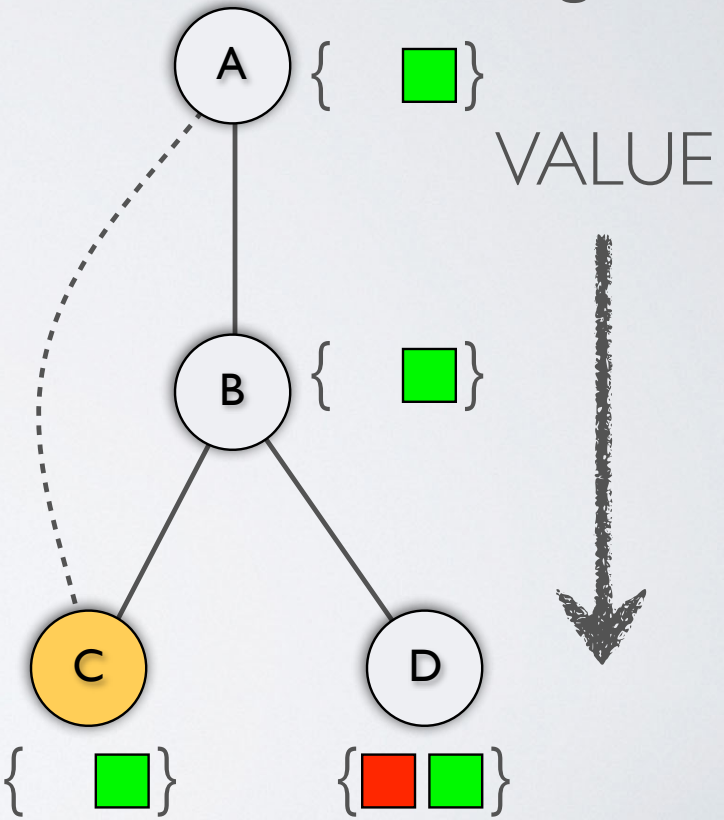


DPOP

A	B	C	(B,C)	(A,C)	
r	r	r	5	5	10
r	r	g	4	8	12
r	g	r	3	5	8
r	g	g	3	8	11
g	r	r	5	10	15
g	r	g	4	3	7
g	g	r	3	10	13
g	g	**g**	3	3	6

- Select value for C = 'g'

Pseudo-tree Ordering

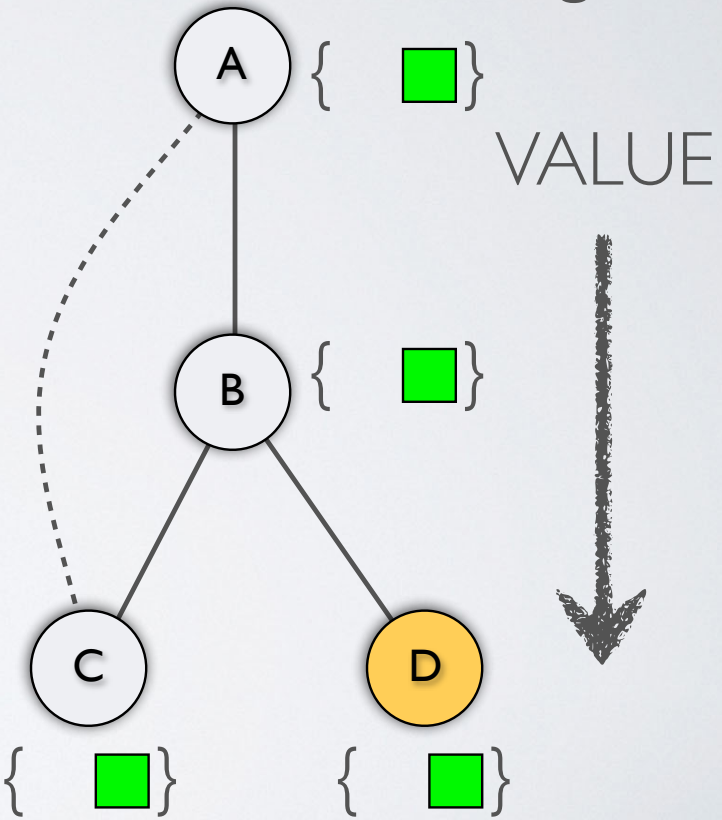


DPOP

B	D	(B,D)
r	r	3
r	g	8
g	r	10
g	**g**	3

- Select value for D = 'g'

Pseudo-tree Ordering



Why Approximate Algorithms

"Very often optimality in practical applications is not achievable"

Approximate algorithms

- Sacrify optimality in favor of computational and communication efficiency
- Well-suited for large scale distributed applications:
  - sensor networks
  - mobile robots

Centralized Local Greedy approaches

- Start from a random assignment for all the variables
- Do local moves if the new assignment improves the value (local gain)
- Local: changing the value of a small set of variables (in most case just one)
- The search stops when there is no local move that provides a positive gain, i.e., when the process reaches a local maximum.

Distributed Local Greedy approaches

When operating in a decentralized context:

- **Problem:** Out-of-date local knowledge
  - Assumption that other agents do not change their values
  - A greedy local move might be harmful/useless
- **Solution:**
  - Stochasticity on the decision to perform a move (DSA)
  - Coordination among neighbours on who is the agent that should move (MGM)

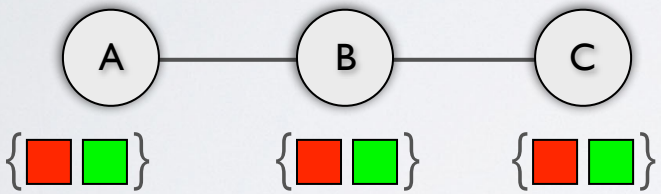
Distributed Stochastic Algorithm (DSA)

Activation probability *to mitigate issues with parallel executions*

[S. Fitzpatrick and L. Meetrens, 2003]

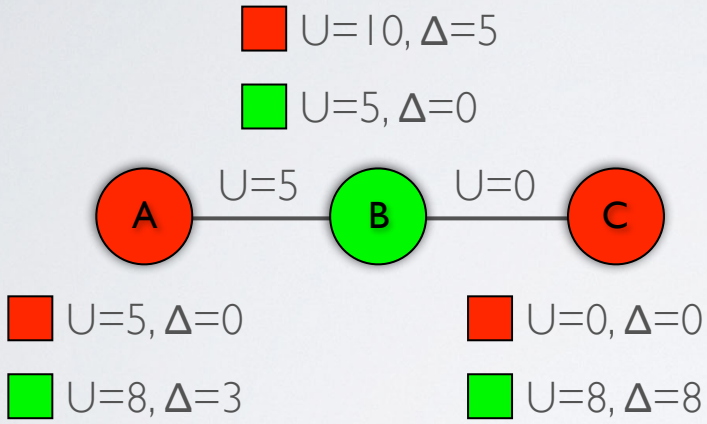
- Initialize agents with a random assignment and communicate values to neighbors
- Each agent:
  - Generates a random number and executes only if it is less than the activation probability
  - When executing choose a value for the variable such that the local gain is maximized
  - Communicate and receive possible variables change to/from neighbors

Example run of DSA



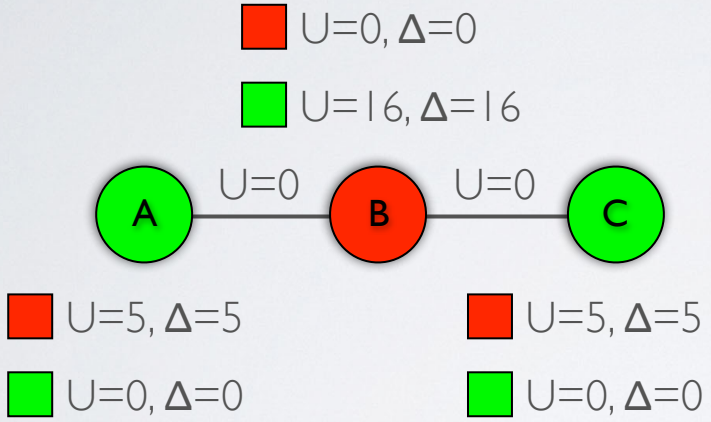
x_i	x_j	Utility (A,B)	Utility (B,C)
		5	5
		5	0
		0	0
		8	8

Example run of DSA



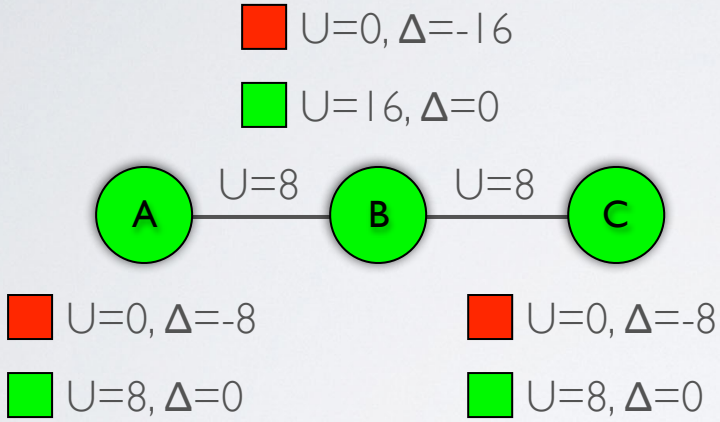
x_i	x_j	Utility (A,B)	Utility (B,C)
		5	5
		5	0
		0	0
		8	8

Example run of DSA



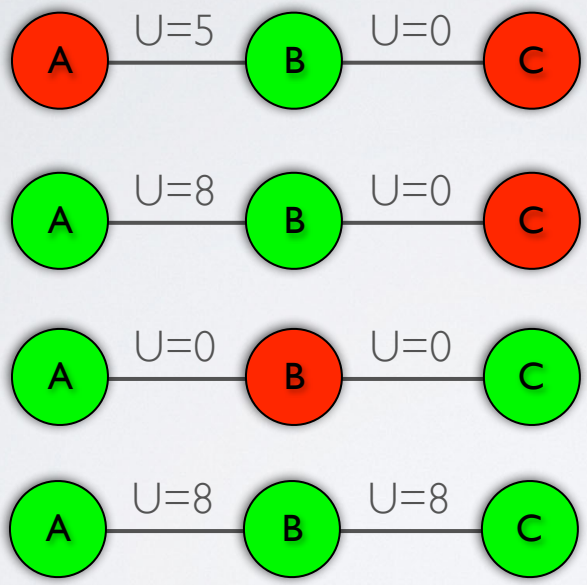
x_i	x_j	Utility (A,B)	Utility (B,C)
		5	5
		5	0
		0	0
		8	8

Example run of DSA



x_i	x_j	Utility (A,B)	Utility (B,C)
		5	5
		5	0
		0	0
		8	8

Example run of DSA



One possible execution trace

x_i	x_j	Utility (A,B)	Utility (B,C)
		5	5
		5	0
		0	0
		8	8

DSA-1: discussion


Extremely low computation/communication


Shows an anytime property (not guaranteed)


Activation probability:

- Must be tuned
- Domain dependent (no general rule)

Maximum Gain Message (MGM-1)

Coordination *among neighbours to decide which single agent can perform the move.*

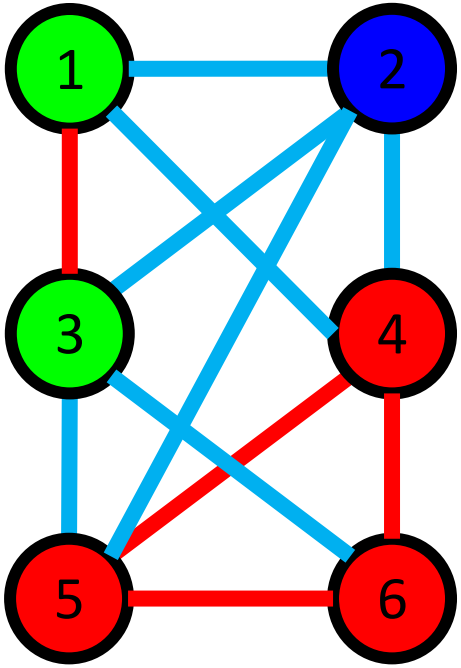
[R. T. Maheswaran et al., 2004]

- Initialize agents with a random assignment and communicate values to neighbors
- Each agent:
  - Compute and exchange possible gains
  - Agent with maximum (positive) gain executes
  - Communicate and receive possible variables changes to/from neighbors

Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

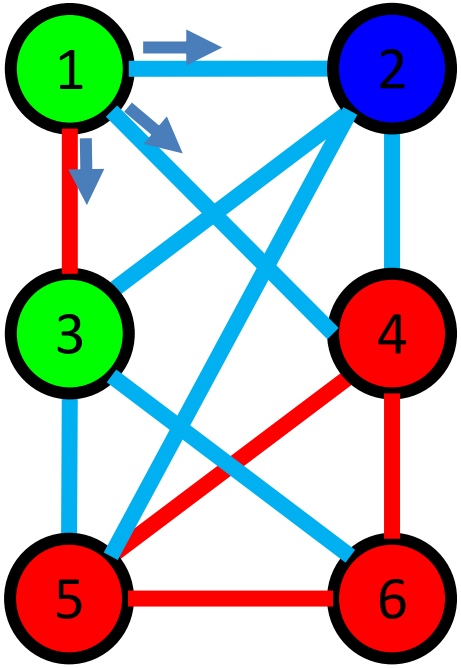
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

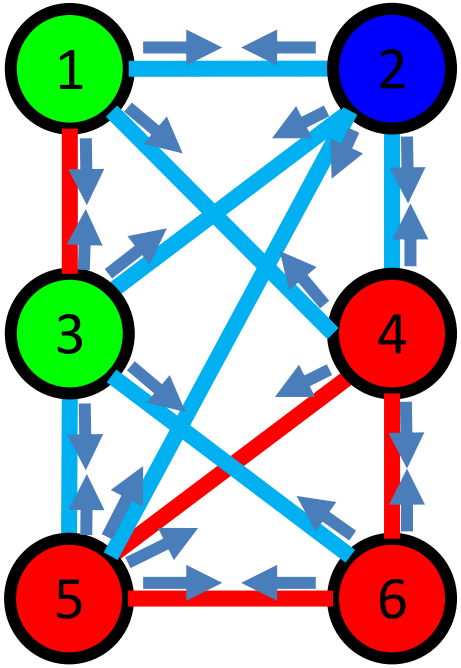
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

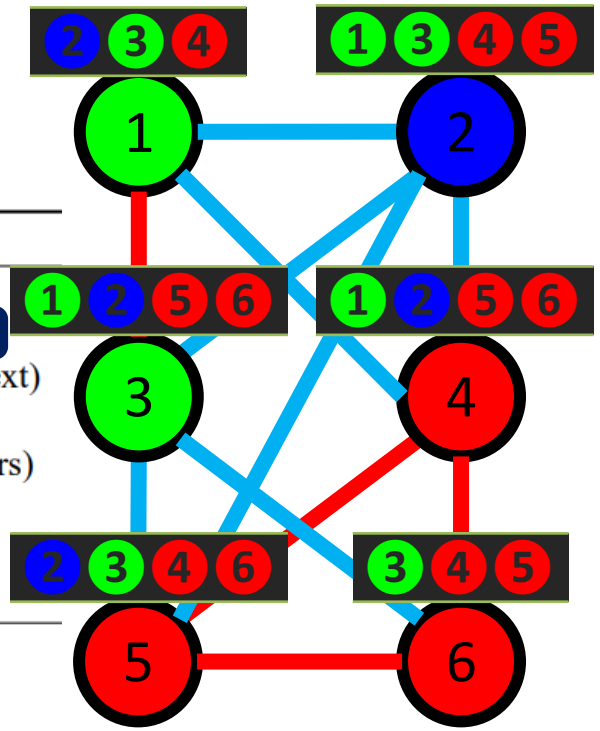
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

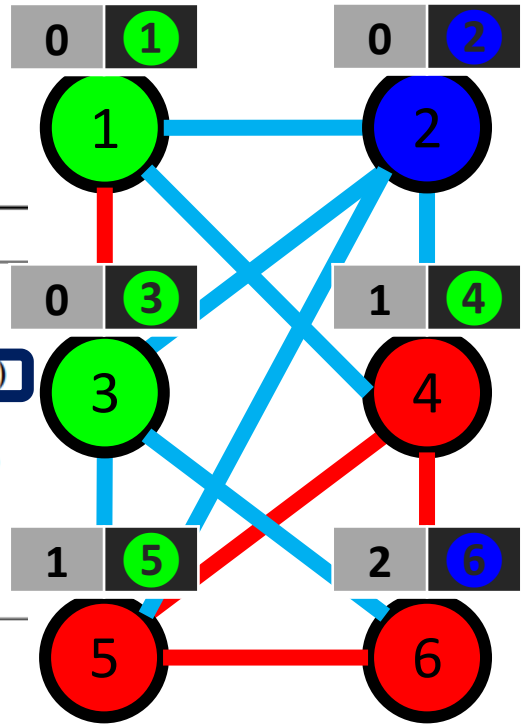
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

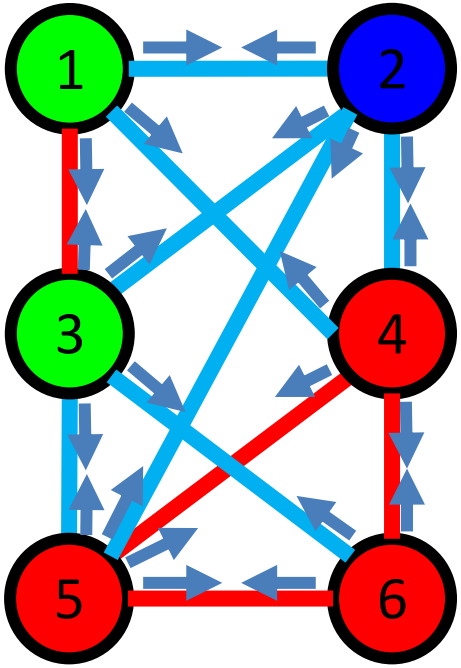
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

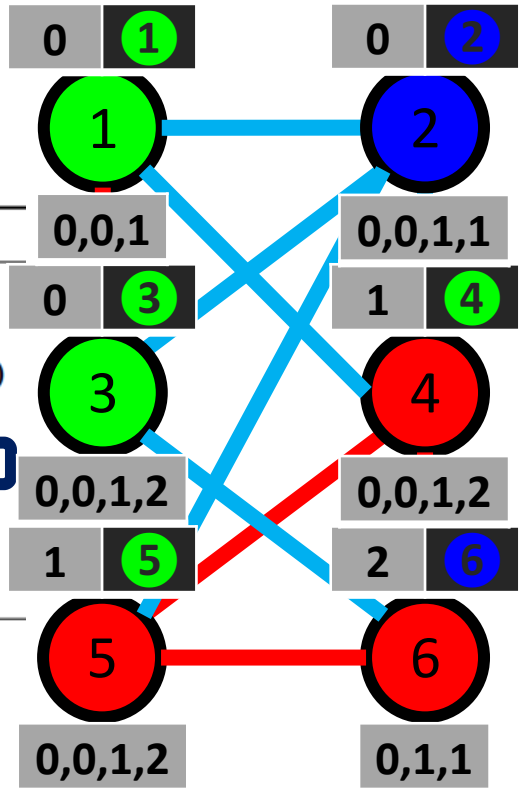
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

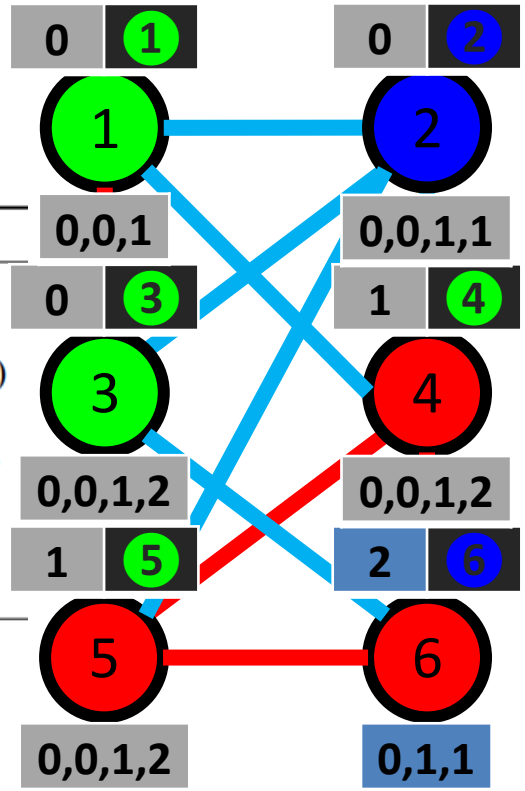
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

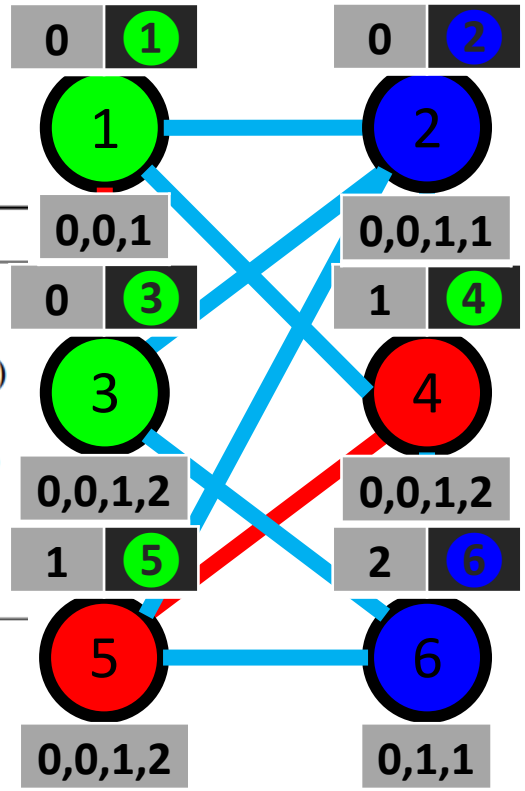
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

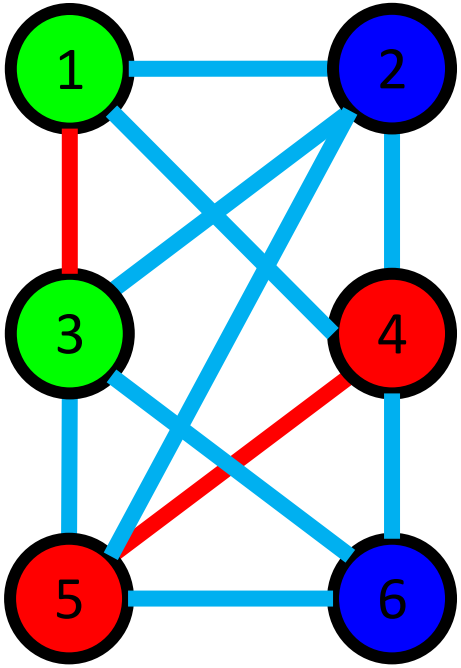
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

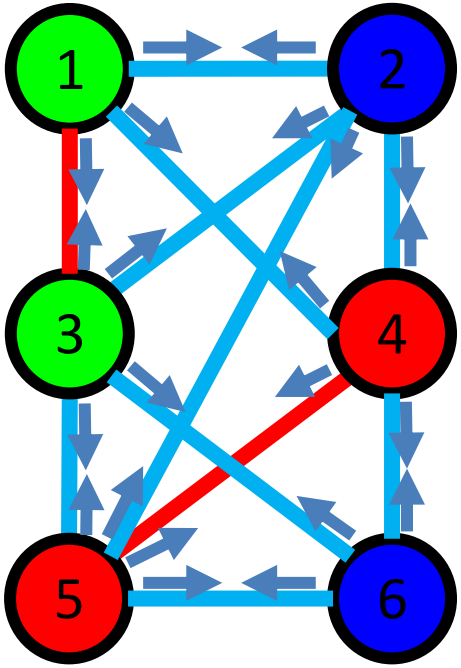
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

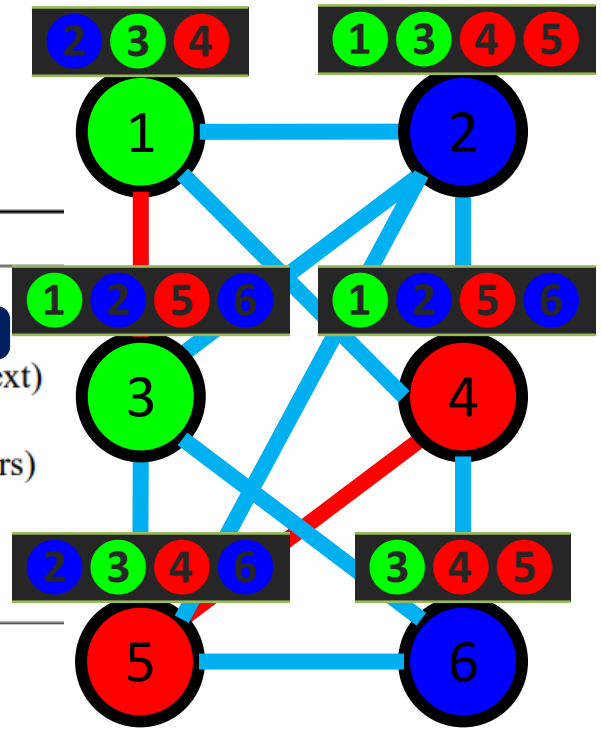
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

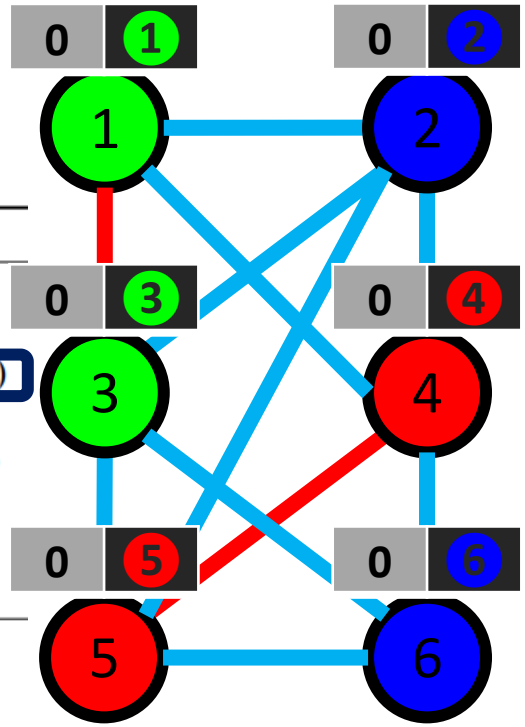
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

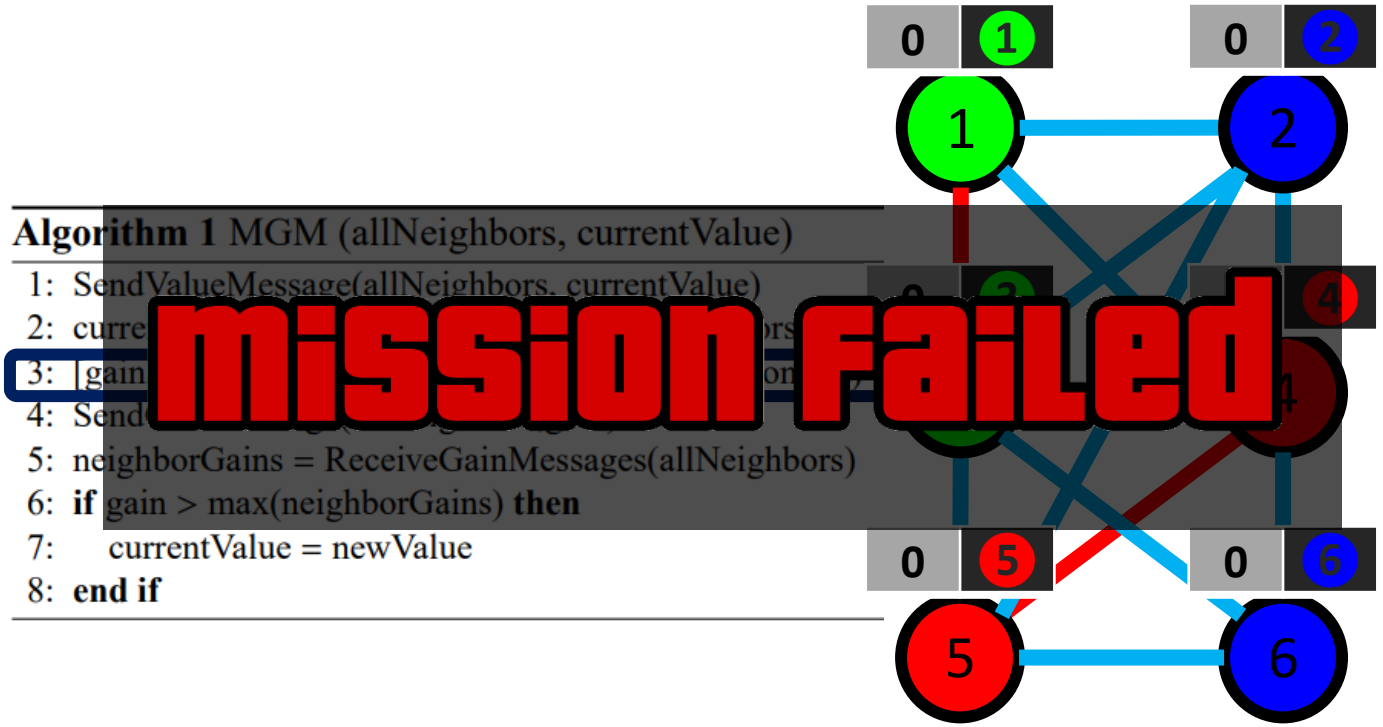
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Example run of MGM-1

Algorithm 1 MGM (allNeighbors, currentValue)

```
1: SendValueMessage(allNeighbors, currentValue)
2: curre                                          ors
3: gain                                           on
4: Send
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```

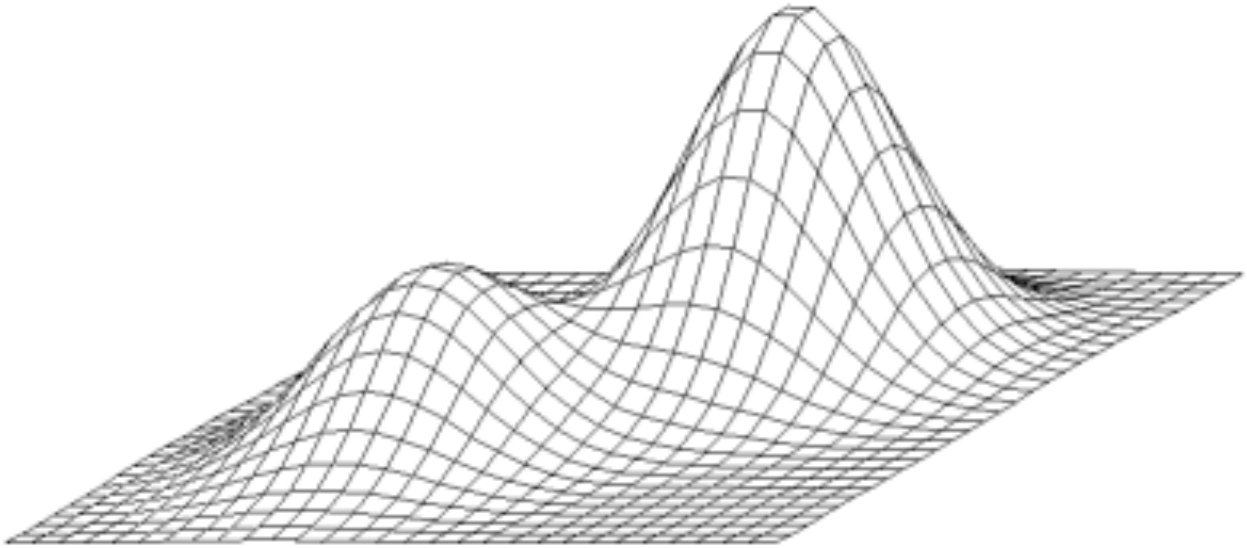


MISSION FAILED

Why?

Local maximum vs. global maximum

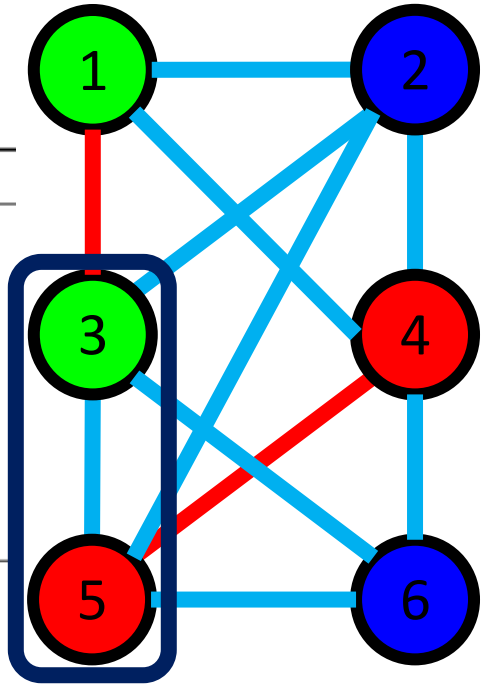
MGM has reached a local maximum and, changing only a value, cannot escape from it



Local maximum vs. global maximum

Algorithm 1 MGM (allNeighbors, currentValue)

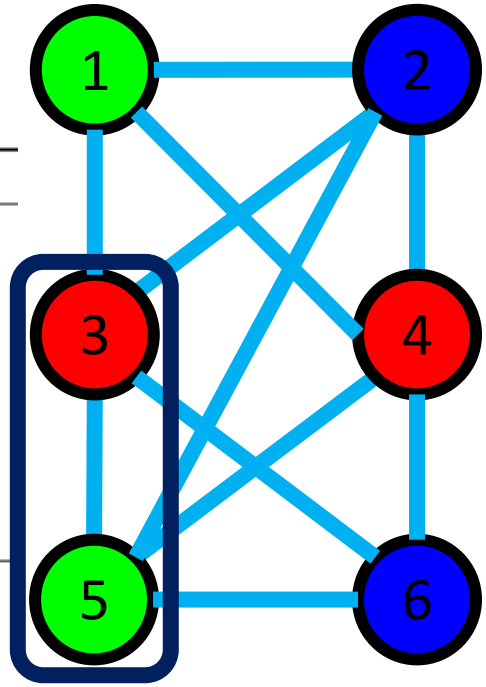
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Local maximum vs. global maximum

Algorithm 1 MGM (allNeighbors, currentValue)

```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



MGM-1: discussion

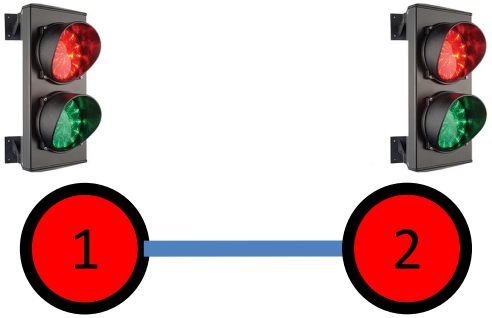
= More communication than DSA but still linear

= Empirically similar to DSA

 Guaranteed to be anytime

 Does not require any parameter tuning.

MGM vs DSA



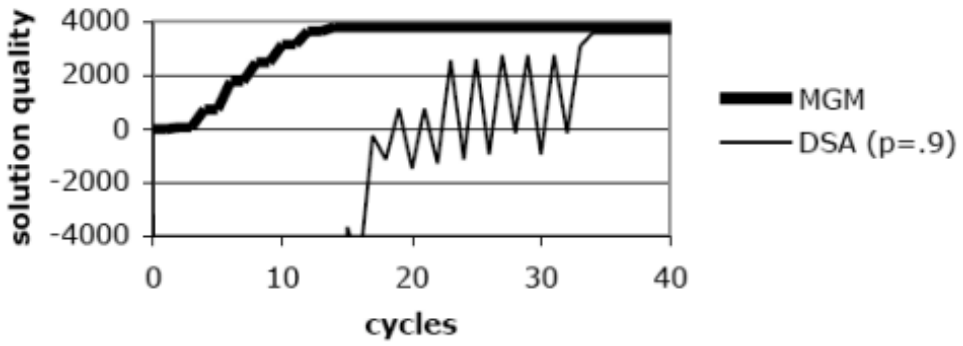
$$U(red, red) = 0$$
$$U(red, green) = U(green, red) = 1$$
$$U(green, green) = -1000$$

If both agents are allowed to alter their value in the same round, we could end up in the adverse state (green, green)

When using DSA, there is always a positive probability for any time horizon that (green, green) will be the resulting assignment

MGM vs DSA

When applying MGM, the global utility strictly increases at each round until convergence



Decentralised greedy approaches


Very little memory and computation

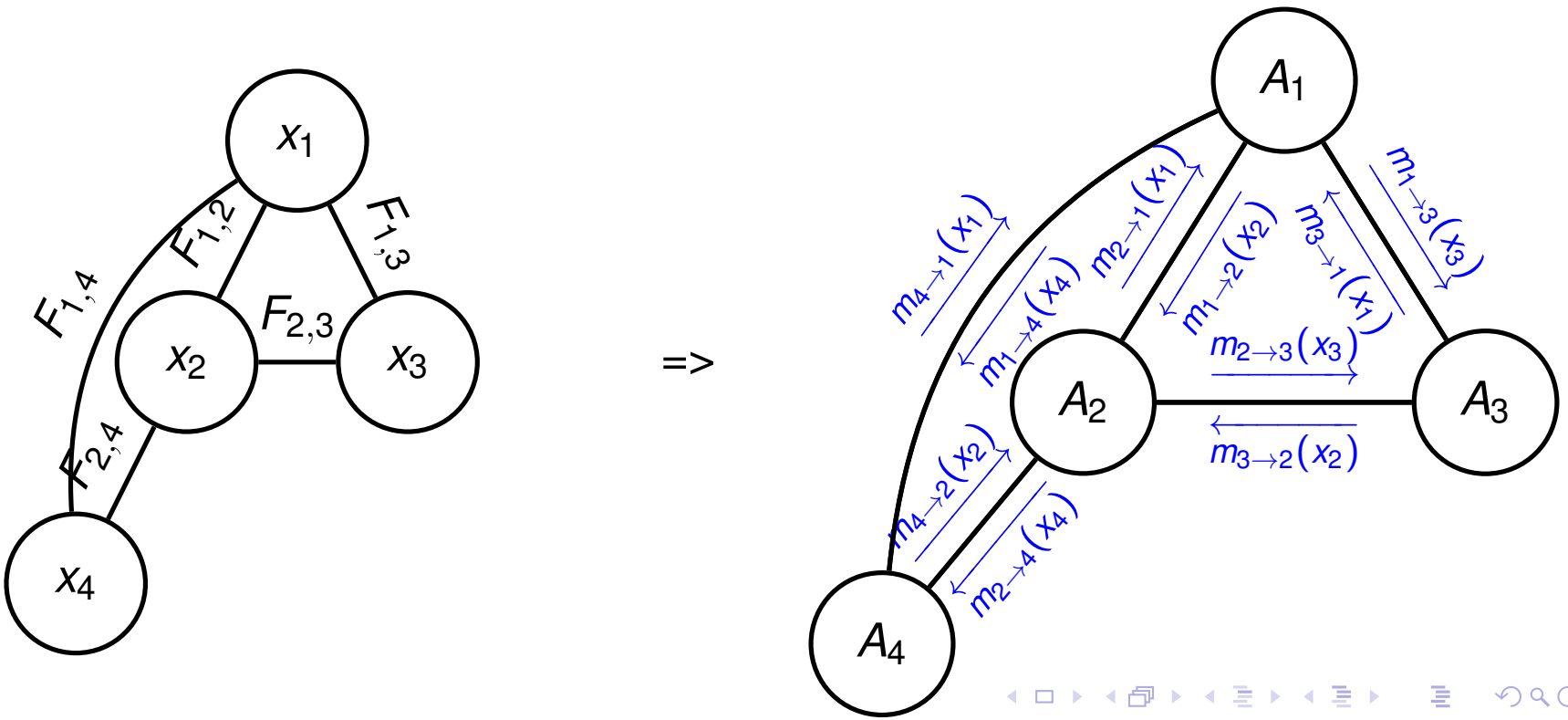

Anytime behaviours


Could result in very bad solutions

- local maxima arbitrarily far from optimal

The Max-Sum algorithm

Agents iteratively exchange messages to build a local function that depends only on the variables they control



Max-Sum messages

At each execution step, each agent A_i sends to each of its neighbors A_j the message:

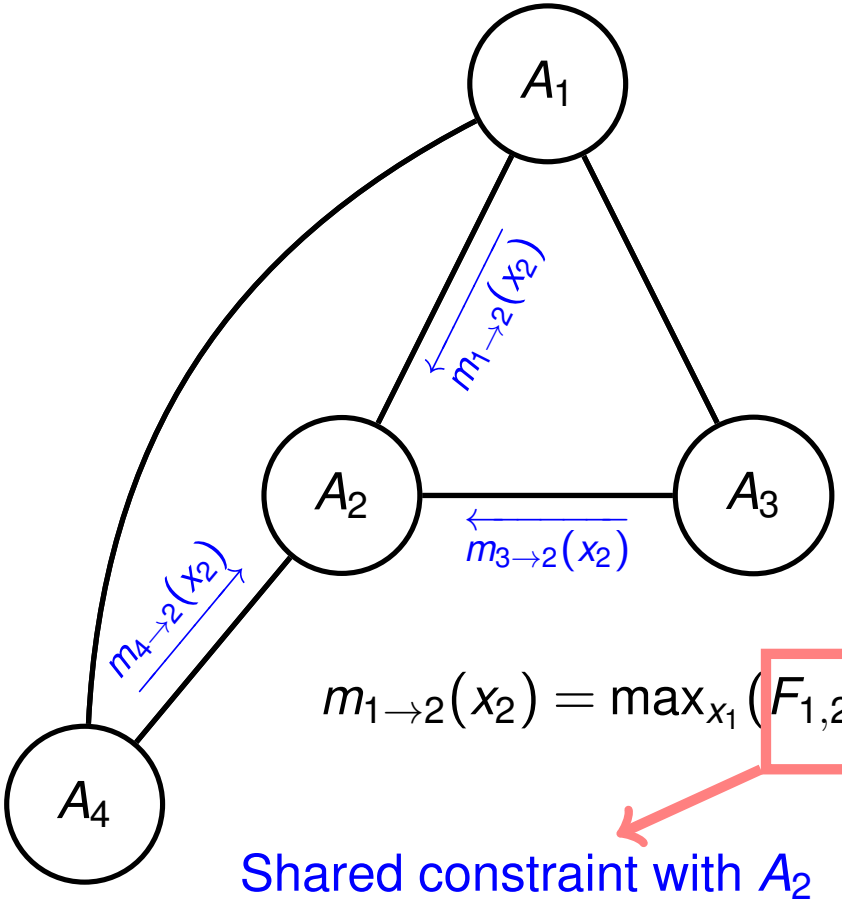
$$m_{i \to j}(x_j) = \alpha_{ij} + \max_{x_i} \left(F_{ij}(x_i, x_j) + \sum_{k \in N(i) \setminus j} m_{k \to i}(x_i) \right)$$

Shared constraint with A_j — All incoming messages except from A_j

where:

- α_{ij} is a normalization constant added to all components of the message so that $\sum_{x_j} m_{i \to j}(x_j) = 0$
- $N(i)$ is the set of indices for variables that are connected to x_i

Max-Sum by example



$$m_{1\to2}(x_2) = \max_{x_1}\left(F_{1,2}(x_1, x_2) + m_{3\to1}(x_1) + m_{4\to1}(x_1)\right)$$

Shared constraint with A_2

All incoming messages except from A_2

Max-Sum assignments

At each iteration, each agent A_i:

- computes its local function as:

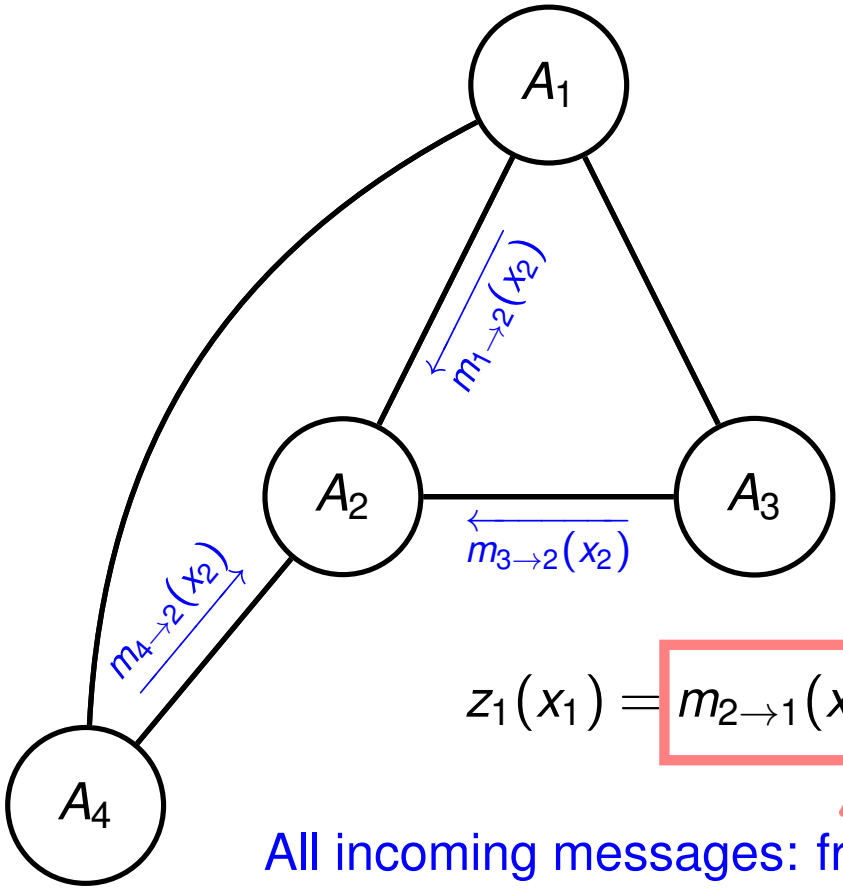
$$z_i(x_i) = \sum_{k \in N(i)} m_{k \to i}(x_i)$$

All incoming messages

- sets its assignment as the value that maximizes its local function:

$$\tilde{x}_i = arg\max_{x_i} z_i(x_i)$$

Max-Sum by example



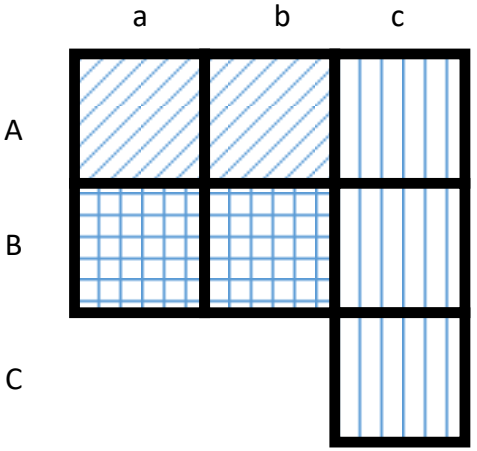
$$z_1(x_1) = m_{2\to 1}(x_1) + m_{3\to 1}(x_1) + m_{4\to 1}(x_1)$$

All incoming messages: from A_4, A_3 and A_1

Example run of Max-Sum

Consider this grid environment, divided in three areas (denoted by patterns)

Each area is controlled by an agent, which has to place a sensor in the area with the constraint that there cannot be two sensors on the same row (A, B, C) nor on the same column (a,b,c)



Agents are {1,2,3}, their variables are the positions of the sensors in their areas {P1, P2, P3}, their domains are {Aa, Ab}, {Ac, Bc, Cc}, and {Ba, Bb}, respectively, and the constraints are:

F_{12}	P_1	P_2
-∞	Aa	Ac
0	Aa	Bc
0	Aa	Cc
-∞	Ab	Ac
0	Ab	Bc
0	Ab	Cc

F_{13}	P_1	P_3
-∞	Aa	Ba
0	Aa	Bb
0	Ab	Ba
-∞	Ab	Bb

F_{23}	P_2	P_3
0	Ac	Ba
0	Ac	Bb
-∞	Bc	Ba
-∞	Bc	Bb
0	Cc	Ba
0	Cc	Bb

Example run of Max-Sum

Assume normalization constants α equal to 0

In the first execution round, agent 1 sends the following messages:

$m_{1\to2}(P_2) = \alpha_{12} + \max_{P1} F_{12}(P_1,P_2) =$

P_2	$m_{1\to2}$
Ac	-∞
Bc	0
Cc	0

$m_{1\to3}(P_3) = \alpha_{13} + \max_{P1} F_{13}(P_1,P_3) =$

P_3	$m_{1\to3}$
Ba	0
Bb	0

Similarly, agent 2 sends the following messages:

P_1	$m_{2\to1}$
Aa	0
Ab	0

P_3	$m_{2\to3}$
Ba	0
Bb	0

And agent 3 sends the following messages:

P_1	$m_{3\to1}$
Aa	0
Ab	0

P_2	$m_{3\to2}$
Ac	0
Bc	-∞
Cc	0

Example run of Max-Sum

Agent 1 computes $z_1(P_1) = m_{2\to1}(P_1) + m_{3\to1}(P_1) =$

P_1	z_1
Aa	0
Ab	0

and selects one value that maximizes $z_1(P_1)$, namely $\sim P_1$ = Aa (according to the lexicographic order)

Agent 2 does the same: computes $z_2(P_2) = m_{1\to2}(P_2) + m_{3\to2}(P_2) =$

P_2	z_2
Ac	-∞
Bc	-∞
Cc	0

and selects the value $\sim P_2$ = Cc

Agent 3 does the same: computes $z_3(P_3) = m_{1\to3}(P_3) + m_{2\to3}(P_3) =$

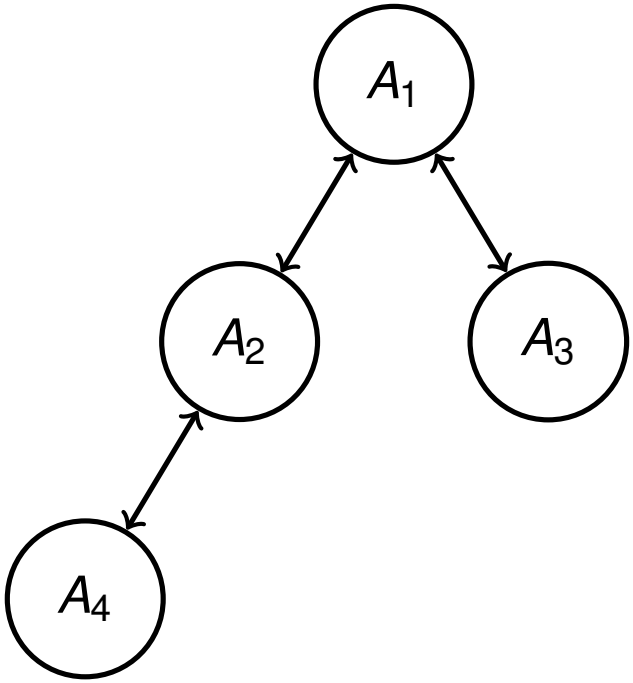
P_3	z_3
Ba	0
Bb	0

and selects the value $\sim P_3$ = Ba (according to the lexicographic order)

Then, Max-Sum continues with the next iteration…

Max-Sum on acyclic graphs

- Optimal on acyclic graphs
  - Different branches are independent
  - z functions provide correct estimations of agents contributions to the global problem
- Convergence guaranteed in a polynomial number of cycles



Max-Sum on cyclic graphs

On cyclic graphs, limited theoretical results:

- Lack of convergence guarantees
- When converges, it does converge to a neighborhood maximum
- Neighborhood maximum: guaranteed to be greater than all other maxima within a particular region of the search space

Quality guarantees

So far, algorithms presented (DSA-1, MGM-1, Max-Sum) do not provide any guarantee on the quality of their solutions

- Quality highly dependent on many factors which cannot always be properly assessed before deploying the system.
- Particularly adverse behaviour on specific pathological instances.

Challenge:

- Quality assessment on approximate algorithms

Quality guarantees for approx. techniques

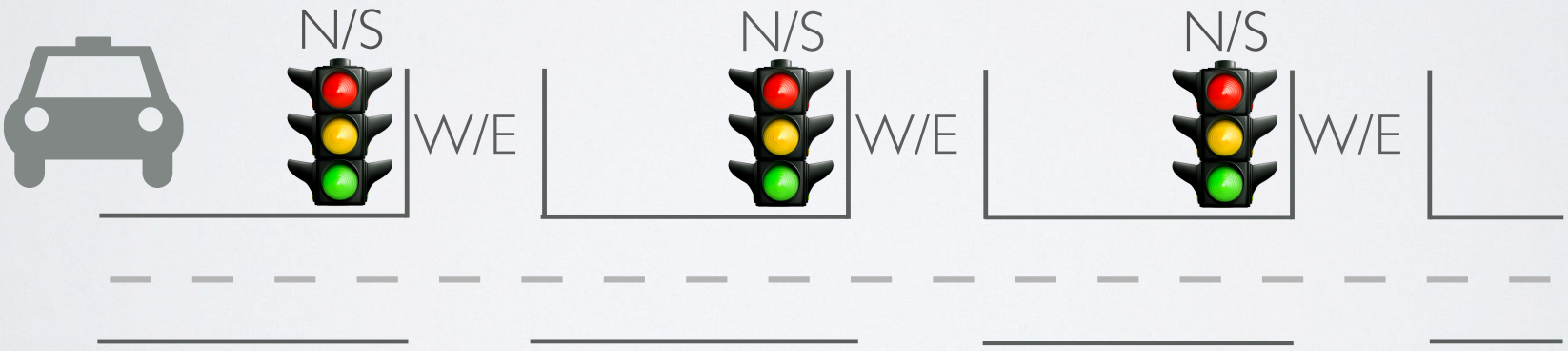
- Key area of research
- Address trade-off between guarantees and computational effort
- Particularly important for:
  - Dynamic settings
  - Severe constrained resources (e.g. embedded devices)
  - Safety critical applications (e.g. search and rescue)

DCOP APPLICATIONS

- Scheduling Problems
  - Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. AAMAS 2004
- Radio Frequency Allocation Problems
  - Improving DPOP with Branch Consistency for Solving Distributed Constraint Optimization Problems. CP 2014
- Sensor Networks
  - Preprocessing techniques for accelerating the DCOP algorithm ADOPT. AAMAS 2005
- Home Automation
  - A Multiagent System Approach to Scheduling Devices in Smart Homes. AAMAS 2017, IJCAI 2016
- Traffic Light Synchronization
  - Evaluating the performance of DCOP algorithms in a real world, dynamic problem. AAMAS 2008
- Disaster Evacuation
  - Disaster Evacuation Support. AAAI 2007; JAIR 2017
- Combinatorial Auction Winner Determination
  - H-DPOP: Using Hard Constraints for Search Space Pruning in DCOP. AAAI 2008

TRAFFIC FLOW CONTROL

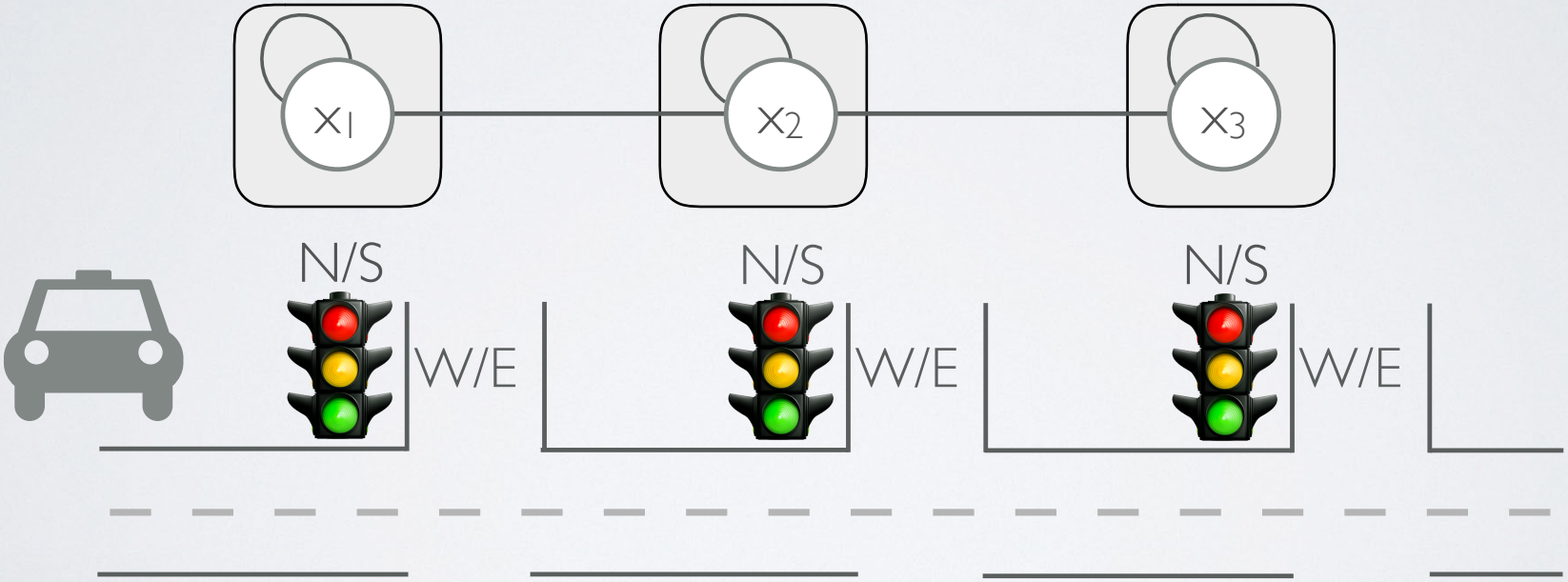
- Given a set of traffic lights in adjacent intersections
- How coordinate them to create green waves?



de Oliveira, D., Bazzan, A. L., & Lesser, V.. Using cooperative mediation to coordinate traffic lights: a case study. AAMAS 2005: 371-378
Junges, R., & Bazzan, A. L. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. AAMAS 2008: 463-470

TRAFFIC FLOW CONTROL

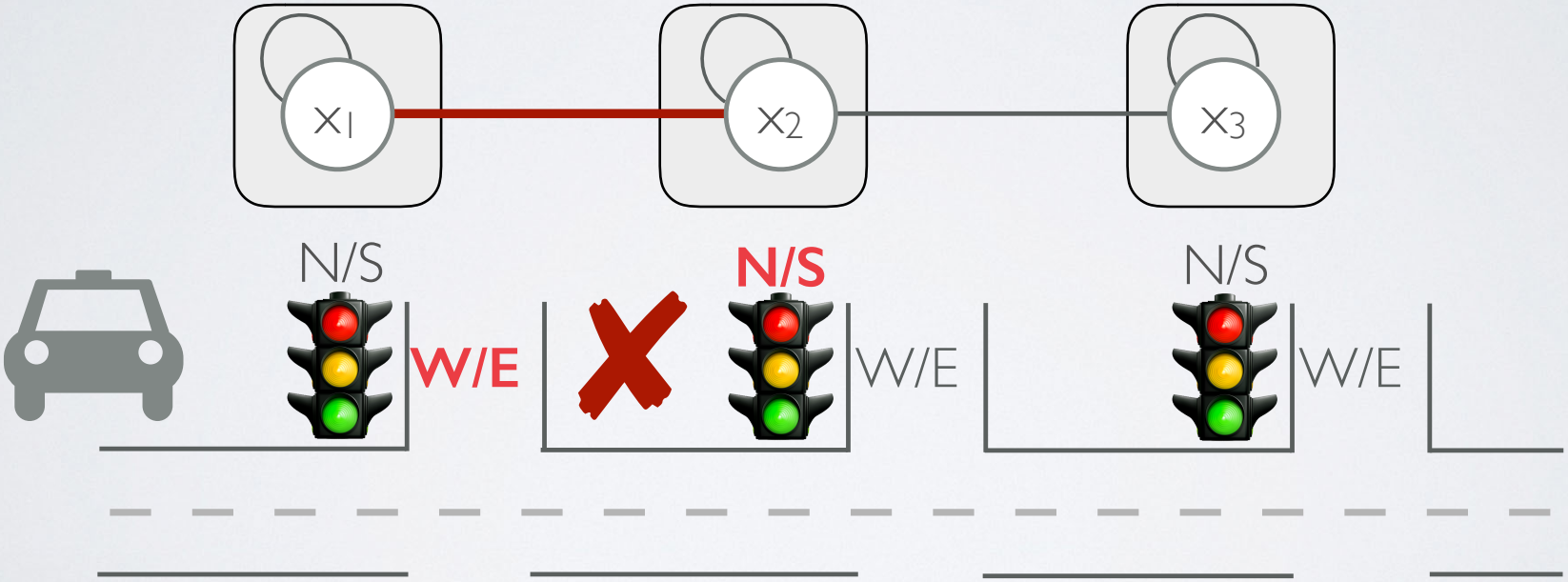
- Agents: Each traffic light
- Values: Flow traffic direction



de Oliveira, D., Bazzan, A. L., & Lesser, V.. Using cooperative mediation to coordinate traffic lights: a case study. AAMAS 2005: 371-378
Junges, R., & Bazzan, A. L. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. AAMAS 2008: 463-470

TRAFFIC FLOW CONTROL

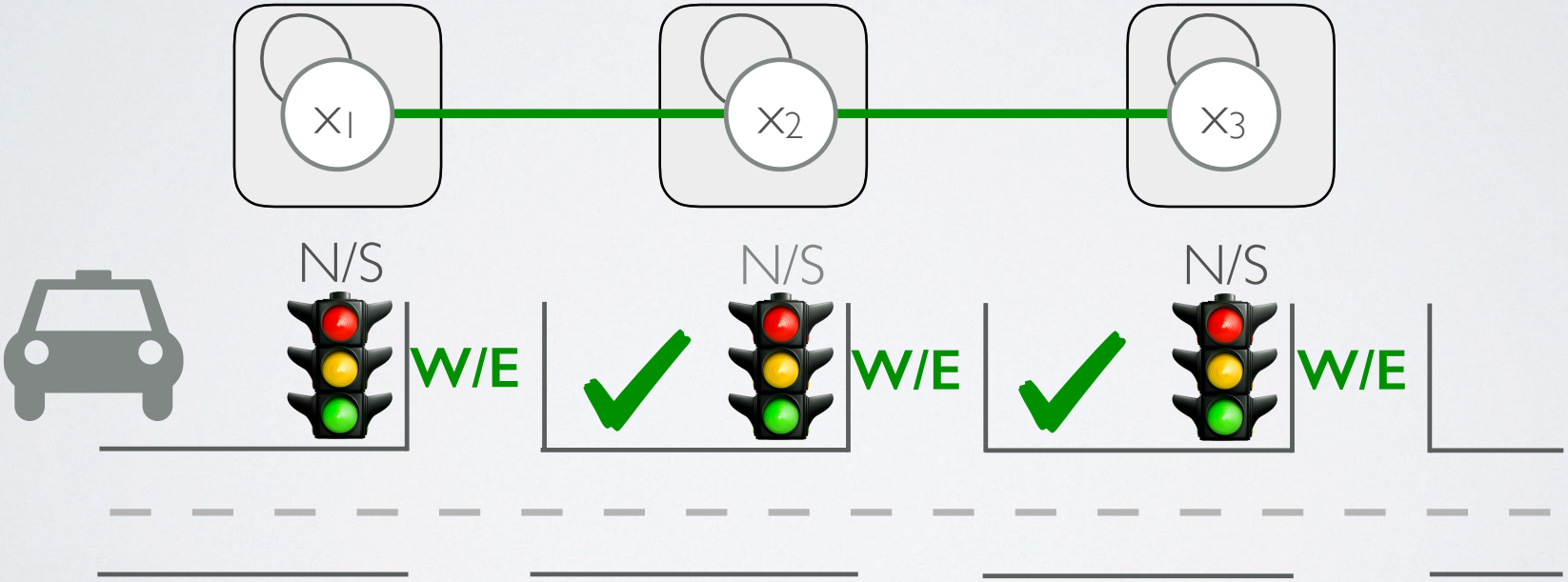
- Agents: Each traffic light
- Values: Flow traffic direction
- Conflict if 2 neighboring signals choose different directions



de Oliveira, D., Bazzan, A. L., & Lesser, V.. Using cooperative mediation to coordinate traffic lights: a case study. AAMAS 2005: 371-378
Junges, R., & Bazzan, A. L. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. AAMAS 2008: 463-470

TRAFFIC FLOW CONTROL

- Cost functions model the number of incoming vehicles
- Maximize the traffic flow

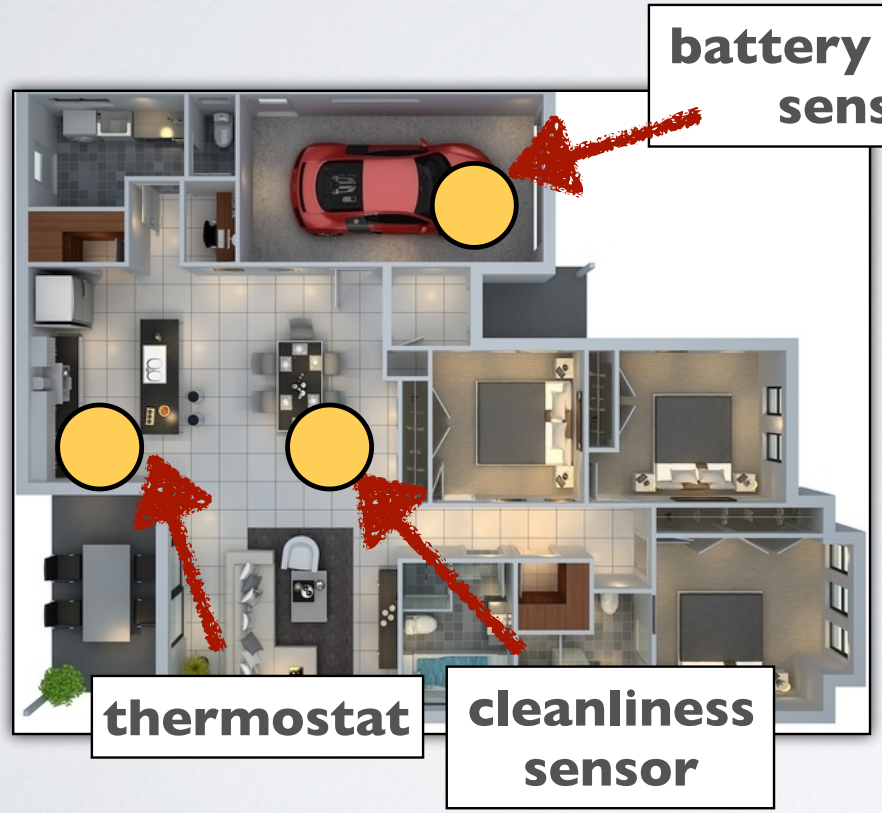


de Oliveira, D., Bazzan, A. L., & Lesser, V.. Using cooperative mediation to coordinate traffic lights: a case study. AAMAS 2005: 371-378
Junges, R., & Bazzan, A. L. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. AAMAS 2008: 463-470

SMART DEVICES

HOME ASSISTANTS

SMART HOMES DEVICE SCHEDULING



A **smart home** has:

- Smart devices (roomba, HVAC) that it can control
- Sensors (cleanliness, temperature)
- A set of locations

Ferdinando Fioretto, William Yeoh, Enrico Pontelli. "A Multiagent System Approach to Scheduling Devices in Smart Homes". AAMAS, 2017.

SMART HOMES DEVICE SCHEDULING

Smart device:

- A set of actions it can perform (clean, charge)
- Power consumption associated to each action.

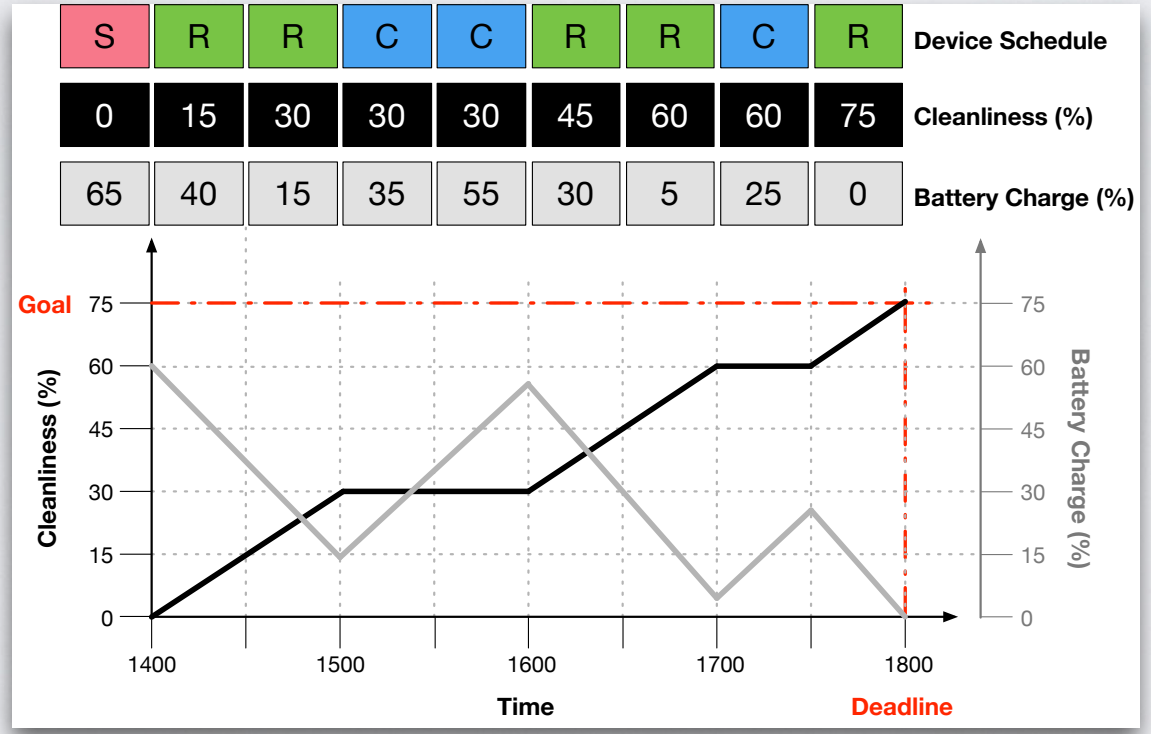
Scheduling Rules:

$$\texttt{living\_room cleanliness} \geq 75 \texttt{ before } 1800$$

Ferdinando Fioretto, William Yeoh, Enrico Pontelli. "A Multiagent System Approach to Scheduling Devices in Smart Homes". AAMAS, 2017.

SMART HOMES
DEVICE SCHEDULING

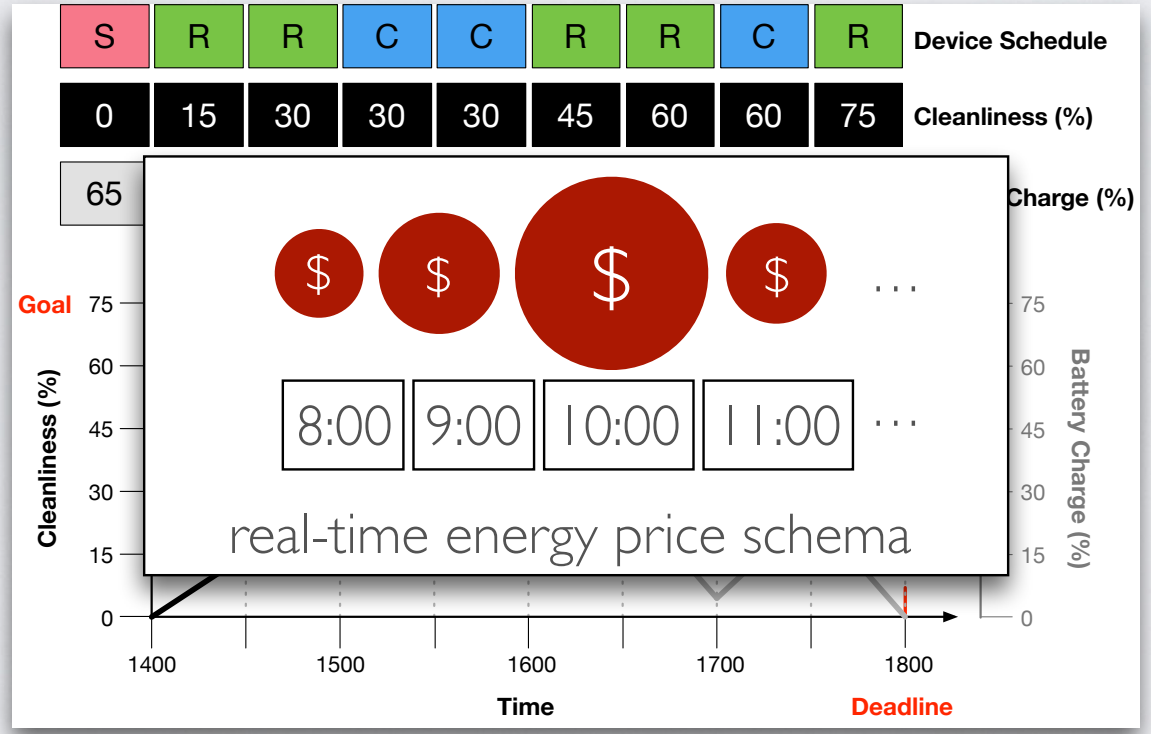
living_room cleanliness $\geq$ *75 before 1800*



Ferdinando Fioretto, William Yeoh, Enrico Pontelli. "A Multiagent System Approach to Scheduling Devices in Smart Homes". AAMAS, 2017.

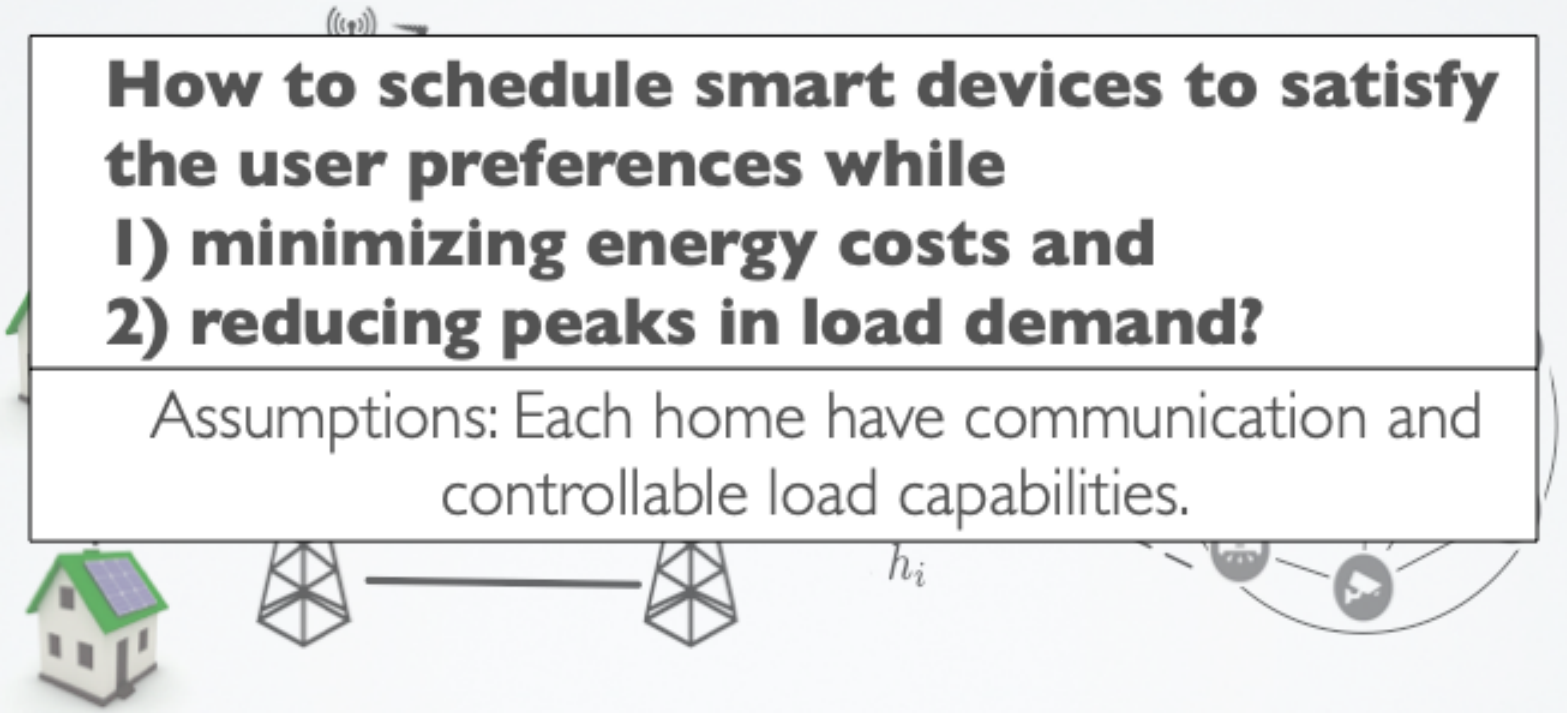
SMART HOMES DEVICE SCHEDULING

living_room cleanliness $\geq$ *75 before 1800*



Ferdinando Fioretto, William Yeoh, Enrico Pontelli. "A Multiagent System Approach to Scheduling Devices in Smart Homes". AAMAS, 2017.

SMART HOMES DEVICE SCHEDULING



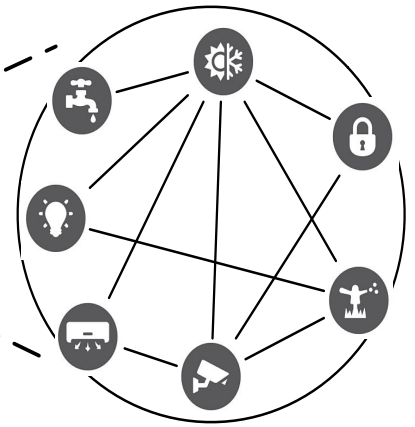
Ferdinando Fioretto, William Yeoh, Enrico Pontelli. "A Multiagent System Approach to Scheduling Devices in Smart Homes". AAMAS, 2017.

SMART HOMES
DEVICE SCHEDULING

Domains

Environmental variables

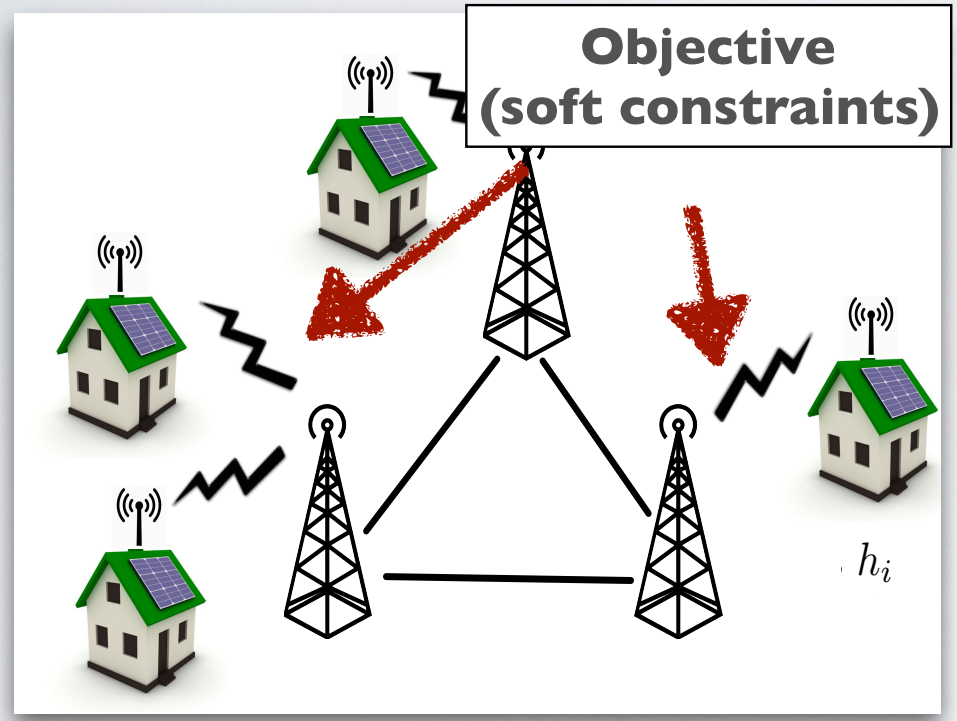
Action	State property	Power (kW/h)
run	cleanliness, battery charge	0.0
charge	battery charge	0.26
stop		0.0

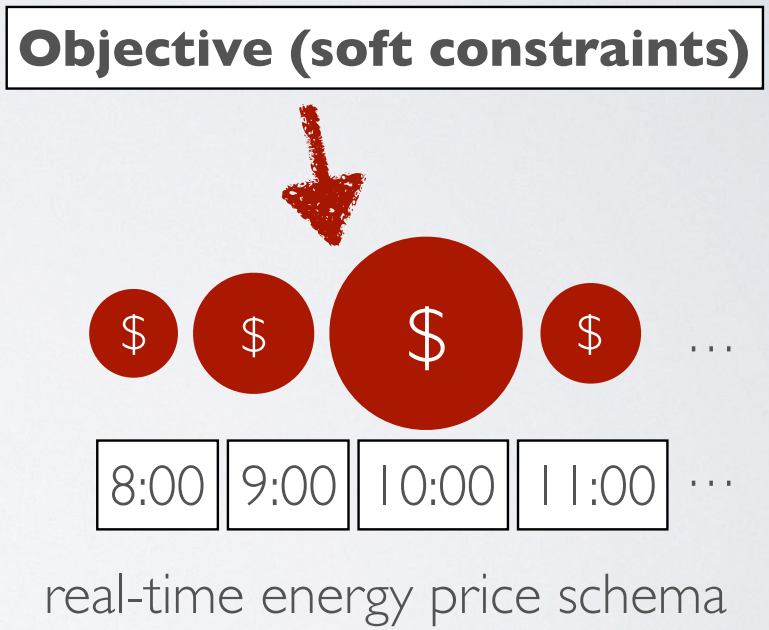


living_room cleanliness $\geq$ *75 before 1800*

Hard constraints

SMART HOMES
DEVICE SCHEDULING





real-time energy price schema

Ferdinando Fioretto, William Yeoh, Enrico Pontelli. "A Multiagent System Approach to Scheduling Devices in Smart Homes". AAMAS, 2017.