Distributed Constraint Optimization

Gauthier Picard

MINES Saint-Étienne
LaHC UMR CNRS 5516

gauthier.picard@emse.fr

— Some contents taken from OPTMAS 2011 and OPTMAS-DCR 2014 Tutorials—

Contents

Constraint Optimization Problems

Sometimes satisfaction is not possible

- Overconstrained problem
- Solution is not binary

Switch from satisfaction to optimization

- Minimizing the number of violated constraints
- Minimizing the cost of violated constraints
- Maximizing the overall utility of the system
- ...

DCOP Framework

Motivations

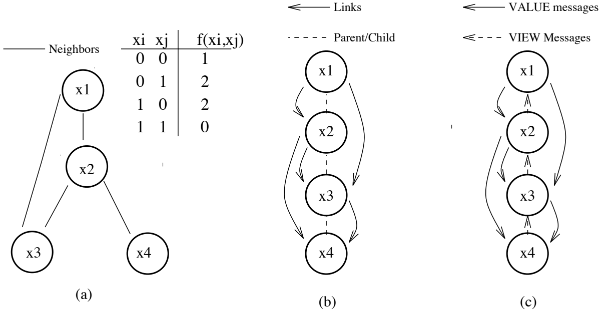
- In dynamic and complex environments not all constraints can be satisfied completely
- Satisfaction $\rightarrow$ **Optimisation** (combinatorial)
  - ex: minimizing the number of unchecked constraints, minimizing the sum of the costs of violated constraints, etc.

Definition (DCOP)

A *DCOP* is a DCSP $\langle A, X, D, C, \phi\rangle$ with

- a cost function $f_{ij} : D_i \times D_j \mapsto \mathbb{N} \cup \infty$ for each pair x_i, x_j
- an objective function $F : D \mapsto \mathbb{N} \cup \infty$ evaluating an assignment $\mathcal{A}$ with $f_{ij}(d_i, d_j)$ for each pair x_i, x_j

DCOP Framework (cont.)



Objective Function

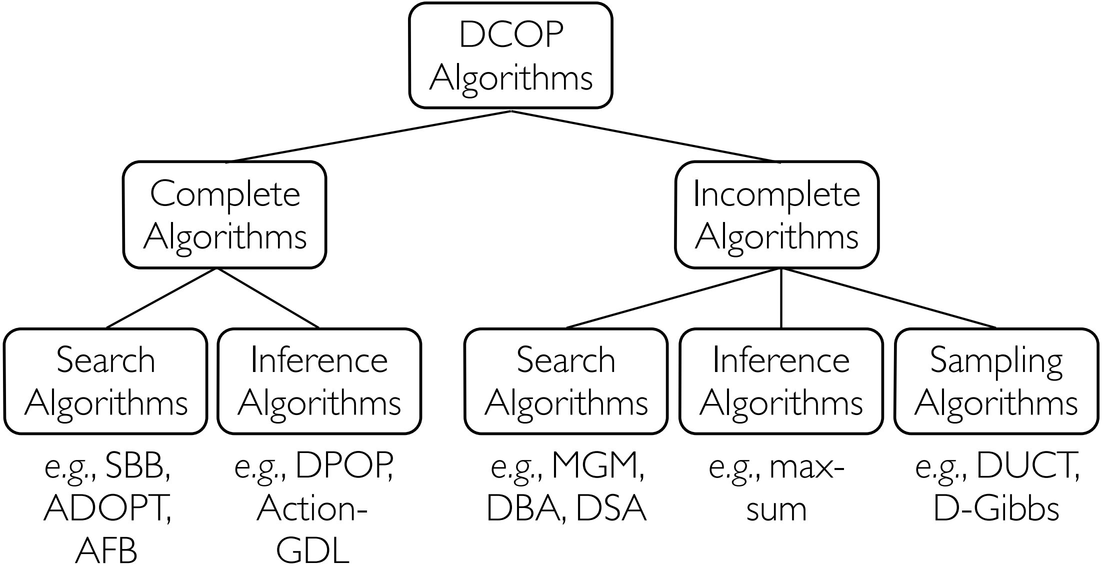
$$F(\mathcal{A}) = \sum_{x_i, x_j \in X} f_{ij}(d_i, d_j) \text{ where } x_i \leftarrow d_i \text{ and } x_i \leftarrow d_i \text{ in } \mathcal{A}$$

In figure *(a)*:

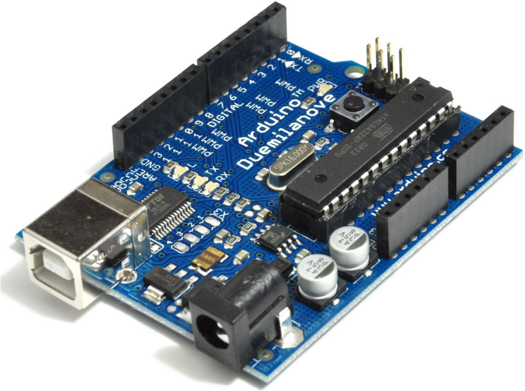
- $F(\{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0)\}) = 4$
- $F(\{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1)\}) = 0$

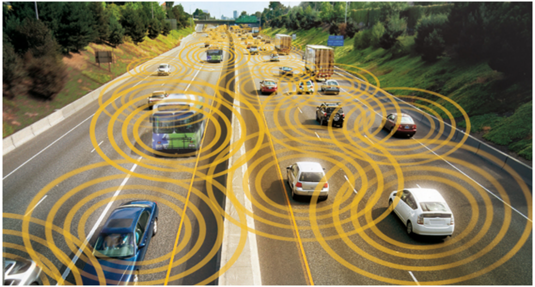
and
$\mathcal{A}^* = \{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1)\}$

DCOP Algorithms



Application Domains





Contents

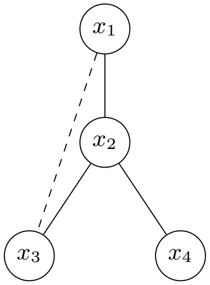
Asynchronous Distributed Optimisation (ADOPT) [Modi et al., 2005]

ADOPT: DFS tree (pseudotree)

ADOPT assumes that agents are arranged in a DFS tree:

- constraint graph $\rightarrow$ rooted graph (select a node as root)
- some links form a tree / others are backedges
- two constrained nodes must be in the same path to the root by tree links (same branch)

Every graph admits a DFS tree: DFS graph traversal



ADOPT Features

- Asynchronous algorithm
- Each time an agent receives a message:
  - Processes it (the agent may take a new value)
  - Sends VALUE messages to its children and pseudochildren
  - Sends a COST message to its parent
- Context: set of (variable value) pairs (as ABT agent view) of ancestor agents (in the same branch)
- Current context:
  - Updated by each VALUE message
  - If current context is not compatible with some child context, the later is initialized (also the child bounds)

ADOPT Procedures

Initialize
$threshold \leftarrow 0$; $CurrentContext \leftarrow \{\}$;
forall $d \in D_i, x_l \in Children$ **do**
  $lb(d, x_l) \leftarrow 0$; $t(d, x_l) \leftarrow 0$;
  $ub(d, x_l) \leftarrow Inf$; $context(d, x_l) \leftarrow \{\}$; **enddo**;
$d_i \leftarrow d$ that minimizes $LB(d)$;
backTrack;

when received (**THRESHOLD**, t, $context$)
if $context$ compatible with $CurrentContext$:
  $threshold \leftarrow t$;
  maintainThresholdInvariant;
  backTrack; **endif**;

when received (**TERMINATE**, $context$)
record TERMINATE received from parent;
$CurrentContext \leftarrow context$;
backTrack;

when received (**VALUE**, (x_j, d_j))
if TERMINATE not received from parent:
  add (x_j, d_j) to $CurrentContext$;
  forall $d \in D_i, x_l \in Children$ **do**
    if $context(d, x_l)$ incompatible with $CurrentContext$:
      $lb(d, x_l) \leftarrow 0$; $t(d, x_l) \leftarrow 0$;
      $ub(d, x_l) \leftarrow Inf$; $context(d, x_l) \leftarrow \{\}$; **endif**; **enddo**;
  maintainThresholdInvariant;
  backTrack; **endif**;

when received (**COST**, x_k, $context$, lb, ub)
$d \leftarrow$ value of x_i in $context$;
remove (x_i, d) from $context$;
if TERMINATE not received from parent:
  forall $(x_j, d_j) \in context$ and x_j is not my neighbor **do**
    add (x_j, d_j) to $CurrentContext$;**enddo**;
  forall $d' \in D_i, x_l \in Children$ **do**
    if $context(d', x_l)$ incompatible with $CurrentContext$:
      $lb(d', x_l) \leftarrow 0$; $t(d', x_l) \leftarrow 0$;
      $ub(d', x_l) \leftarrow Inf$; $context(d', x_l) \leftarrow \{\}$;**endif**;**enddo**;**endif**;
if $context$ compatible with $CurrentContext$:
  $lb(d, x_k) \leftarrow lb$;
  $ub(d, x_k) \leftarrow ub$;
  $context(d, x_k) \leftarrow context$;
  maintainChildThresholdInvariant;
  maintainThresholdInvariant; **endif**;
backTrack;

procedure backTrack
if $threshold == UB$:
  $d_i \leftarrow d$ that minimizes $UB(d)$;
else if $LB(d_i) > threshold$:
  $d_i \leftarrow d$ that minimizes $LB(d)$;**endif**;
SEND (**VALUE**, (x_i, d_i))
    to each lower priority neighbor;
maintainAllocationInvariant;
if $threshold == UB$:
  if TERMINATE received from parent
  or x_i is root:
    SEND (**TERMINATE**,
    $CurrentContext \cup \{(x_i, d_i)\}$)
      to each child;
    Terminate execution; **endif**;**endif**;
SEND (**COST**, x_i, $CurrentContext$, LB, UB)
  to parent;

Algorithm 1: ADOPT Procedures

ADOPT Messages

- value$(parent \to children \cup pseudochildren, a)$: parent informs descendants that it has taken value a
- cost$(child \to parent, lowerbound, upperbound, context)$: child informs parent of the best cost of its assignement; attached context to detect obsolescence
- threshold$(parent \to child, t)$: minimum cost of solution in child is at least t
- termination$(parent \to children)$: sent when $LB = UB$

ADOPT Data Structures

1. **Current context** (agent view): values of higher priority constrained agents

x_i	x_j	$\dots$
a	c	$\dots$

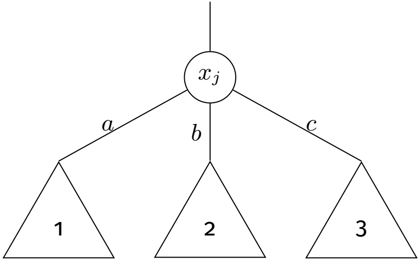
2. **Bounds** (for each value, child)

   - lower bounds
   - upper bounds
   - thresholds
   - contexts

x_j	a	b	c	d
$lb(x_k)$	3	0	0	0
$ub(x_k)$	∞	∞	∞	∞
$th(x_k)$	1	0	0	0
$context(x_k)$				

- Stored contextes must be active: $context \in currentcontext$
- If a context becomes no active, it is removed ($lb \leftarrow 0, th \leftarrow 0, ub \leftarrow \infty$)

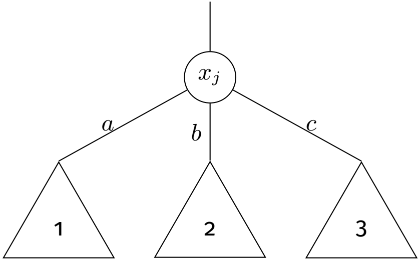
ADOPT Bounds



ADOPT Bounds

$\boxed{\delta(value) = \text{cost with higher agents}}$

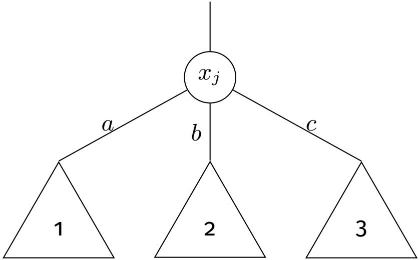
$$\delta(b) = \sum_{i \in curctx} c_{ij}(a, b)$$



ADOPT Bounds

$\boxed{\delta(value) = \text{cost with higher agents}}$

$$\delta(b) = \sum_{i \in curctx} c_{ij}(a, b)$$

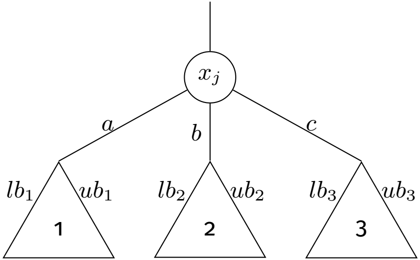


$$OPT(x_j, ctx) = \min_{d \in d_j} \delta(d) + \sum_{x_k \in children} OPT(x_k, ctx \cup (x_j, d))$$

ADOPT Bounds

$$\delta(value) = \text{cost with higher agents}$$

$$\delta(b) = \sum_{i \in curctx} c_{ij}(a, b)$$

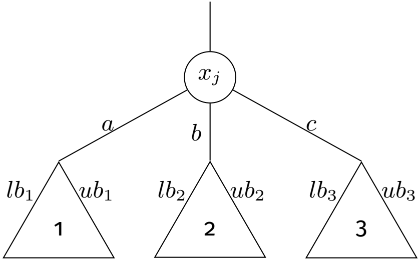


$$OPT(x_j, ctx) = \min_{d \in d_j} \delta(d) + \sum_{x_k \in children} OPT(x_k, ctx \cup (x_j, d))$$

ADOPT Bounds

$\delta(value)$ = cost with higher agents

$$\delta(b) = \sum_{i \in curctx} c_{ij}(a, b)$$



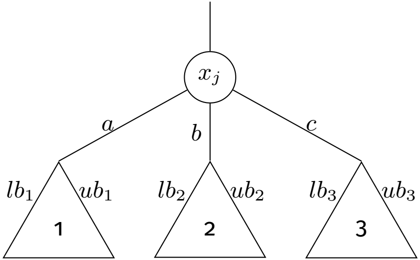
$[lb_k, ub_k]$ = cost of lower agents

$$OPT(x_j, ctx) = \min_{d \in d_j} \delta(d) + \sum_{x_k \in children} OPT(x_k, ctx \cup (x_j, d))$$

ADOPT Bounds

$\delta(value)$ = cost with higher agents

$$\delta(b) = \sum_{i \in curctx} c_{ij}(a,b)$$



$[lb_k, ub_k]$ = cost of lower agents

$$OPT(x_j, ctx) = \min_{d \in d_j} \delta(d) + \sum_{x_k \in children} OPT(x_k, ctx \cup (x_j, d))$$

$$LB(b) = \delta(b) + \sum_{x_k \in children} lb(b, x_k)$$

$$LB = \min_{b \in d_j} LB(b)$$

$$UB(b) = \delta(b) + \sum_{x_k \in children} ub(b, x_k)$$

$$UB = \min_{b \in d_j} UB(b)$$

ADOPT Value Assignment

- An ADOPT agent takes the value with minimum LB
- Eager behavior:
  - Agents may constantly change value
  - Generates many context changes
- Threshold:
  - lower bound of the cost that children have from previous search
  - parent distributes threshold among children
  - incorrect distribution does not cause problems: the child with minor allocation would send a COST to the parent later, and the parent will rebalance the threshold distribution

ADOPT Properties

- For any x_i, $LB \leq OPT(x_l, ctx) \leq UB$
- For any x_i, its threshold reaches UB
- For any x_i, its final threshold is equal to $OPT(x_l, ctx)$

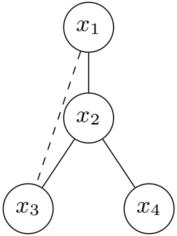
$\rightarrow$ ADOPT terminates with the optimal solution

ADOPT Example

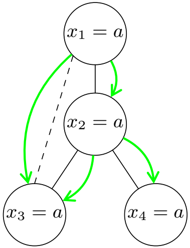
- 4 variables (4 agents) x_1, x_2, x_3 and x_4 with $D = \{a, b\}$
- 4 binary identical cost functions

x_i	x_j	cost
a	a	1
a	b	2
b	a	2
b	b	0

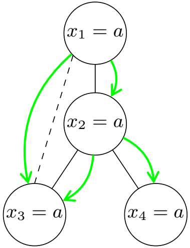
- Constraint graph

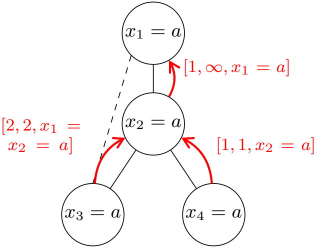


ADOPT Example (cont.)

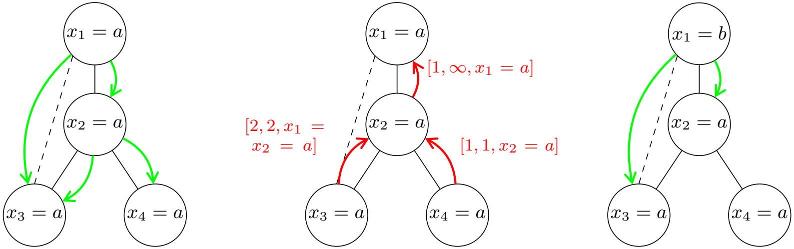


ADOPT Example (cont.)

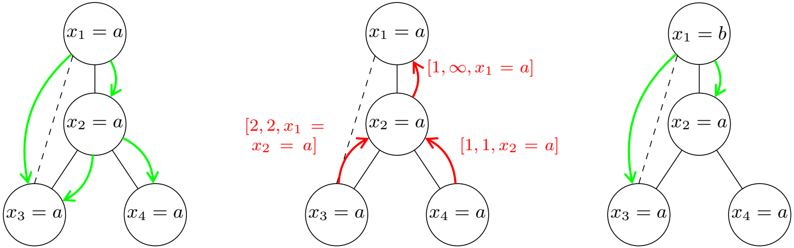


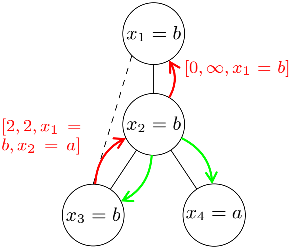


ADOPT Example (cont.)

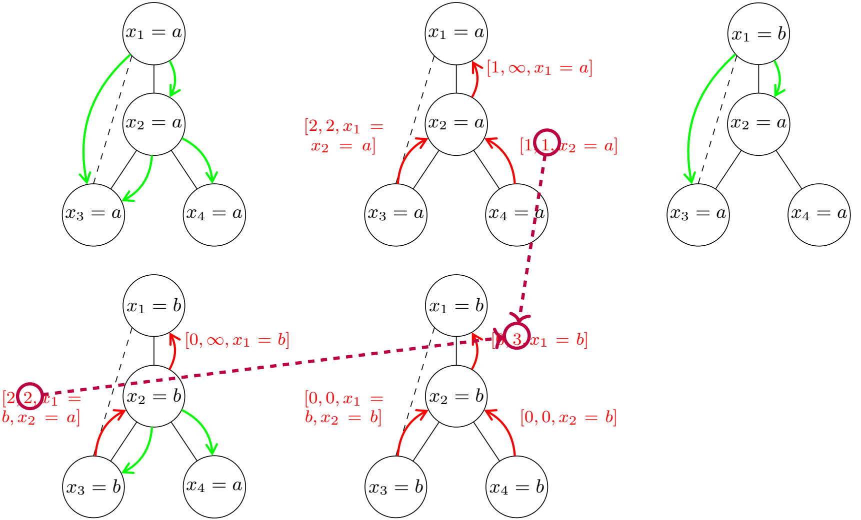


ADOPT Example (cont.)

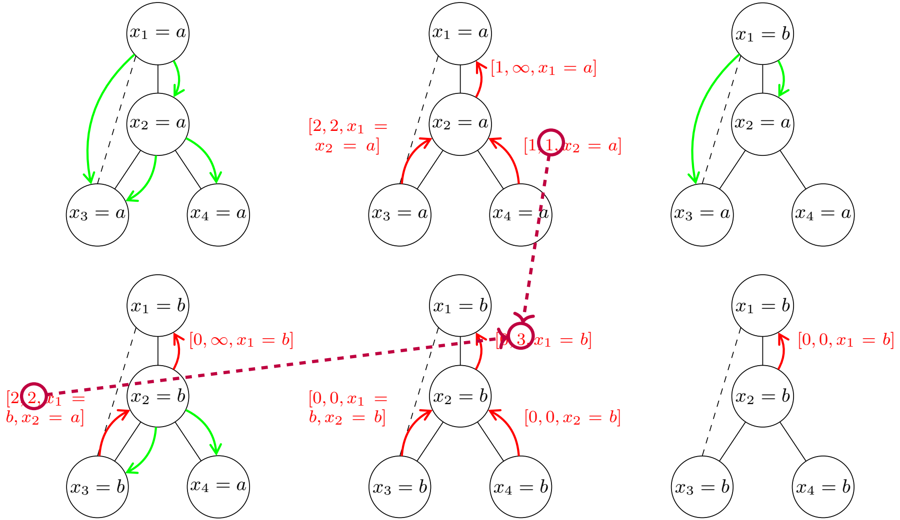




ADOPT Example (cont.)



ADOPT Example (cont.)



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

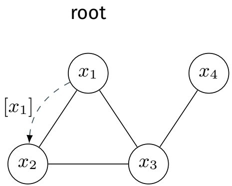
3-phase distributed algorithm

	PHASES	MESSAGES
1.	DFS Tree construction	token passing
2.	Utility phase: from leaves to root	**util** (child $\rightarrow$ parent, constraint table [-child])
3.	Value phase: from root to leaves	**value** (parent $\rightarrow$ children $\cup$ pseudochildren, parent value)

DFS Tree Phase

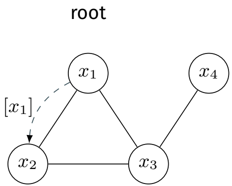
- **Distributed DFS graph traversal**: token, ID, $neighbors(X)$
  1. X owns the token: adds its own ID and sends it in turn to each of its neighbors, which become children
  2. Y receives the token from X: it marks X as visited. First time Y receives the token then $parent(Y) = X$. Other IDs in token which are also $neighbors(Y)$ are **pseudoparent**. If Y receives token from neighbor W to which it was never sent, W is pseudochild.
  3. When all $neighbors(X)$ visited, X removes its ID from token and sends it to $parent(X)$.
- A node is selected as root, which starts
- When all neighbors of root are visited, the DFS traversal ends

DFS Tree Phase: Example

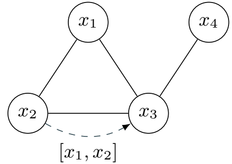


x_1 parent of x_2

DFS Tree Phase: Example



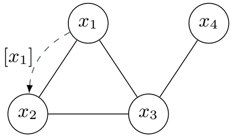
x_1 parent of x_2



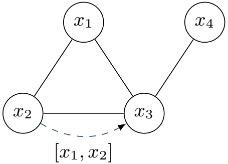
x_2 parent of x_3
x_1 pseudoparent of x_3

DFS Tree Phase: Example

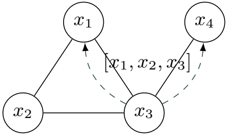
root



x_1 parent of x_2

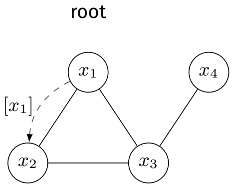


x_2 parent of x_3
x_1 pseudoparent of x_3

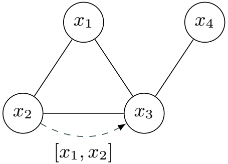


x_3 parent of x_4
x_3 pseudoparent of x_1

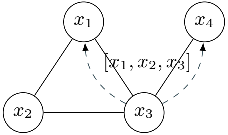
DFS Tree Phase: Example



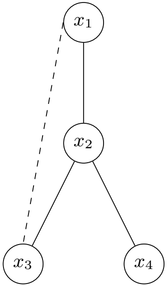
x_1 parent of x_2



x_2 parent of x_3
x_1 pseudoparent of x_3



x_3 parent of x_4
x_3 pseudoparent of x_1



Util Phase

Agent X:

- receives from each child Y_i a cost function: $C(Y_i)$
- combines (adds, joins) all these cost functions with the cost functions with $parent(X)$ and $pseudoparents(X)$
- projects X out of the resulting cost function, and sends it to $parent(X)$

From the leaves to the root

Util Phase: Example

Util Phase: Example

X	T	
a	a	1
a	b	2
b	a	2
b	b	0

parent

X

children

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Util Phase: Example

X	T	
a	a	1
a	b	2
b	a	2
b	b	0

parent

X

children

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

add

X	Y	Z	T	
a	a	a	a	3
a	a	a	b	4
a	a	b	a	4
a	a	b	b	5
a	b	a	a	4
a	b	a	b	5
a	b	b	a	5
a	b	b	b	6
b	a	a	a	6
b	a	a	b	4
b	a	b	a	4
b	a	b	b	2
b	b	a	a	4
b	b	a	b	2
b	b	b	a	2
b	b	b	b	0

All value combinations
Costs are the sum of applicable costs

Util Phase: Example

X	T	
a	a	1
a	b	2
b	a	2
b	b	0

parent

X

children

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

add

X	Y	Z	T	
a	a	a	a	3
a	a	a	b	4
a	a	b	a	4
a	a	b	b	5
a	b	a	a	4
a	b	a	b	5
a	b	b	a	5
a	b	b	b	6
b	a	a	a	6
b	a	a	b	4
b	a	b	a	4
b	a	b	b	2
b	b	a	a	4
b	b	a	b	2
b	b	b	a	2
b	b	b	b	0

All value combinations
Costs are the sum of applicable costs

Project out X

~~X~~	Y	Z	T	
~~a~~	a	a	a	3
~~a~~	~~a~~	~~a~~	~~b~~	~~4~~
~~a~~	a	b	a	4
~~a~~	~~a~~	~~b~~	~~b~~	~~5~~
~~a~~	b	a	a	4
~~a~~	~~b~~	~~a~~	~~b~~	~~5~~
~~a~~	~~b~~	~~b~~	~~a~~	~~5~~
~~a~~	~~b~~	~~b~~	~~b~~	~~6~~
~~b~~	~~a~~	~~a~~	~~a~~	~~6~~
~~b~~	a	a	b	4
~~b~~	~~a~~	~~b~~	~~a~~	~~4~~
~~b~~	a	b	b	2
~~b~~	~~b~~	~~a~~	~~a~~	~~4~~
~~b~~	b	a	b	2
~~b~~	b	b	a	2
~~b~~	b	b	b	0

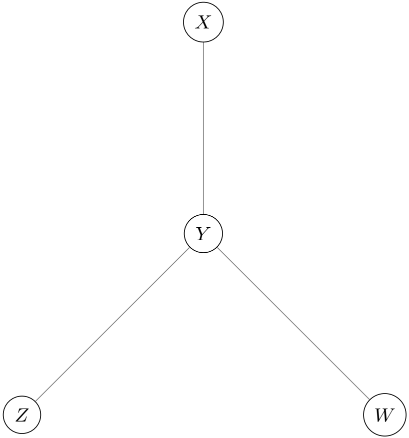
Remove X
Remove duplicates
Keep the min cost

Value Phase

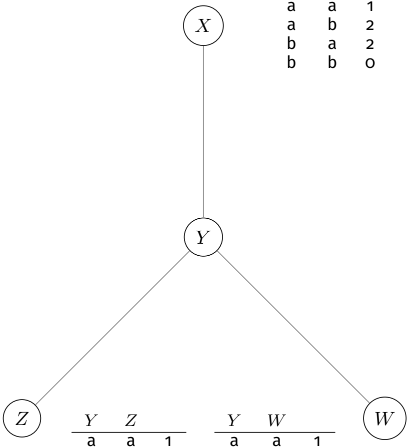
1. The root finds the **value that minimizes the received cost function** in the util phase, and informs its descendants (children $\cup$ pseudochildren)
2. Each agent **waits to receive** the value of its parent / pseudoparents
3. Keeping fixed the value of parent/pseudoparents, finds **the value that minimizes the received cost function** in the Util phase
4. Informs of this value to its children/pseudochildren

This process starts at the root and ends at the leaves

DTREE : DPOP for DCOPs without backedges



DTREE : DPOP for DCOPs without backedges

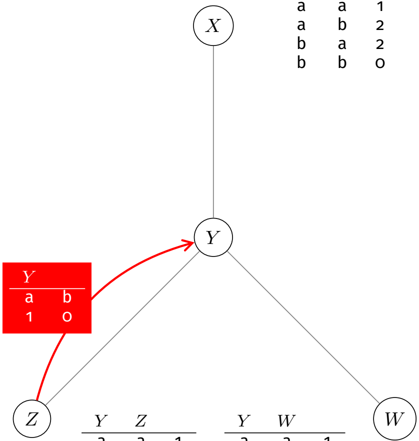


X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DTREE : DPOP for DCOPs without backedges



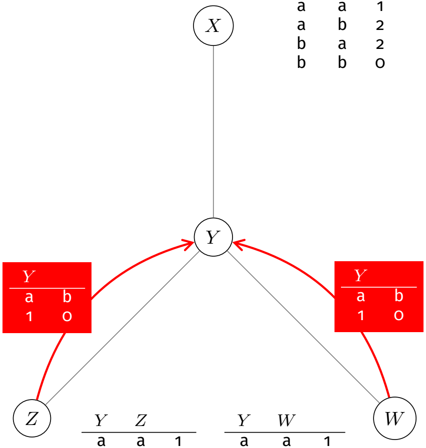
X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

Y	
a	b
1	0

Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DTREE : DPOP for DCOPs without backedges



X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

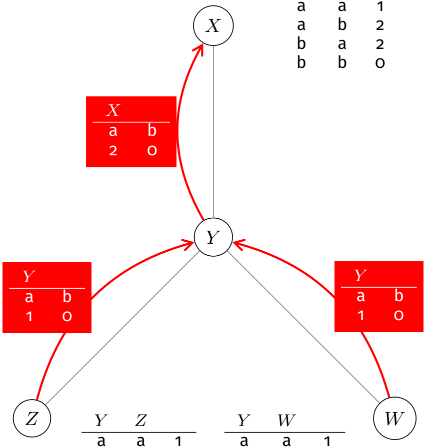
Y	
a	b
1	0

Y	
a	b
1	0

Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DTREE : DPOP for DCOPs without backedges



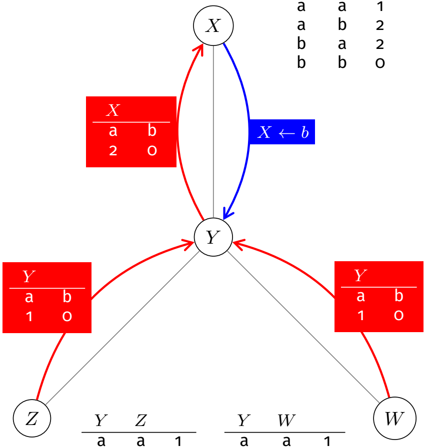
X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DTREE : DPOP for DCOPs without backedges

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

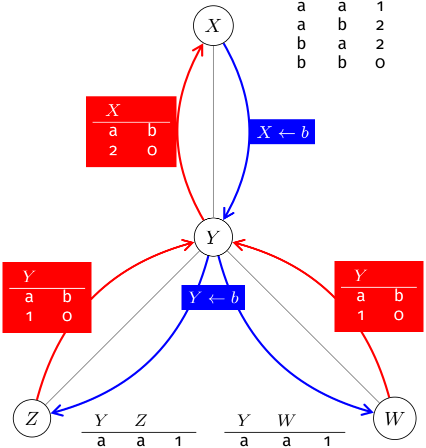


Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DTREE : DPOP for DCOPs without backedges

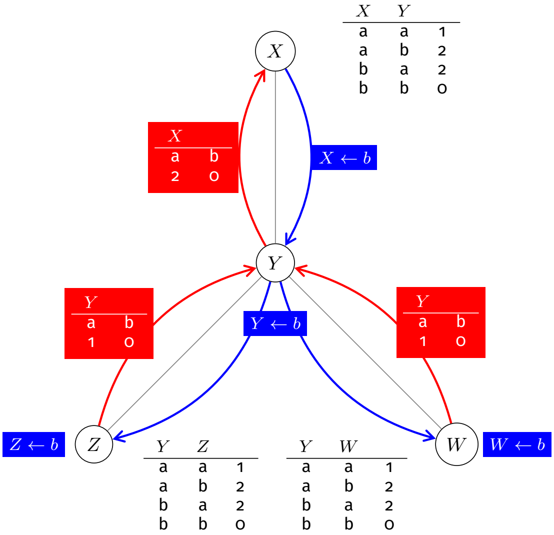
X	Y	
a	a	1
a	b	2
b	a	2
b	b	0



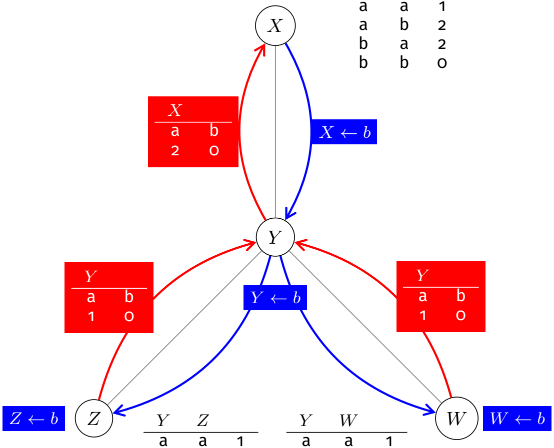
Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DTREE : DPOP for DCOPs without backedges



DTREE : DPOP for DCOPs without backedges



X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

X	
a	b
2	0

Y	
a	b
1	0

Y	
a	b
1	0

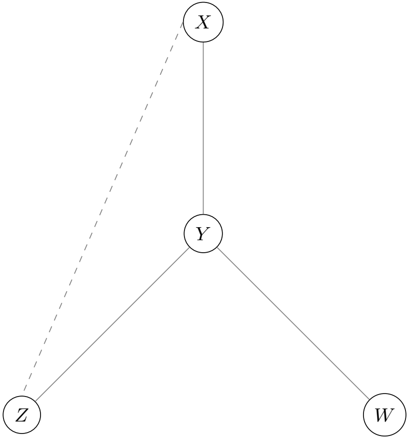
Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

Optimal solution:

- linear number of messages
- message size: linear

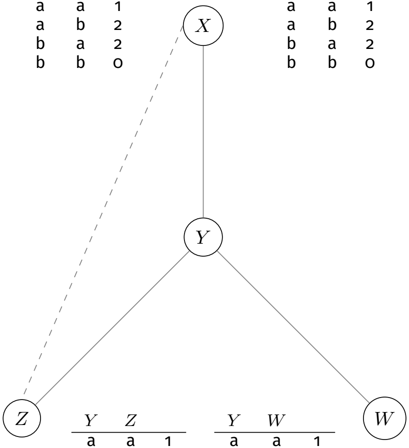
DPOP for any DCOP



DPOP for any DCOP

X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

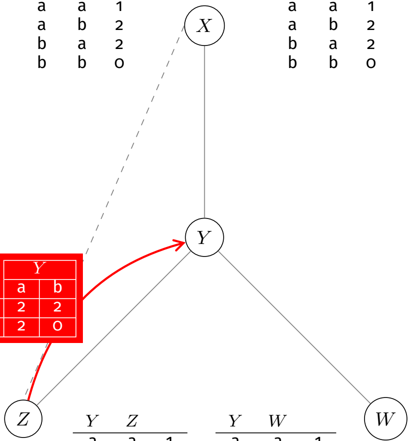
X	Y	
a	a	1
a	b	2
b	a	2
b	b	0



Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DPOP for any DCOP



X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

		Y	
		a	b
X	a	2	2
	b	2	0

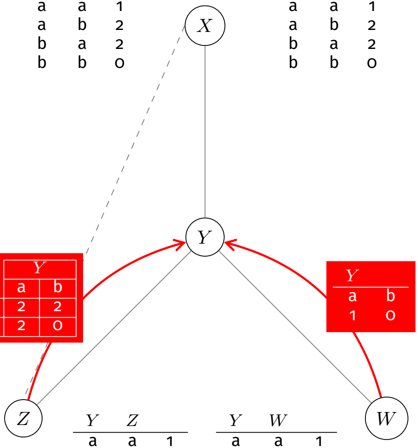
Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DPOP for any DCOP

X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0



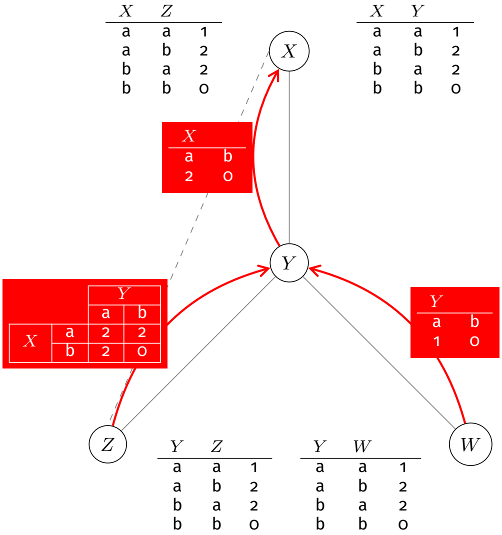
		Y	
		a	b
X	a	2	2
	b	2	0

Y	
a	b
1	0

Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DPOP for any DCOP



X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

X	a	b
	2	0

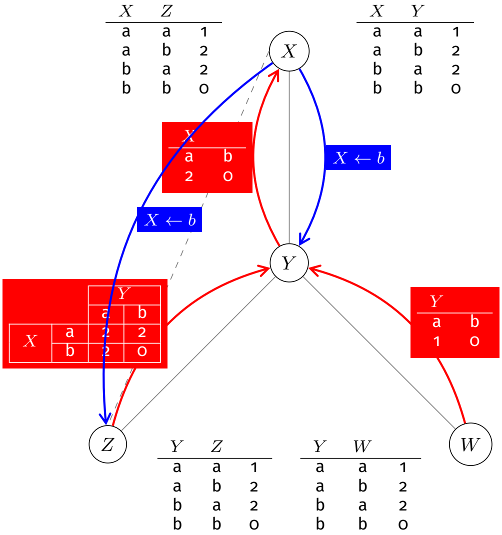
		Y	
		a	b
X	a	2	2
	b	2	0

Y	a	b
	1	0

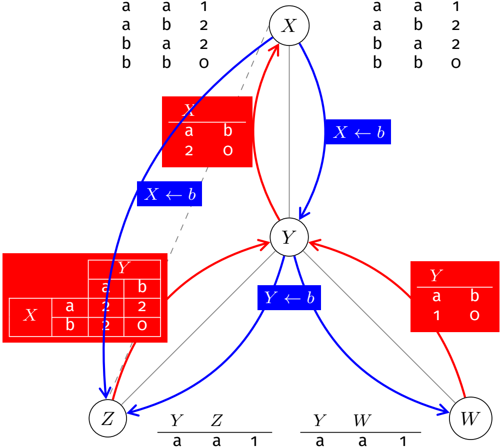
Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DPOP for any DCOP



DPOP for any DCOP



X	Z	
a	a	1
a	b	2
b	a	2
b	b	0

X	Y	
a	a	1
a	b	2
b	a	2
b	b	0

X	
a	b
2	0

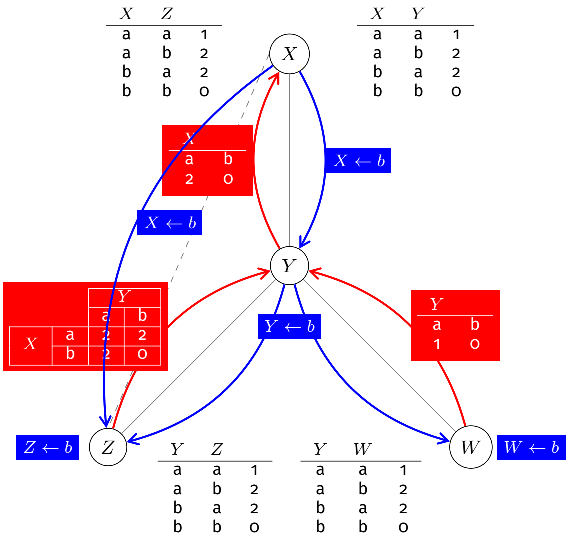
		Y	
		a	b
X	a	2	2
	b	2	0

Y	
a	b
1	0

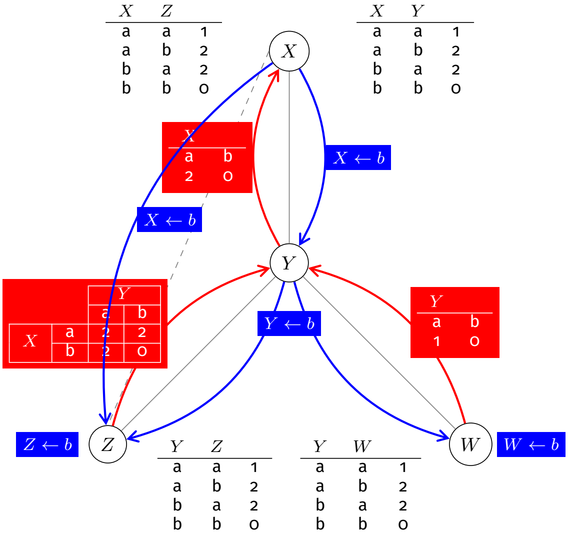
Y	Z	
a	a	1
a	b	2
b	a	2
b	b	0

Y	W	
a	a	1
a	b	2
b	a	2
b	b	0

DPOP for any DCOP



DPOP for any DCOP



Optimal solution:

- linear number of messages
- message size: exponential

Contents

Approximate Algorithms for DCOPs

Complete algorithms

- e.g. ADOPT [Modi et al., 2005] and DPOP [Petcu and Faltings, 2005]
  - ✓ complete
  - ✗ slow

Aproximate algorithms exist (fast, but sub-optimal in many case)

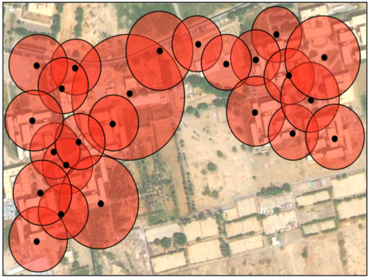
- Search algorithms
  - DBA [Yokoo, 2001], DSA [Zhang et al., 2005], MGM [Maheswaran et al., 2004]
- Inference algorithms
  - Max-sum [Farinelli et al., 2008]

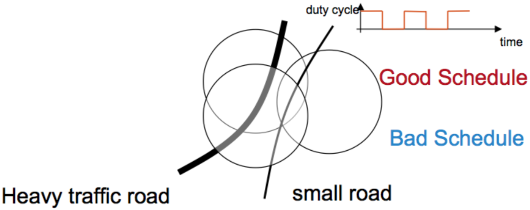
Why Approximate Algorithms

- Motivations
  - Often optimality in practical applications is not achievable
  - Fast good enough solutions are all we can have
- Example – Graph coloring
  - Medium size problem (about 20 nodes, three colors per node)
  - Number of states to visit for optimal solution in the worst case $3^{20} = 3M$ states
- Key problem
  - Provides guarantees on solution quality

Exemplar Application: Surveillance

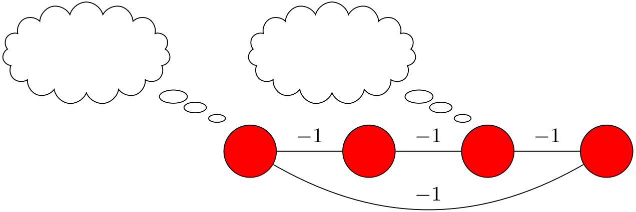
- Event Detection
  - Vehicles passing on a road
- Energy Constraints
  - Sense/Sleep modes
  - Recharge when sleeping
- Coordination
  - Activity can be detected by single sensor
  - Roads have different traffic loads
- Aim
  - Focus on road with more traffic load





Centralized Local Greedy approaches

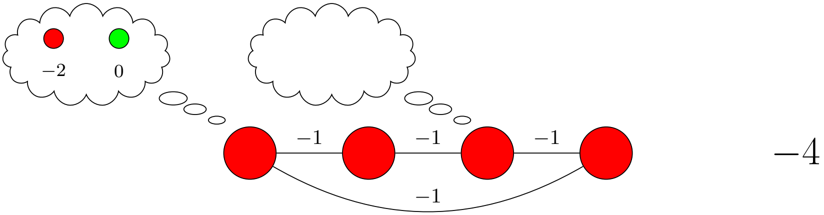
- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one



-4

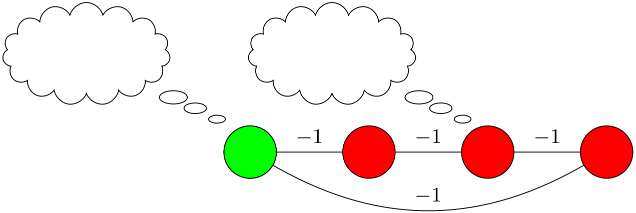
Centralized Local Greedy approaches

- Greedy local search
  - Start from random solution
  - Do **local** changes if global solution improves
  - **Local**: change the value of a subset of variables, usually one



Centralized Local Greedy approaches

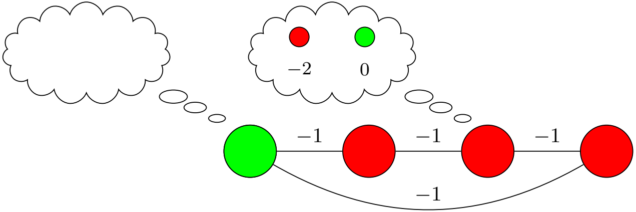
- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one



$$-2$$

Centralized Local Greedy approaches

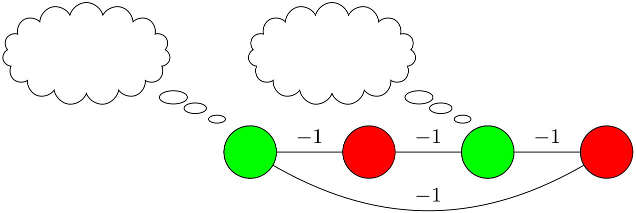
- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one



-2

Centralized Local Greedy approaches

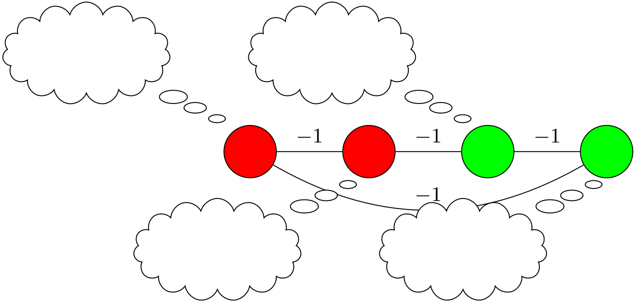
- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one



-0

Centralized Local Greedy approaches

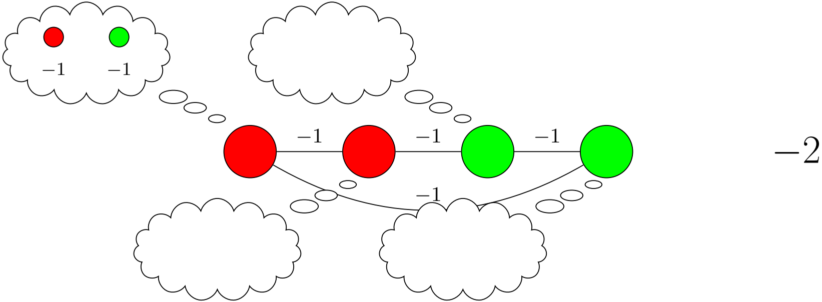
- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing



$$-2$$

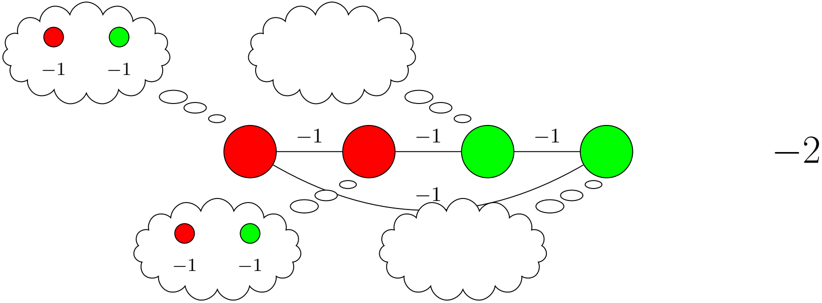
Centralized Local Greedy approaches

- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing



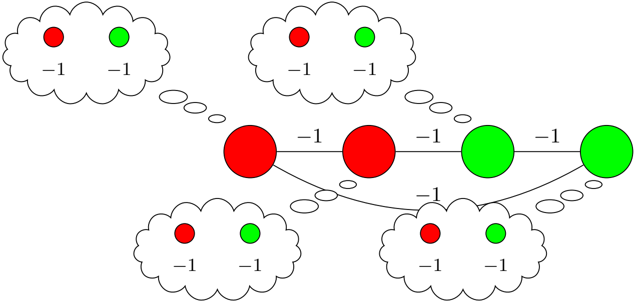
Centralized Local Greedy approaches

- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing



Centralized Local Greedy approaches

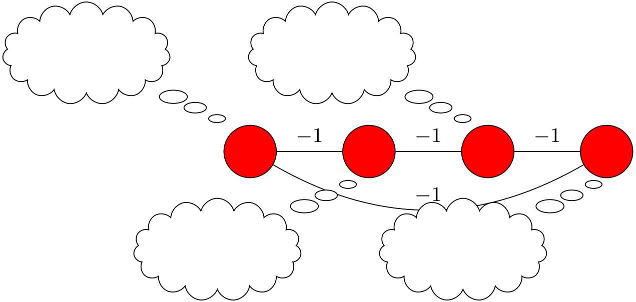
- **Problems**
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing



-2

Distributed Local Greedy approaches

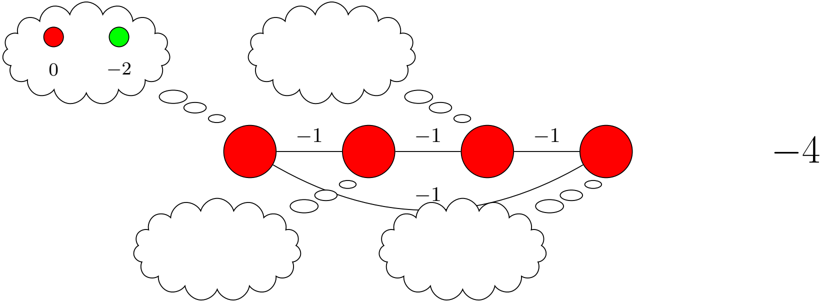
- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination



-4

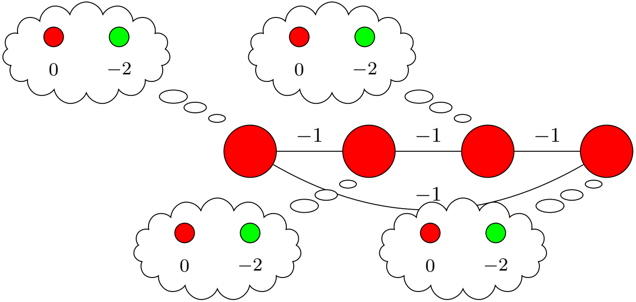
Distributed Local Greedy approaches

- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination



Distributed Local Greedy approaches

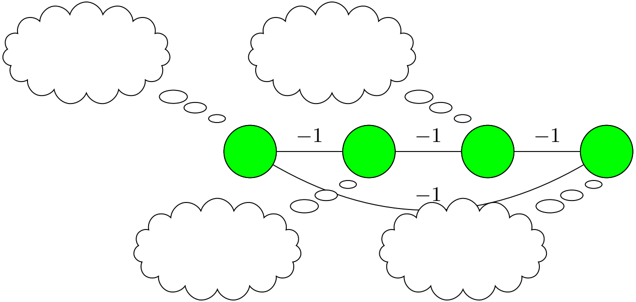
- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination



-4

Distributed Local Greedy approaches

- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination

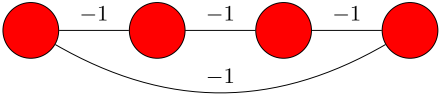


$$-4$$

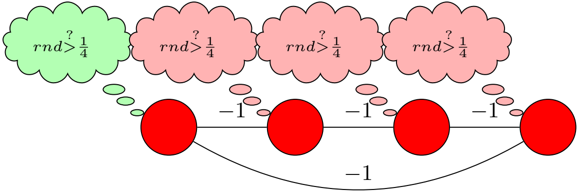
Distributed Stochastic Search Algorithm (DSA) [Zhang et al., 2005]

- Greedy local search with activation probability to mitigate issues with parallel executions
- DSA-1: change value of one variable at time
- Initialize agents with a random assignment and communicate values to neighbors
- Each agent:
  - Generates a random number and execute only if rnd less than activation probability
  - When executing changes value maximizing local gain
  - Communicate possible variable change to neighbors

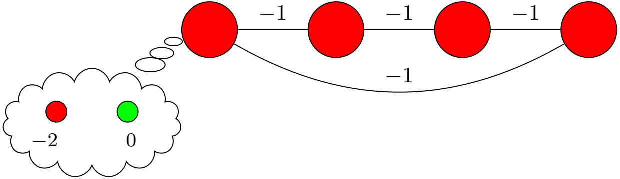
DSA-1: Execution Example



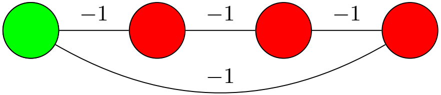
DSA-1: Execution Example



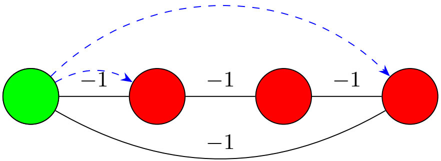
DSA-1: Execution Example



DSA-1: Execution Example



DSA-1: Execution Example



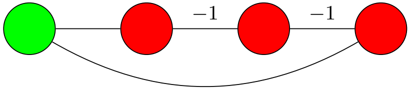
DSA-1: Discussion

- Extremely "cheap" (computation/communication)
- Good performance in various domains
  - e.g. target tracking [Fitzpatrick and Meertens, 2003; Zhang et al., 2003]
  - Shows an anytime property (not guaranteed)
  - Benchmarking technique for coordination
- Problems
  - Activation probablity must be tuned [Zhang et al., 2003]
  - No general rule, hard to characterise results across domains

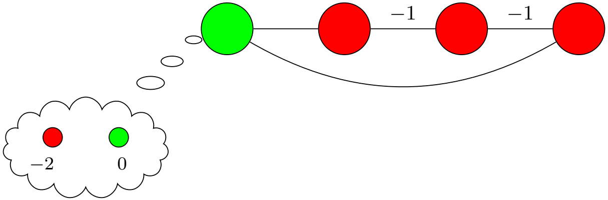
Maximum Gain Message (MGM-1) [MAHESWARAN et al., 2004]

- Coordinate to decide who is going to move
  - Compute and exchange possible gains
  - Agent with maximum (positive) gain executes
- Analysis
  - Empirically, similar to DSA
  - More communication (but still linear)
  - No Threshold to set
  - Guaranteed to be monotonic (Anytime behavior)

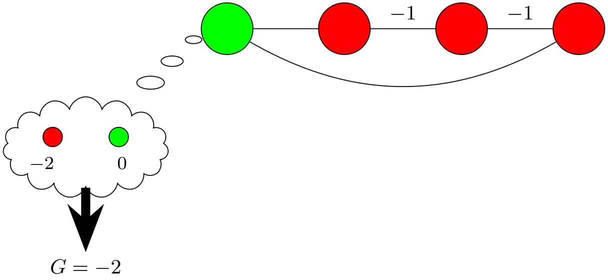
MGM-1: Example



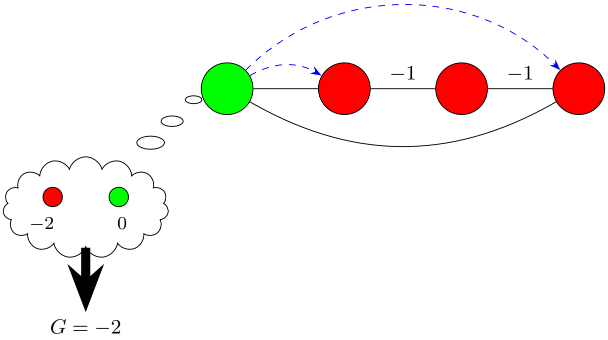
MGM-1: Example



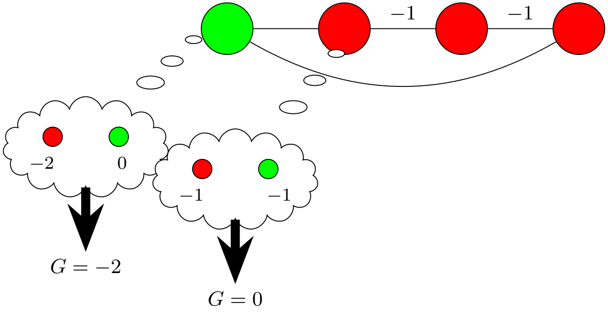
MGM-1: Example



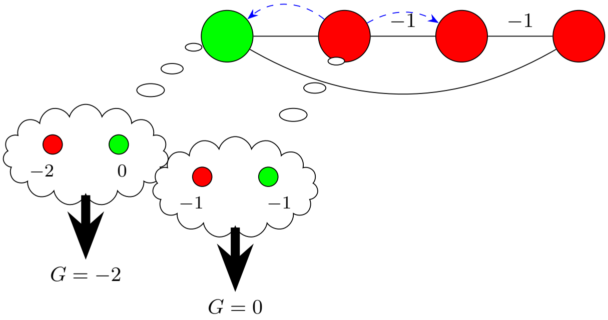
MGM-1: Example



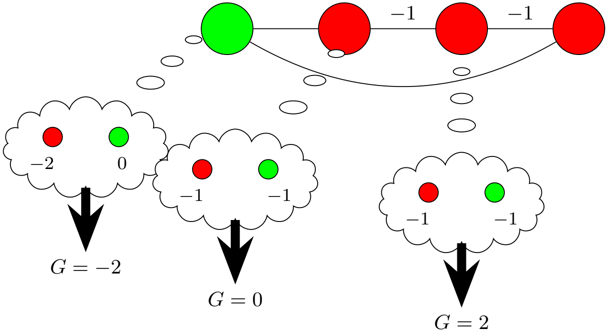
MGM-1: Example



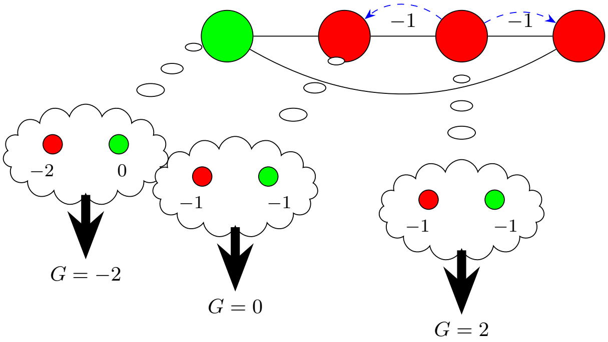
MGM-1: Example



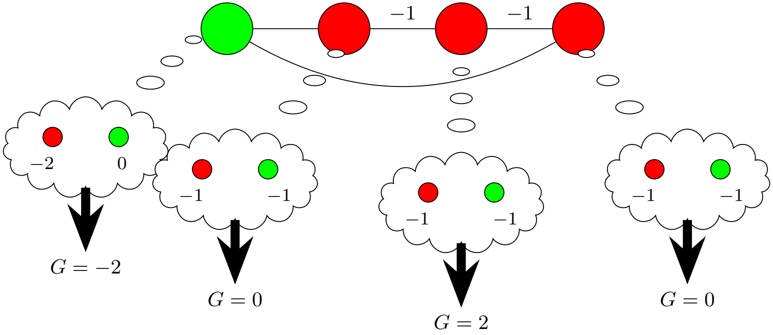
MGM-1: Example



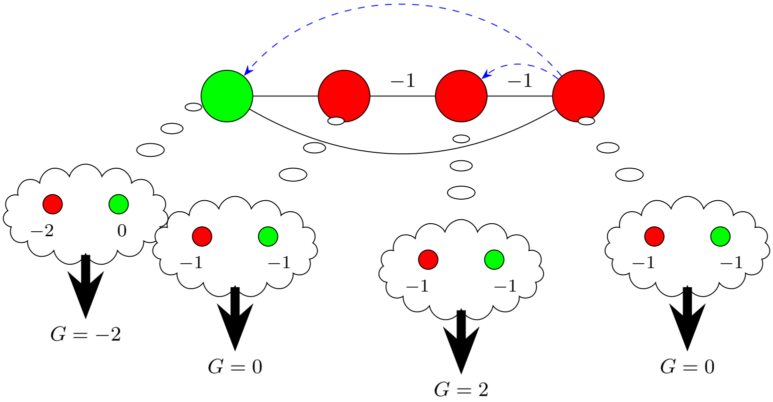
MGM-1: Example



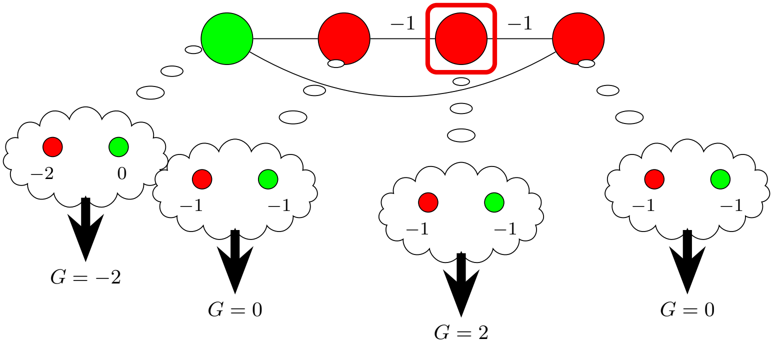
MGM-1: Example



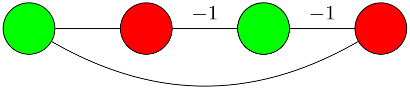
MGM-1: Example



MGM-1: Example



MGM-1: Example



To sum up on local greedy approaches

- Exchange local values for variables
  - Similar to search based methods (e.g. ADOPT)
- Consider only local information when maximizing
  - Values of neighbors
- Anytime behaviors
- Could result in very bad solutions

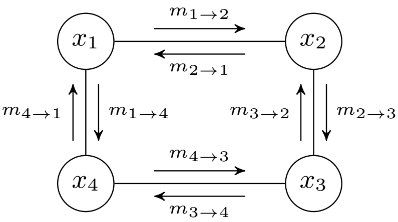
GDL-based approaches

- Generalized Distributive Law [Aji and McEliece, 2000]
  - Unifying framework for inference in Graphical models
  - Builds on basic mathematical properties of semi-rings
  - Widely used in Info theory, Statistical physics, Probabilistic models
- Max-sum
  - DCOP settings: maximise social welfare

	K	"$(+,0)$"	"$(\cdot,1)$"	short name
1.	A	$(+,0)$	$(\cdot,1)$	
2.	$A[x]$	$(+,0)$	$(\cdot,1)$	
3.	$A[x,y,\ldots]$	$(+,0)$	$(\cdot,1)$	
4.	$[0,\infty)$	$(+,0)$	$(\cdot,1)$	sum-product
5.	$(0,\infty]$	$(\min,\infty)$	$(\cdot,1)$	min-product
6.	$[0,\infty)$	$(\max,0)$	$(\cdot,1)$	max-product
7.	$(-\infty,\infty]$	$(\min,\infty)$	$(+,0)$	min-sum
8.	$[-\infty,\infty)$	$(\max,-\infty)$	$(+,0)$	max-sum
9.	$\{0,1\}$	(OR, 0)	(AND, 1)	Boolean
10.	2^S	$(\cup,\emptyset)$	$(\cap,S)$	
11.	Λ	$(\vee,0)$	$(\wedge,1)$	
12.	Λ	$(\wedge,1)$	$(\vee,0)$.	

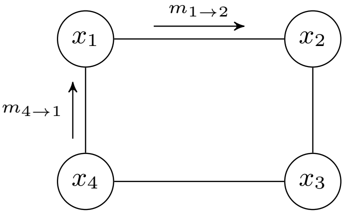
Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



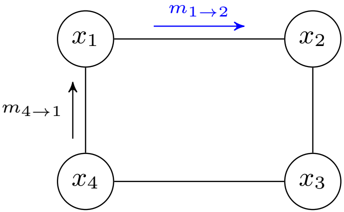
Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



Max-Sum

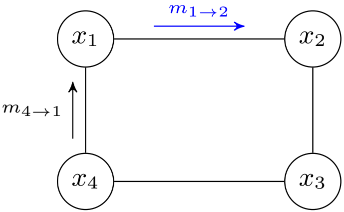
Agents iteratively computes local functions that depend only on the variable they control



$$m_{1\to 2}(x_2) = max_{x_1}(F_{12}(x_1, x_2) + m_{4\to 1}(x_1))$$

Max-Sum

Agents iteratively computes local functions that depend only on the variable they control

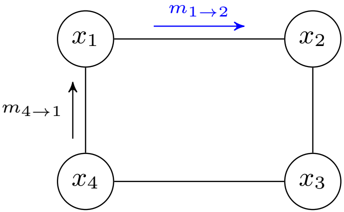


Shared constraint

$$m_{1\to 2}(x_2) = max_{x_1}(\boxed{\quad\quad\quad} + m_{4\to 1}(x_1))$$

Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



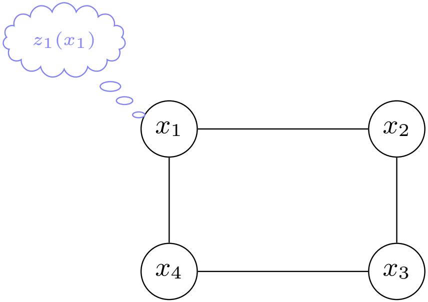
Shared constraint

$$m_{1\to 2}(x_2) = max_{x_1}(\quad + \quad))$$

All incoming messages except x_2

Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



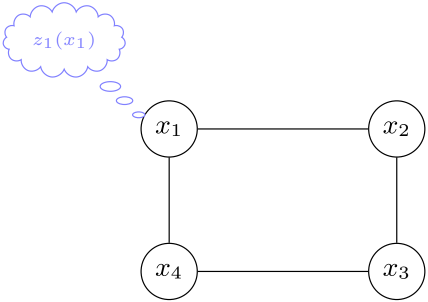
Shared constraint

$$m_{1\to 2}(x_2) = max_{x_1}(\quad + \quad))$$

All incoming messages except x_2

Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



Shared constraint

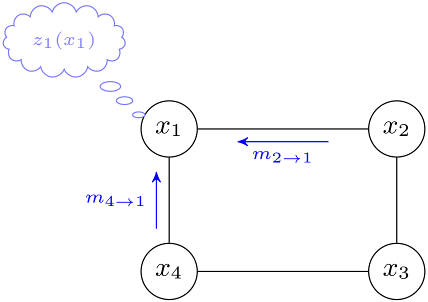
$$m_{1\to 2}(x_2) = max_{x_1}(\quad + \quad))$$

All incoming messages except x_2

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

Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



Shared constraint

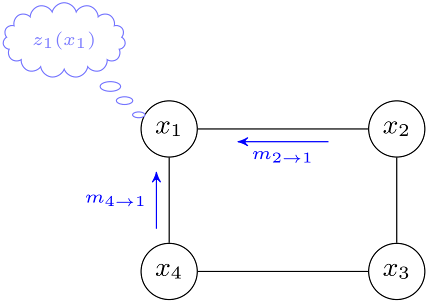
$$m_{1\to2}(x_2) = max_{x_1}(\qquad + \qquad))$$

All incoming messages except x_2

$$z_1(x_1) = m_{4\to1}(x_1) + m_{2\to1}(x_1)$$

Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



Shared constraint

$$m_{1\rightarrow 2}(x_2) = max_{x_1}(\qquad + \qquad))$$

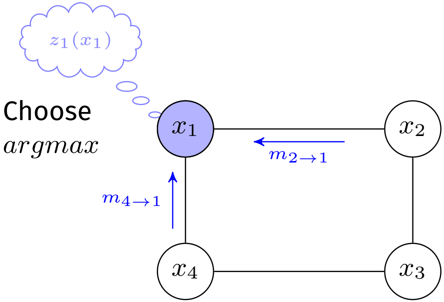
All incoming messages except x_2

$$z_1(x_1) = \qquad$$

All incoming

Max-Sum

Agents iteratively computes local functions that depend only on the variable they control



Shared constraint

$$m_{1\to2}(x_2) = max_{x_1}(\quad + \quad))$$

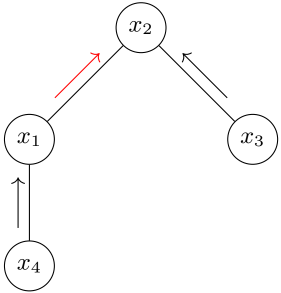
All incoming messages except x_2

$$z_1(x_1) =$$

All incoming

Max-Sum on acyclic graphs

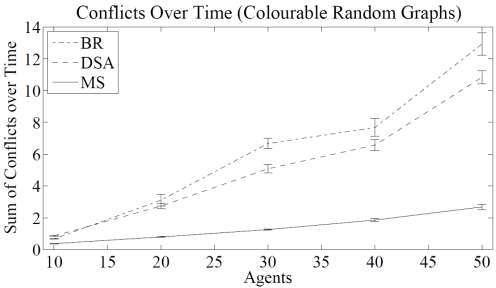
- Max-sum Optimal on acyclic graphs
  - Different branches are independent
  - Each agent can build a correct estimation of its contribution to the global problem (z functions)
- Message equations very similar to Util messages in DPOP
  - Sum messages from children and shared constraint
  - Maximize out agent variable
  - GDL generalizes DPOP [VINYALS et al., 2011]

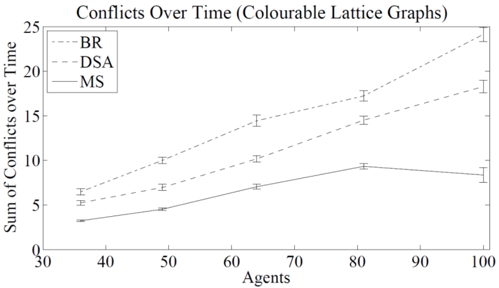


$$m_{1\to 2}(x_2) = max_{x_1}(F_{12}(x_1, x_2) + m_{4\to 1}(x_1))$$

Max-sum Performance

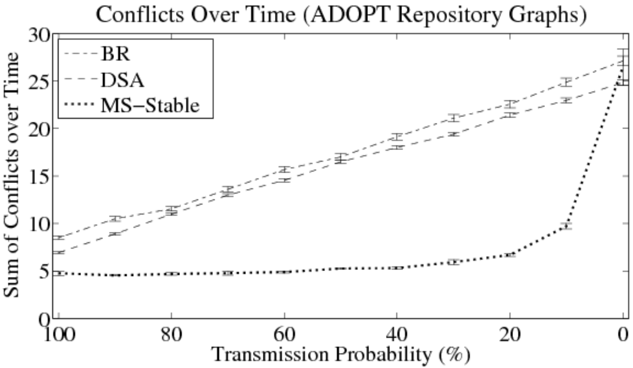
- Good performance on loopy networks [Farinelli et al., 2008]
  - When it converges very good results
    - Interesting results when only one cycle [Weiss, 2000]
  - We could remove cycle but pay an exponential price (see DPOP)





Max-Sum for low power devices

- Low overhead
  - Msgs number/size
- Asynchronous computation
  - Agents take decisions whenever new messages arrive
- Robust to message loss



Contents

Panorama

Algorithm	Type	Memory	Messages	Remarks
ADOPT	COP	Polynomial	Exponential	Complete
DPOP	COP	Exponential	Linear	Complete
DSA	COP	Linear	?	Not complete
MGM	COP	Linear	?	Not complete
Max-Sum	COP	Exponential	Linear on acyclic	Complete on trees

Table: DCOP algorithms

References

Aji, S.M. and R.J. McEliece (2000). "The generalized distributive law". In: *Information Theory, IEEE Transactions on* 46.2, pp. 325–343. ISSN: 0018-9448. DOI: 10.1109/18.825794.

Farinelli, A., A. Rogers, A. Petcu, and N. R. Jennings (2008). "Decentralised Coordination of Low-power Embedded Devices Using the Max-sum Algorithm". In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS '08. International Foundation for Autonomous Agents and Multiagent Systems, pp. 639–646. ISBN: 978-0-9817381-1-6.

Fitzpatrick, Stephen and Lambert Meertens (2003). "Distributed Coordination through Anarchic Optimization". In: *Distributed Sensor Networks: A Multiagent Perspective*. Ed. by Victor Lesser, Charles L. Ortiz, and Milind Tambe. Boston, MA: Springer US, pp. 257–295. ISBN: 978-1-4615-0363-7.

Maheswaran, R.T., J.P. Pearce, and M. Tambe (2004). "Distributed Algorithms for DCOP: A Graphical-Game-Based Approach". In: *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS), San Francisco, CA*, pp. 432–439.

Modi, P. J., W. Shen, M. Tambe, and M. Yokoo (2005). "ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees". In: *Artificial Intelligence* 161.2, pp. 149–180.

Petcu, Adrian and Boi Faltings (2005). "A scalable method for multiagent constraint optimization". In: *IJCAI International Joint Conference on Artificial Intelligence*, pp. 266–271. ISBN: 1045-0823.

Vinyals, Meritxell, Juan A. Rodríguez-Aguilar, and Jesus Cerquides (2011). "Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law". In: *Autonomous Agents and Multi-Agent Systems* 3.22, pp. 439–464. ISSN: 1387-2532. DOI: 10.1007/s10458-010-9132-7.

Weiss, Yair (Jan. 2000). "Correctness of Local Probability Propagation in Graphical Models with Loops". In: *Neural Comput.* 12.1, pp. 1–41. ISSN: 0899-7667. DOI: 10.1162/089976600300015880. URL: http://dx.doi.org/10.1162/089976600300015880.

Yokoo, M. (2001). *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer.

References (cont.)

ZHANG, W., G. WANG, Z. XING, and L. WITTENBURG (2005). "Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks.". In: *Journal of Artificial Intelligence Research (JAIR)* 161.1-2, pp. 55–87.

ZHANG, Weixiong, Guandong WANG, Zhao XING, and Lars WITTENBURG (2003). "A Comparative Study of Distributed Constraint Algorithms". In: *Distributed Sensor Networks: A Multiagent Perspective*. Ed. by Victor LESSER, Charles L. ORTIZ, and Milind TAMBE. Boston, MA: Springer US, pp. 319–338.