Distributed Constraint Optimization for the Internet-of-Things

Pierre Rust Gauthier Picard

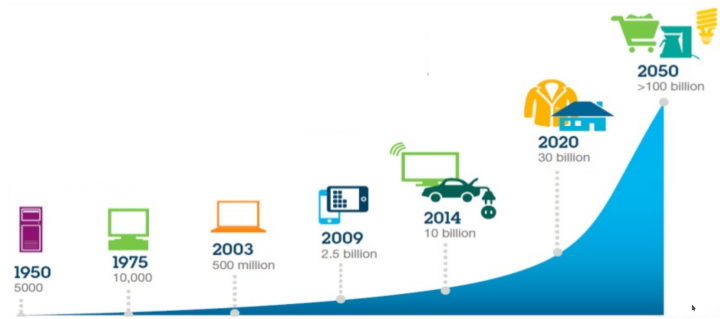
Orange Labs
MINES Saint-Étienne
LaHC UMR CNRS 5516

pierre.rust@orange.com

Internet-of-Things (IoT) and its Control

- Huge (marketing ?) trend today
- 25 billion of connected objects in 2020 ? (Gartner)
- Hardware and communication is cheaper and cheaper
- Contrained devices
  - limited cpu and memory resources
  - limited communication capabilities

- Connected things' actions should be **coordinated**
- Current approach: centralizing decisions
  - Communications
  - Resilience
  - Scalability
  - Cost



Distributed Coordination and Decision Making

Autonomous and spontaneous

Coordinating objects to achieve objectives

- Coordination
  - Decentralized
  - Spontaneous
  - Autonomous
- No central point
- Self-adaption to environmental changes
- Self-repair in case of one component failure

About decisions

$$x_i \ ?$$

About decisions

$$x_i \ ?$$

s.t. "I'm happy with x_i"

About decisions

x_i ?

s.t. "I'm happy with x_i"

x_j ?

s.t. "agent i is fine with x_j"

About decisions

x_i ?

s.t. "I'm happy with x_i"

x_j ?

s.t. "agent i is fine with x_j"

How can agents autonomously make their decisions
in a coordinated way, without external control ?

About decisions

x_i ?

s.t. "I'm happy with x_i"

x_j ?

s.t. "agent i is fine with x_j"

How can agents autonomously make their decisions in a coordinated way, without external control ?

$\Rightarrow$ Decentralized decision making

About decisions

x_i ?

s.t. "I'm happy with x_i"

x_j ?

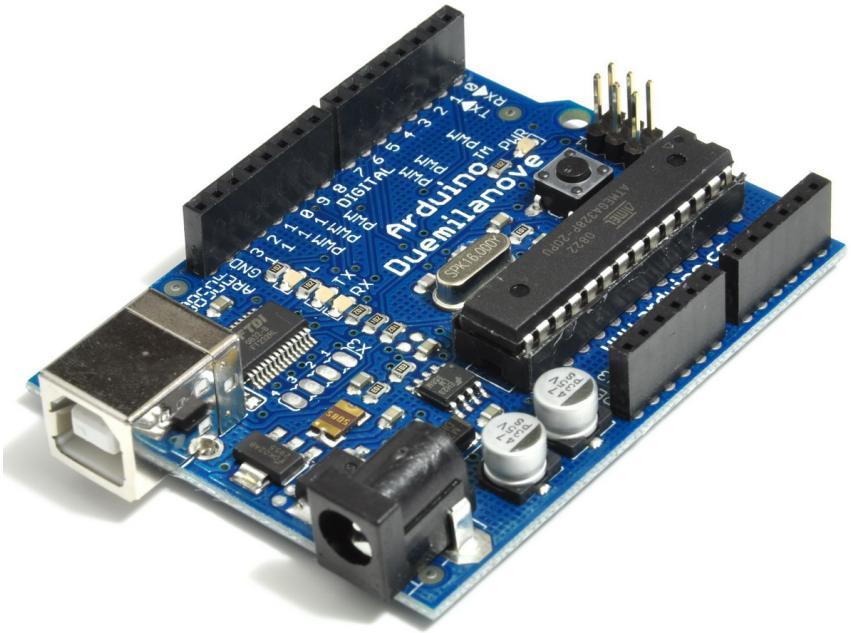
s.t. "agent i is fine with x_j"

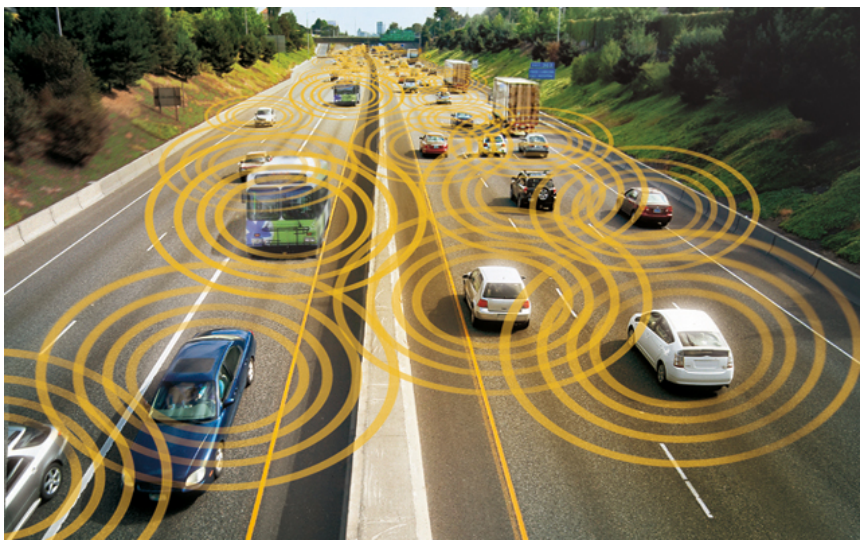
How can agents autonomously make their decisions in a coordinated way, without external control ?

$\Rightarrow$ Decentralized decision making

- Agents have to coordinate to perform best actions
- Agents form a team $\rightarrow$ best actions for the team

Application Domains





Menu

DCOP Framework

Focus on Some Solution Methods

Hands on PyDCOP I

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP II

Dynamic DCOPs

Conclusion

Menu

DCOP[1]

Distributed Constraints Optimization Problem

Definition (DCOP)

A DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu \rangle$, where:

- $\mathcal{A} = \{a_1, \ldots, a_{|A|}\}$ is a set of agents
- $\mathcal{X} = \{x_1, \ldots, x_n\}$ are variables
- $\mathcal{D} = \{\mathcal{D}_{x_1}, \ldots, \mathcal{D}_{x_n}\}$ is a set of finite domains, for the x_i variables
- $\mathcal{C} = \{f_1, \ldots, f_m\}$ is a set of soft constraints, where each c_i defines a cost $\in \mathbb{R} \cup \{\infty\}$ for each combination of assignments to a subset of variables
- μ is a function mapping variables to their associated agent

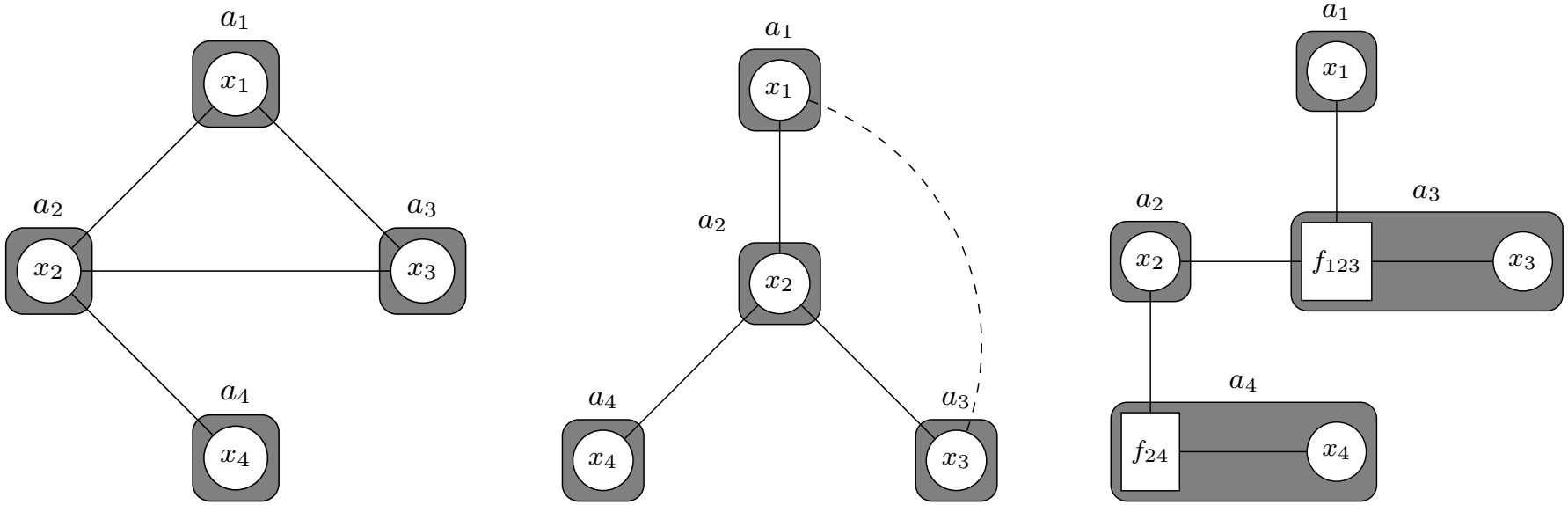
Definition (Solution)

A *solution* to the DCOP is an assignment $\mathcal{A}$ to all variables that minimizes $\sum_i f_i$

[1] Some contents taken from OPTMAS 2011 and OPTMAS-DCR 2014

DCOP

Example and Graphical Representation



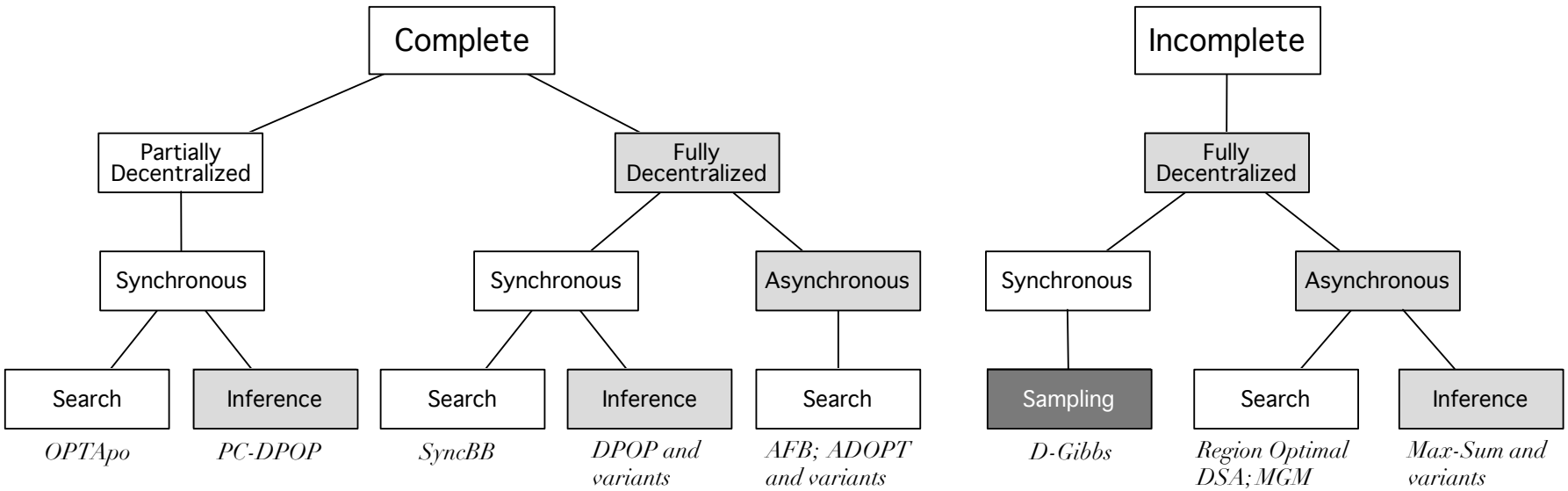
Objective Function

$$F(\mathcal{A}) = \sum_{x_i, x_j \in X} f_{ij} \quad \text{where } f_{ij} = (\mathbf{x}_i + \mathbf{x}_j + 1) mod 3$$

In figure *(a)*:

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

DCOP Algorithms



(FIORETTO et al., 2018)

Menu

DCOP Framework

Focus on Some Solution Methods
- DPOP
- Max-Sum
- DSA
- MGM

Hands on PyDCOP I

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP II

Dynamic DCOPs

Conclusion

Distributed Pseudotree Optimization Procedure (DPOP)

(PETCU and FALTINGS, 2005)

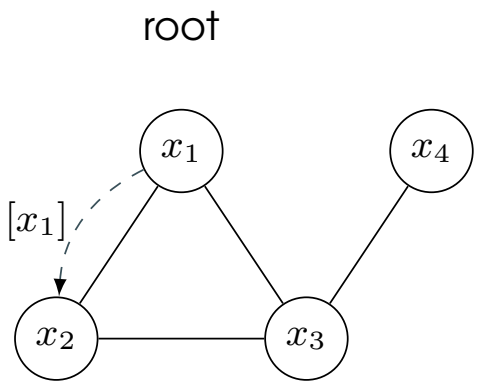
3-phase distributed algorithm

	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

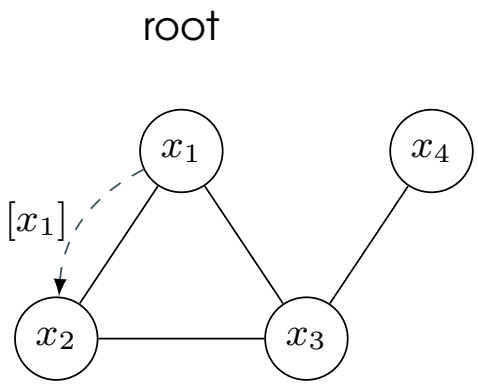
- **Distributed DFS graph traversal**: token, ID, $neighbors(X)$
  1. X owns the token: adds its own ID and sends it in turn to each of its neighbors, which become children
  2. Y receives the token from X: it marks X as visited. First time Y receives the token then $parent(Y) = X$. Other IDs in token 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

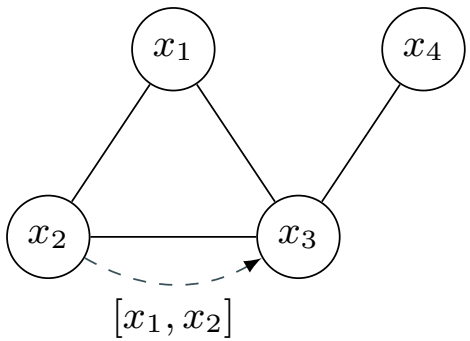


x_1 parent of x_2

DFS Tree Phase: Example

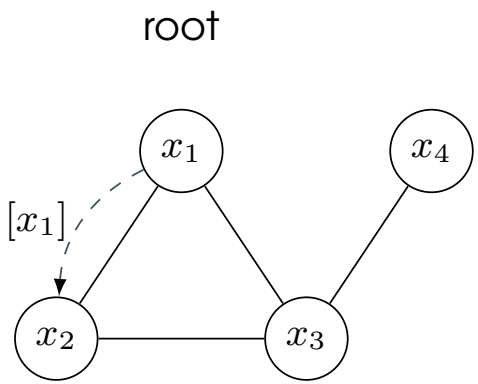


x_1 parent of x_2

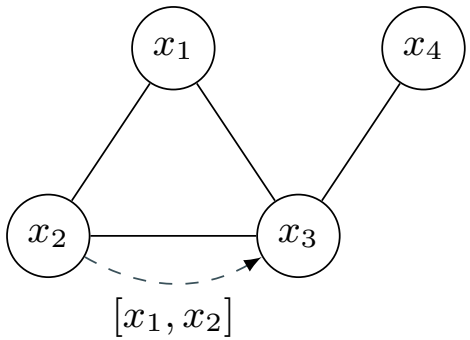


x_2 parent of x_3
x_1 pseudoparent of x_3

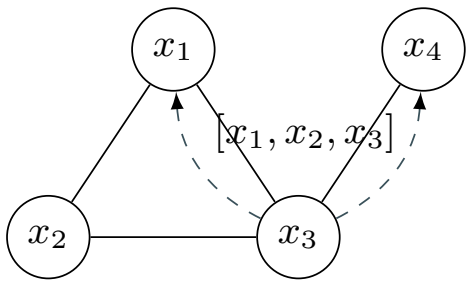
DFS Tree Phase: Example



x_1 parent of x_2

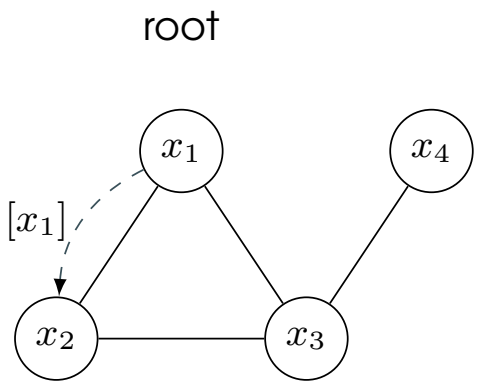


x_2 parent of x_3
x_1 pseudoparent of x_3

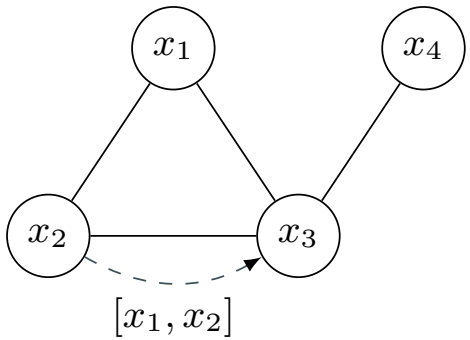


x_3 parent of x_4
x_3 pseudoparent of x_1

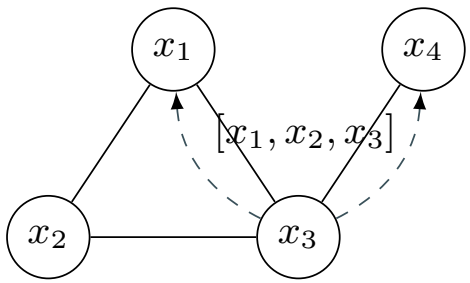
DFS Tree Phase: Example



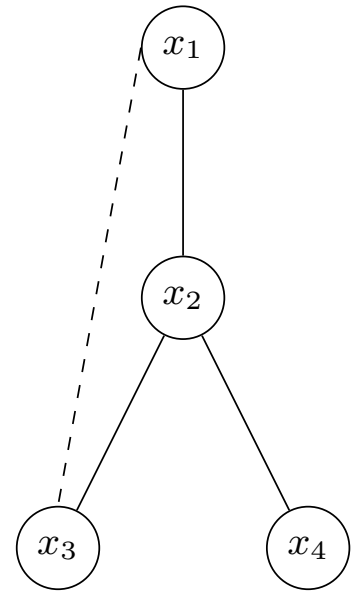
x_1 parent of x_2



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

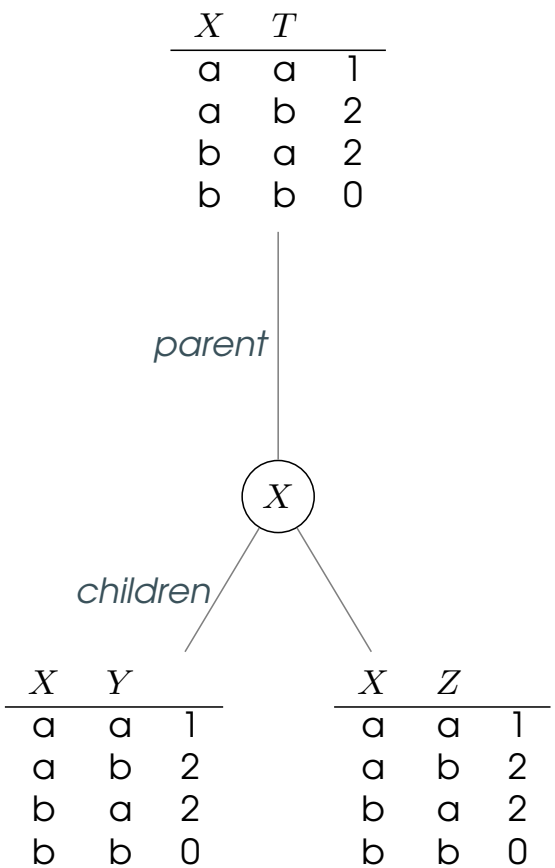
Agent X:

- 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 $parent(X)$

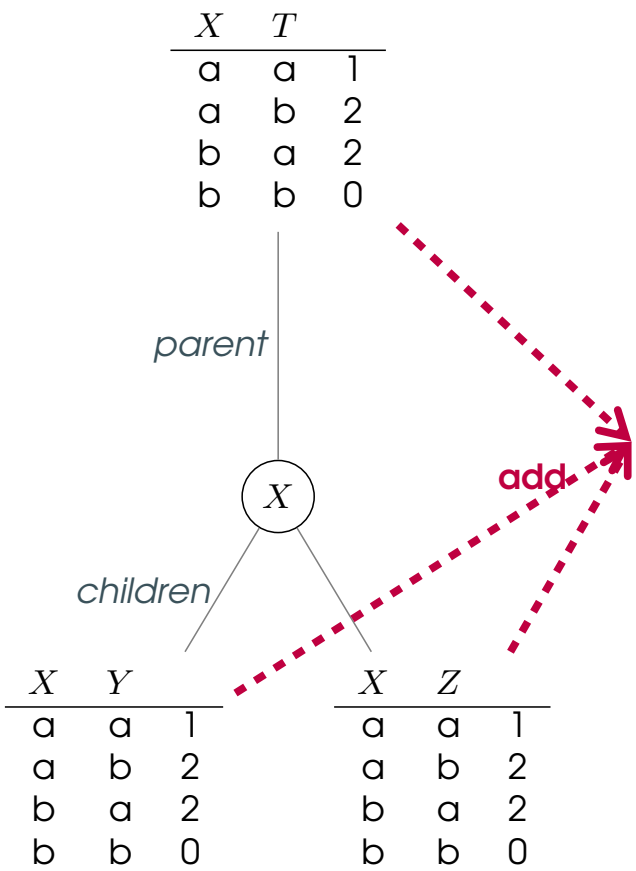
From the leaves to the root

Util Phase: Example

Util Phase: Example



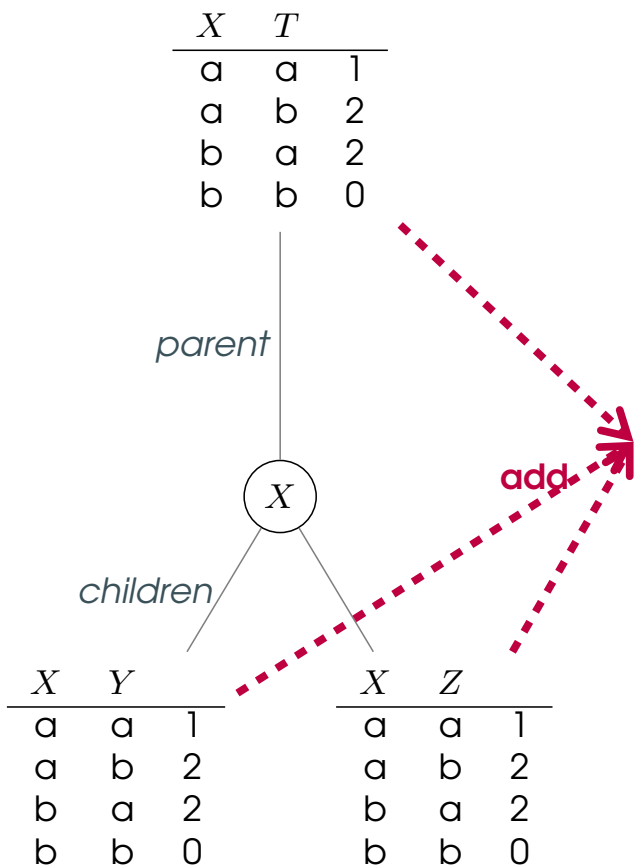
Util Phase: Example



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	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

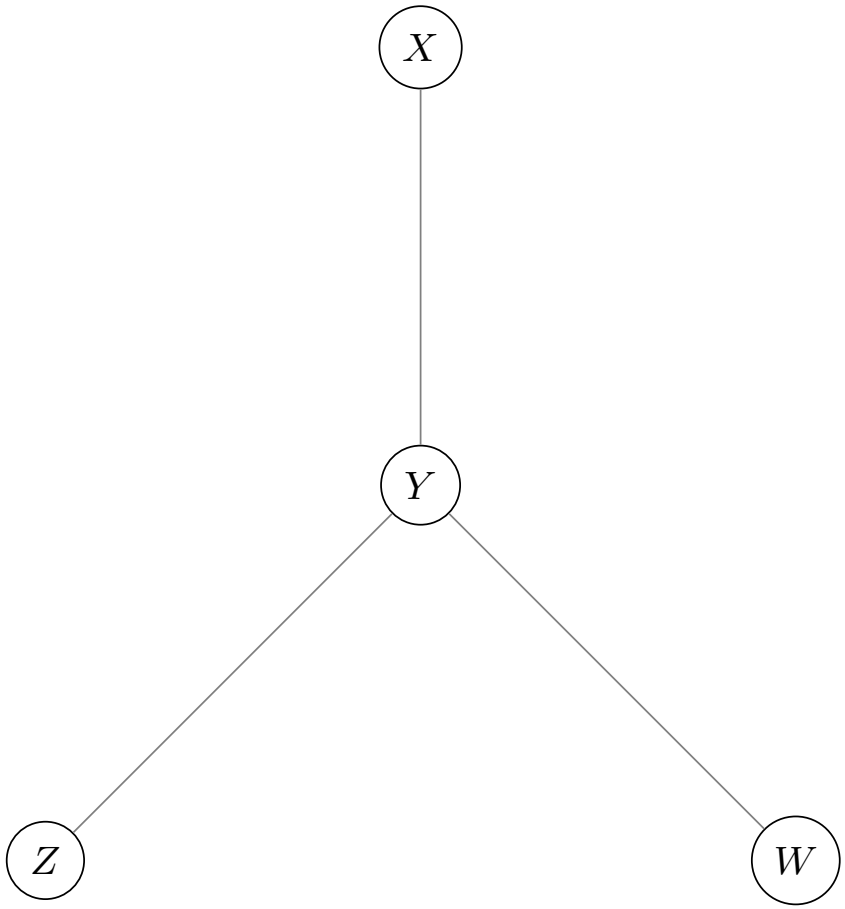
Remove X
Remove duplicates
Keep the min cost

Value Phase

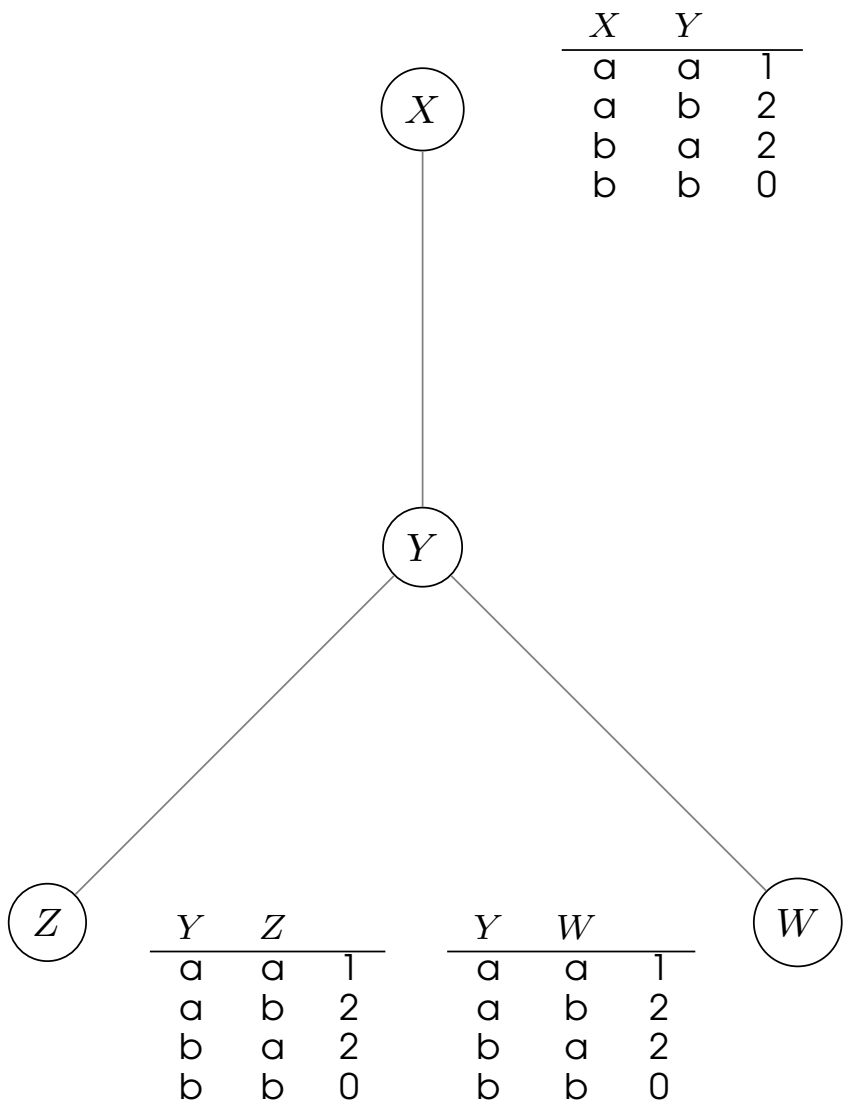
1. The root finds the **value that minimizes the received cost function** in the util phase, and informs its descendants (children ∪ pseudochildren)
2. Each agent **waits to receive** the value of its parent / pseudoparents
3. Keeping fixed the value of parent/pseudoparents, finds **the value that minimizes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

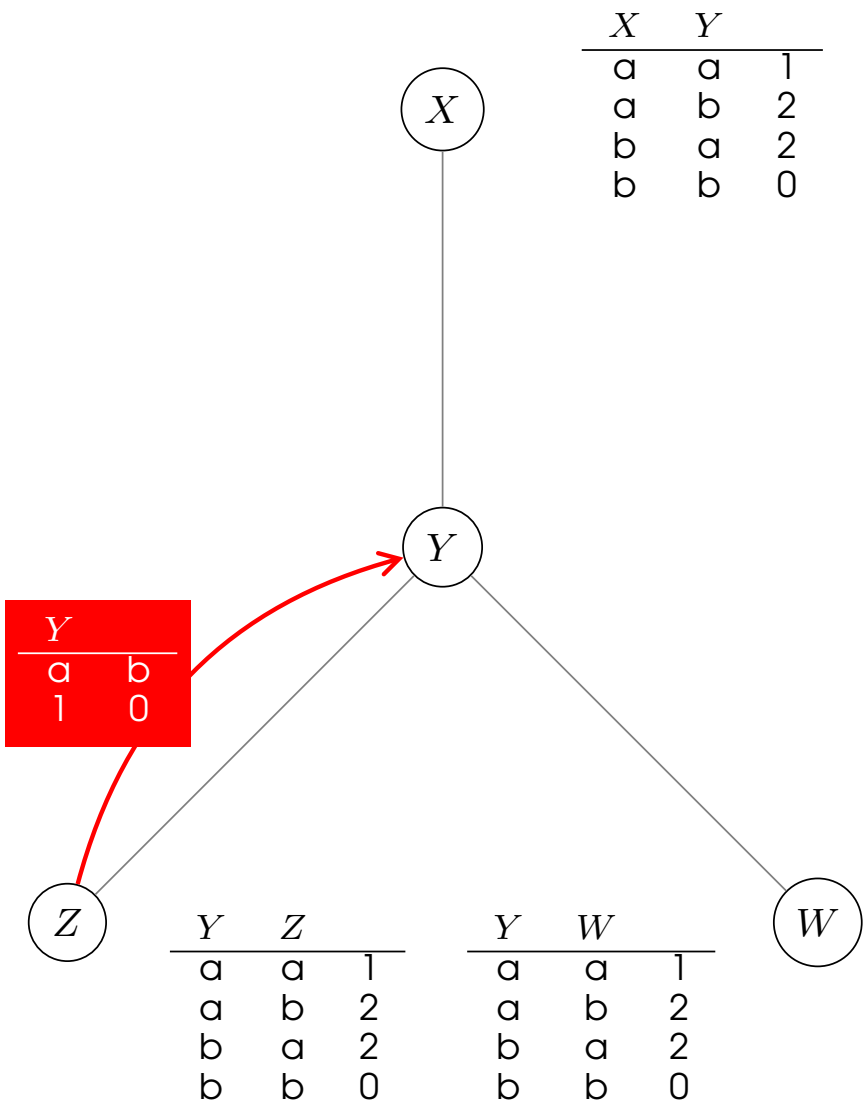
DTREE : DPOP for DCOPs without backedges



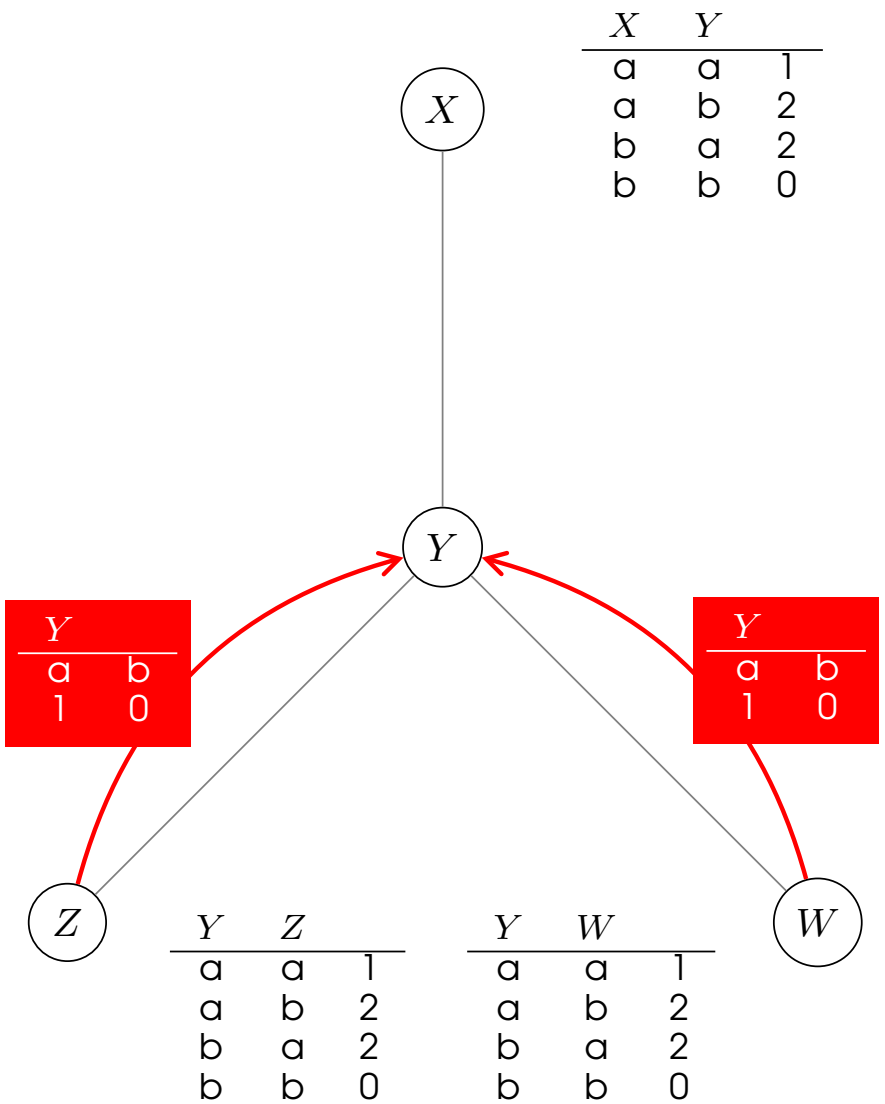
DTREE : DPOP for DCOPs without backedges



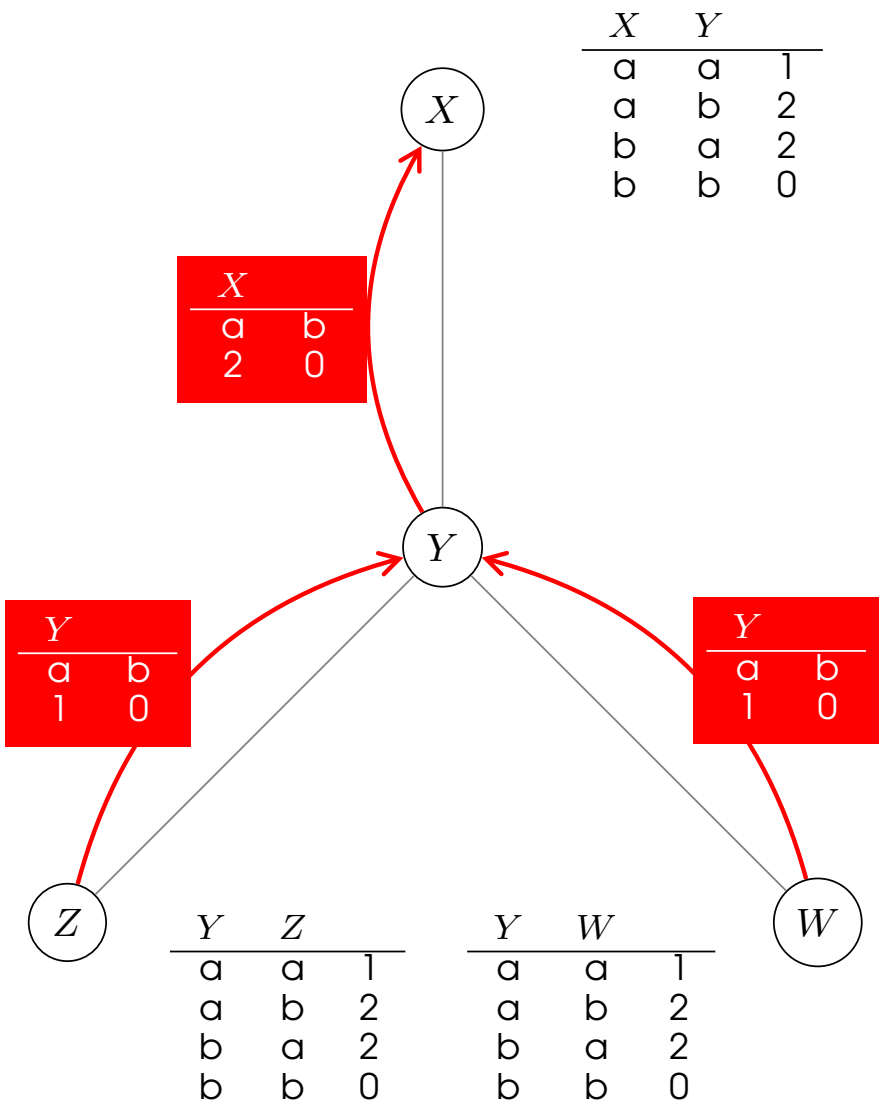
DTREE : DPOP for DCOPs without backedges



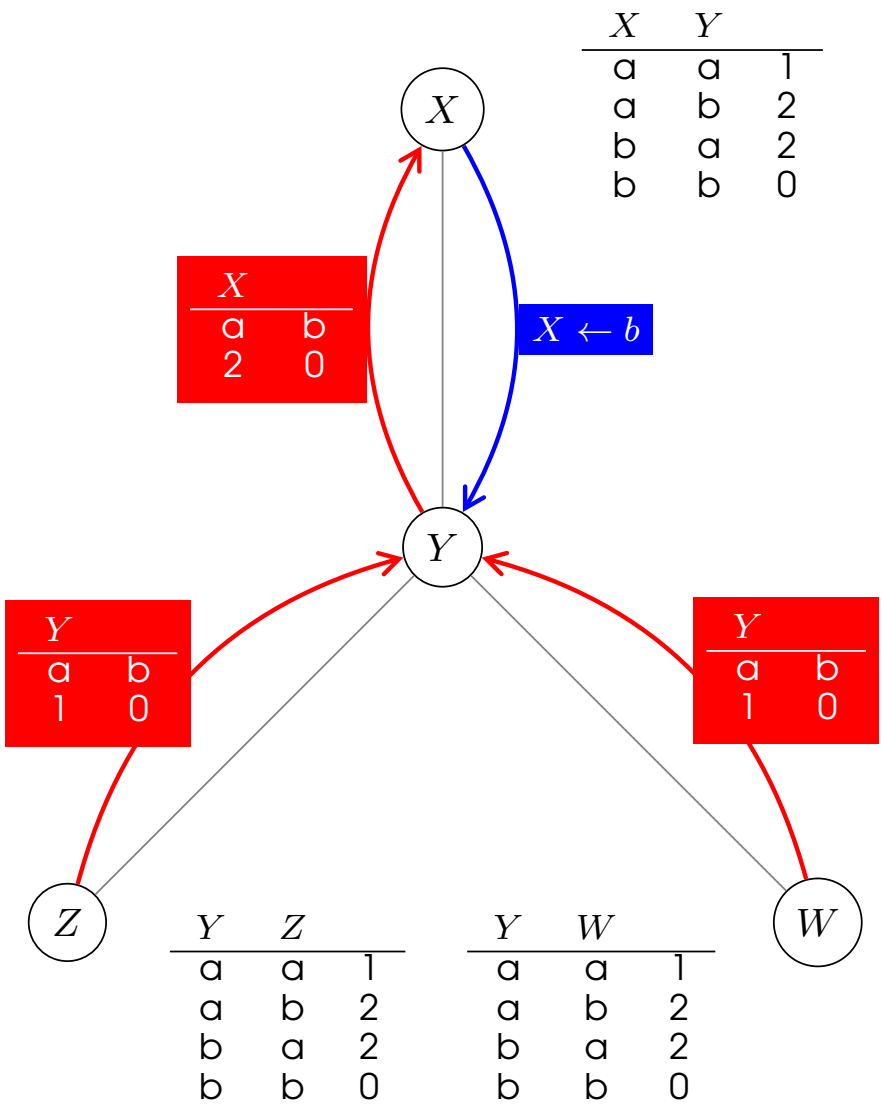
DTREE : DPOP for DCOPs without backedges



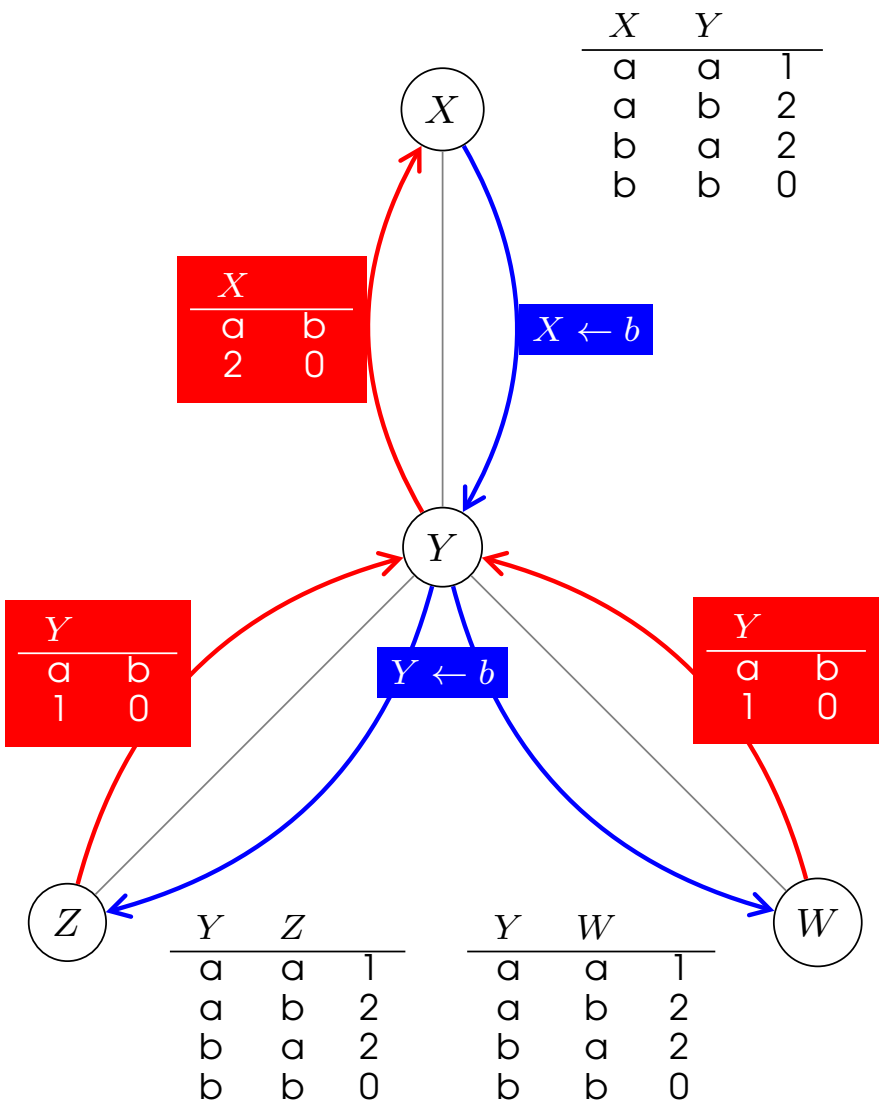
DTREE : DPOP for DCOPs without backedges



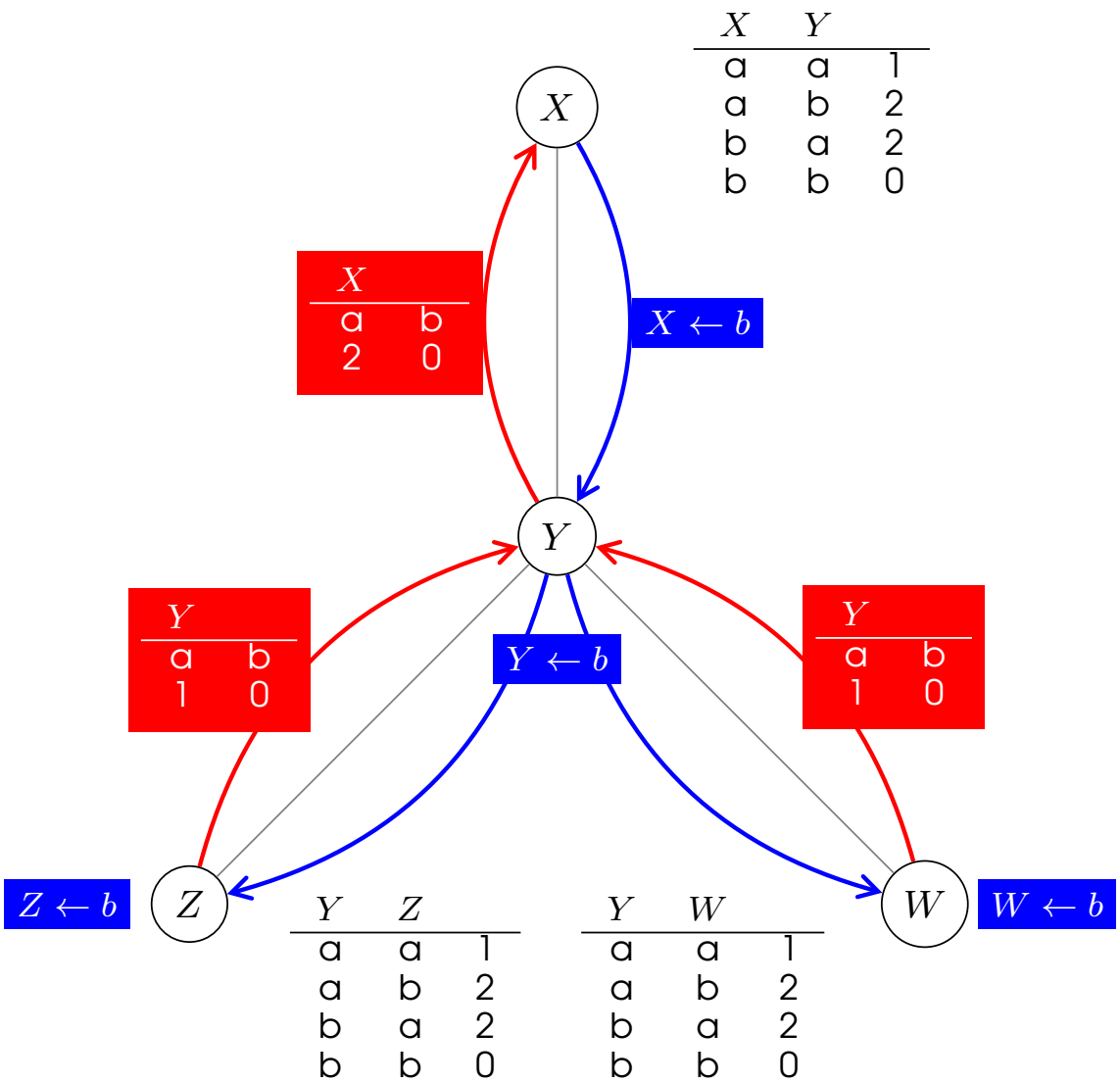
DTREE : DPOP for DCOPs without backedges



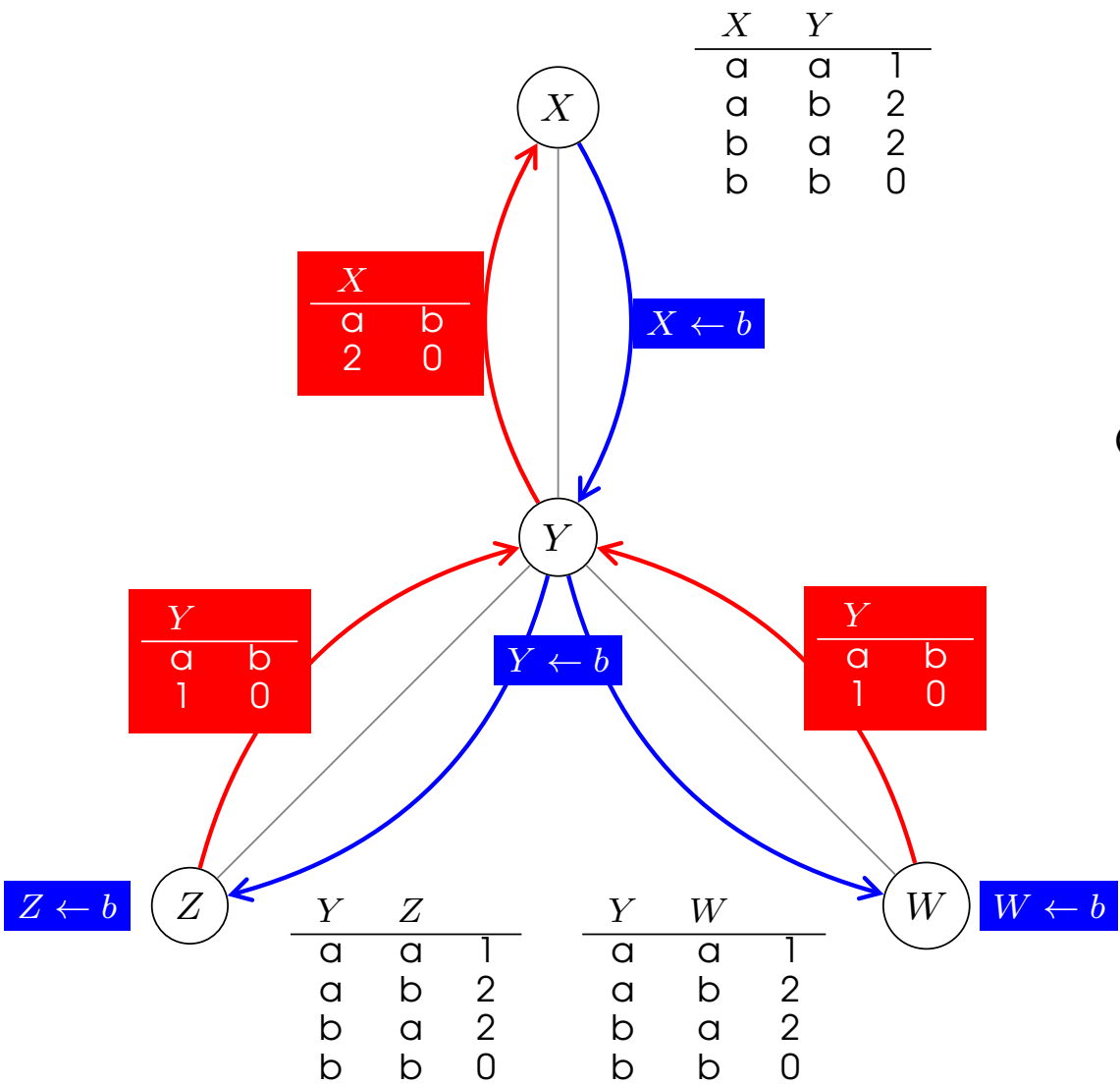
DTREE : DPOP for DCOPs without backedges



DTREE : DPOP for DCOPs without backedges



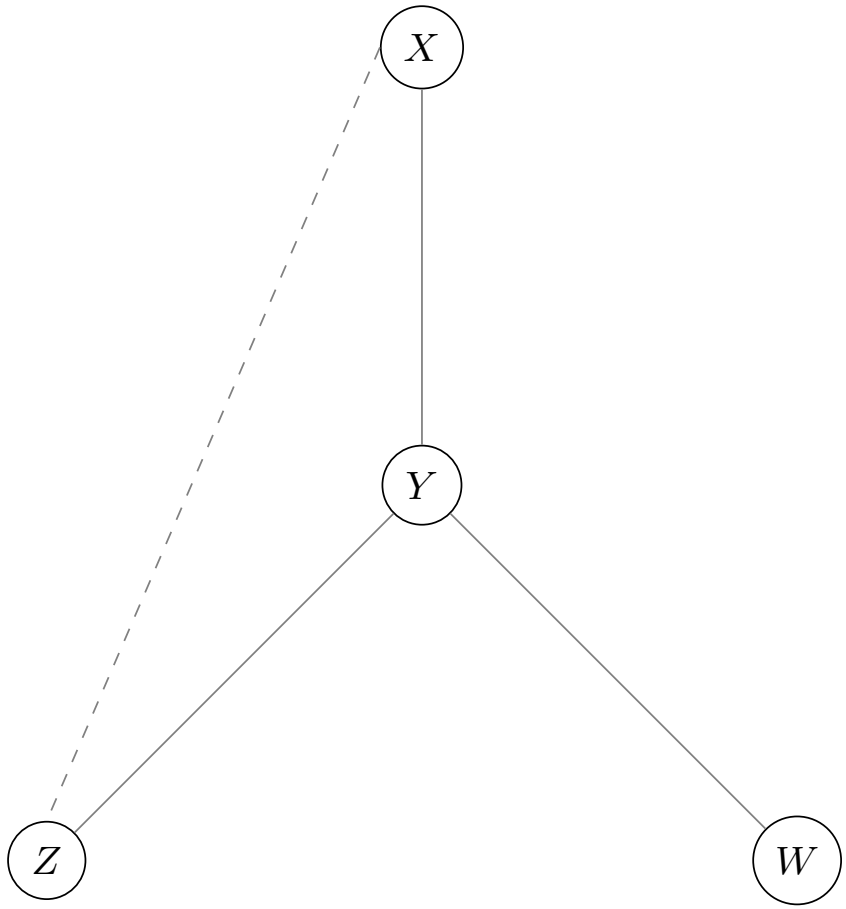
DTREE : DPOP for DCOPs without backedges



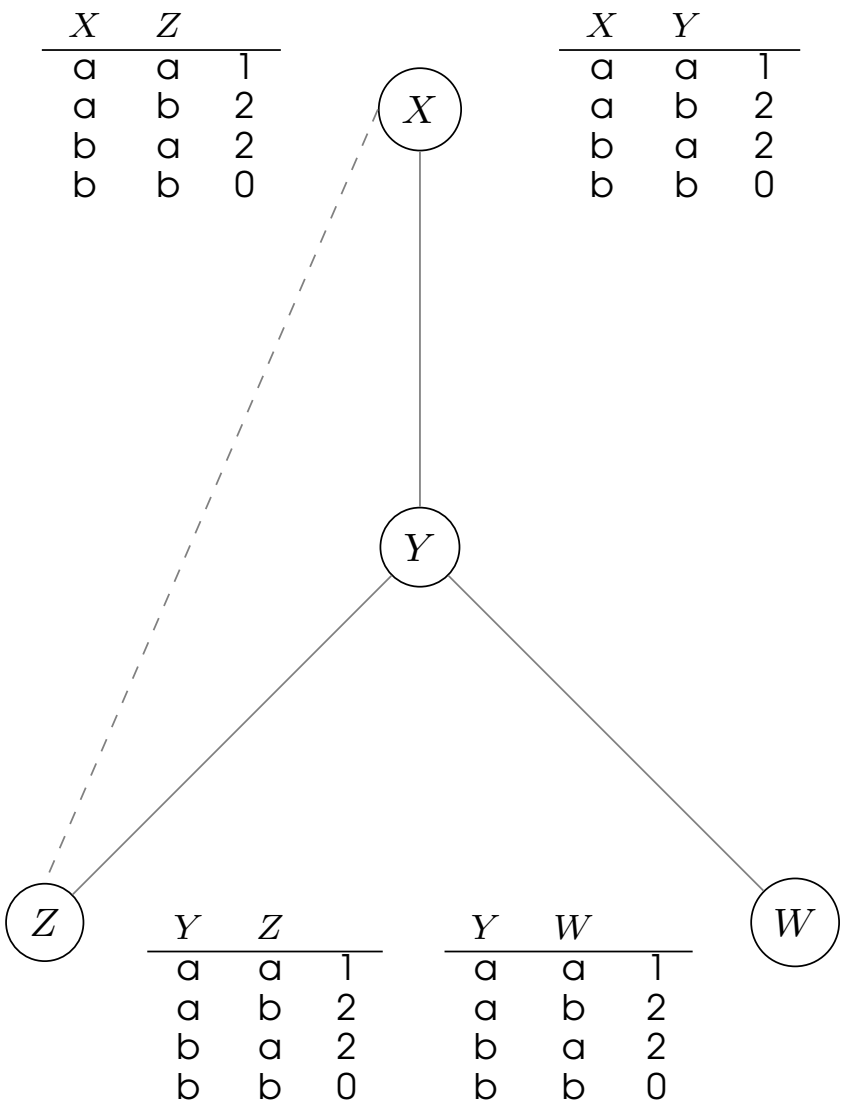
Optimal solution:

- linear number of messages
- message size: linear

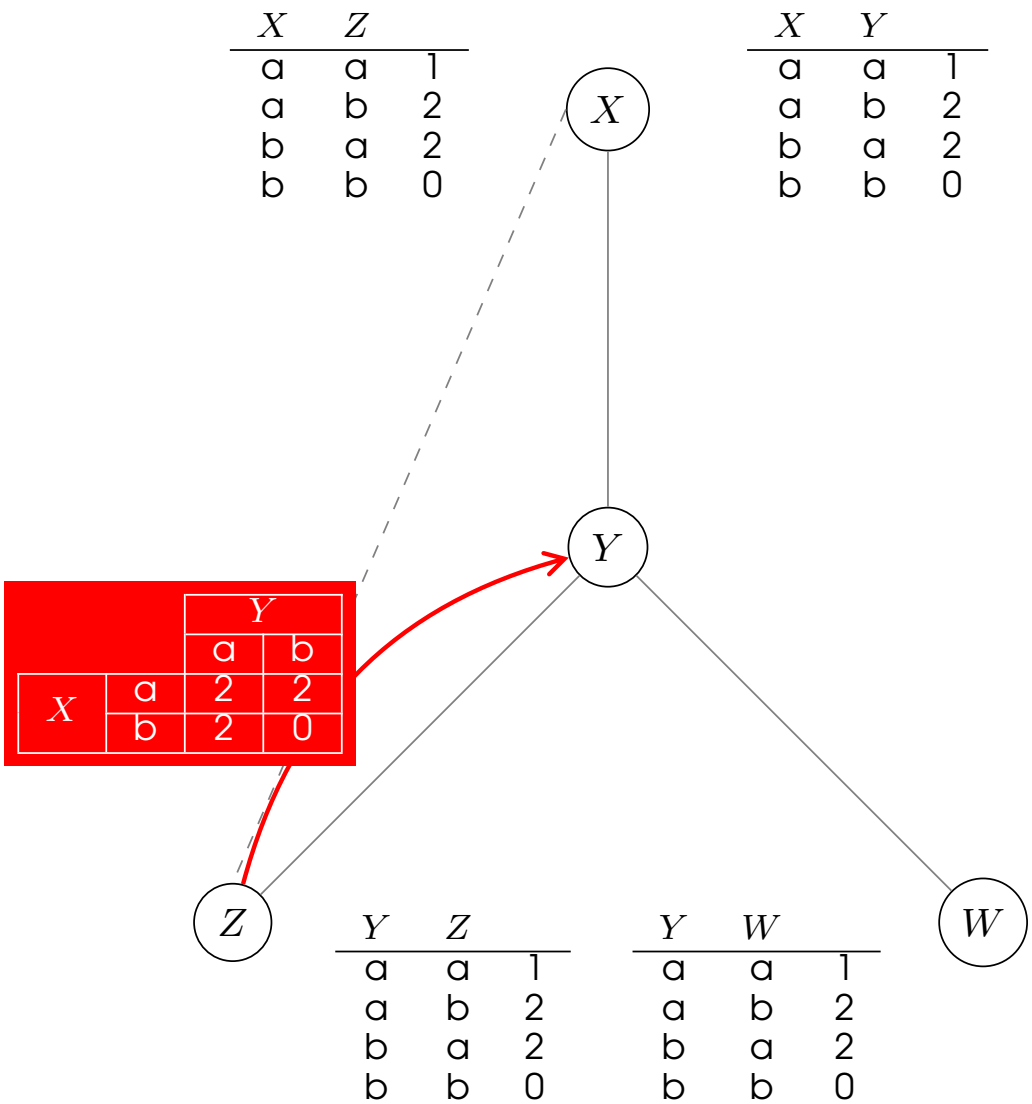
DPOP for any DCOP



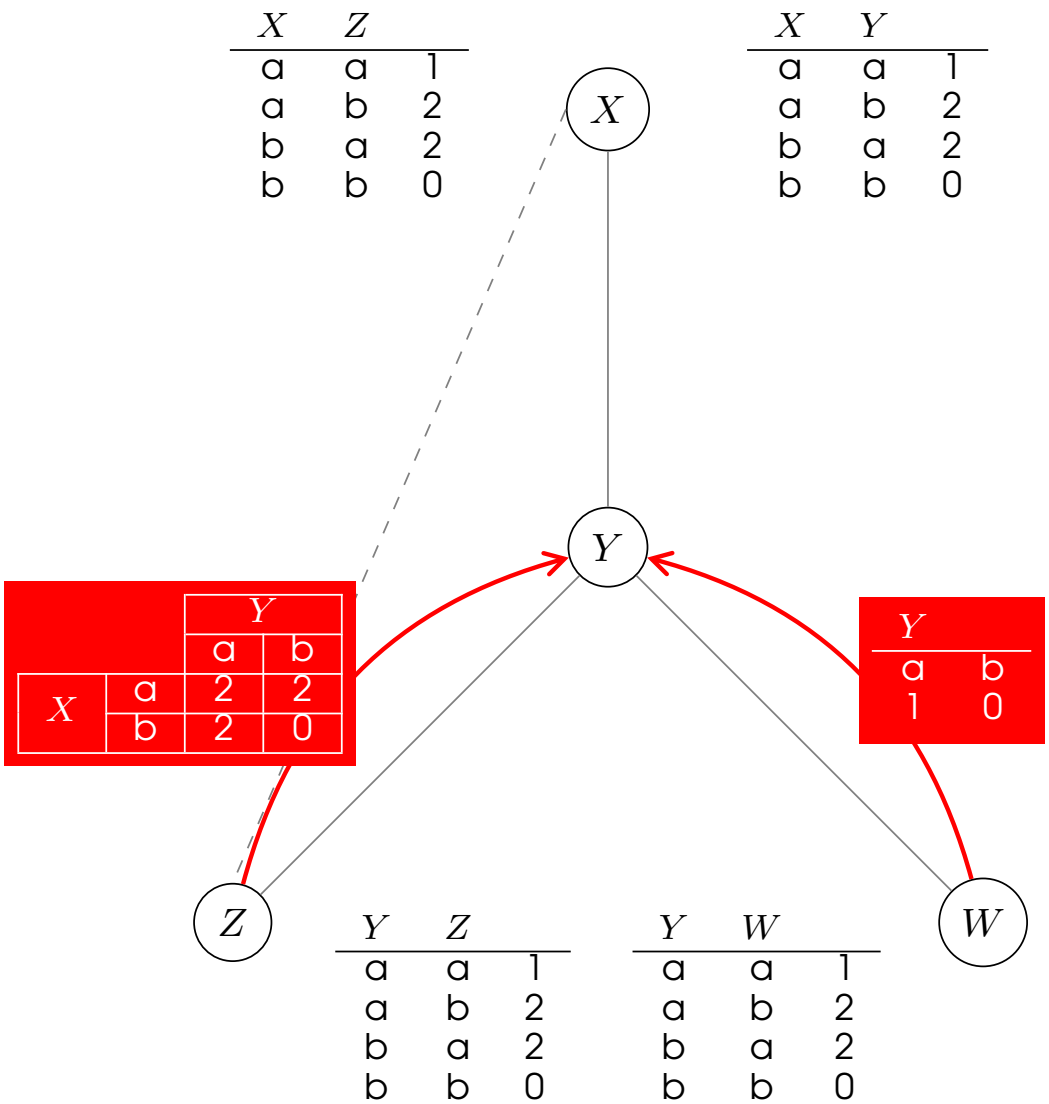
DPOP for any DCOP



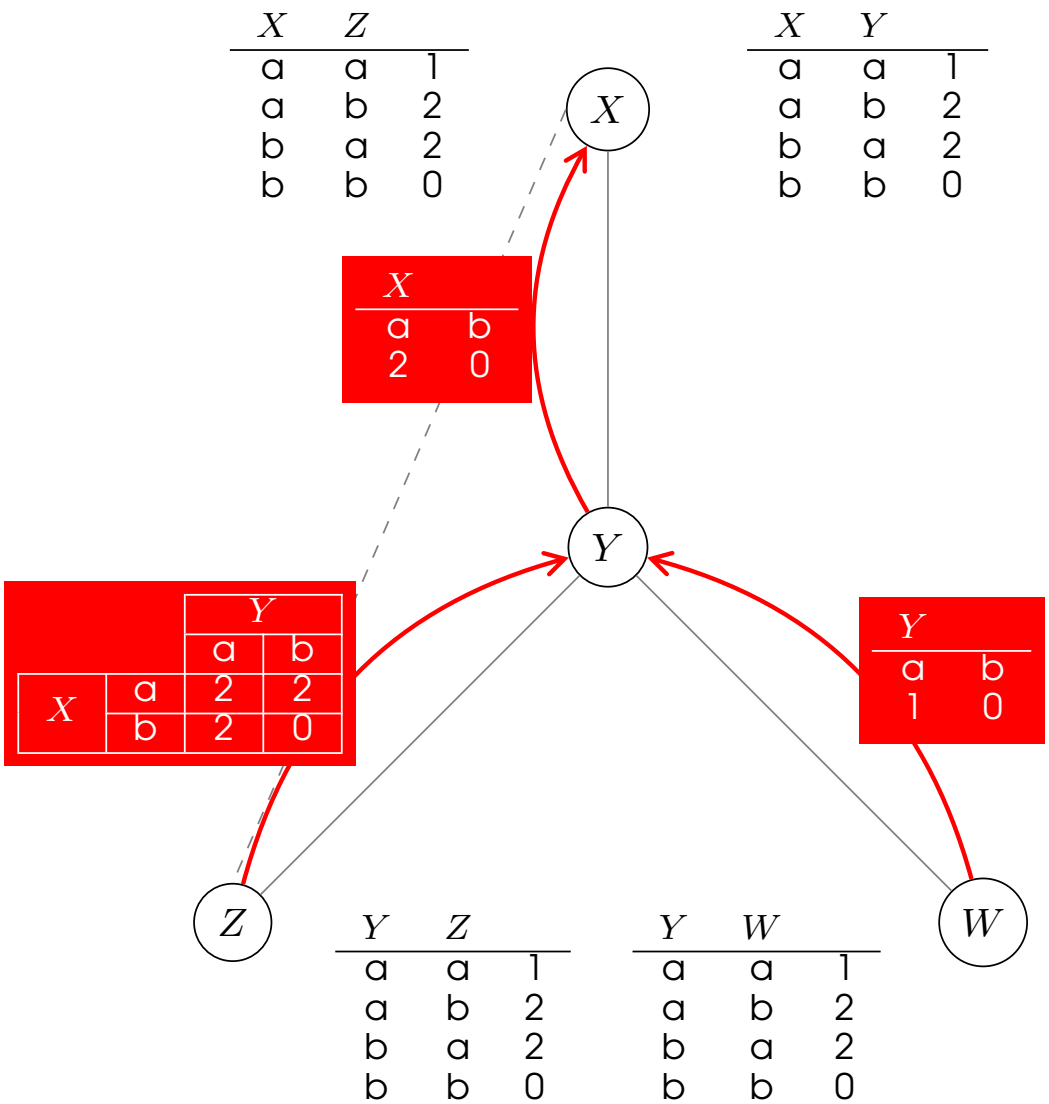
DPOP for any DCOP



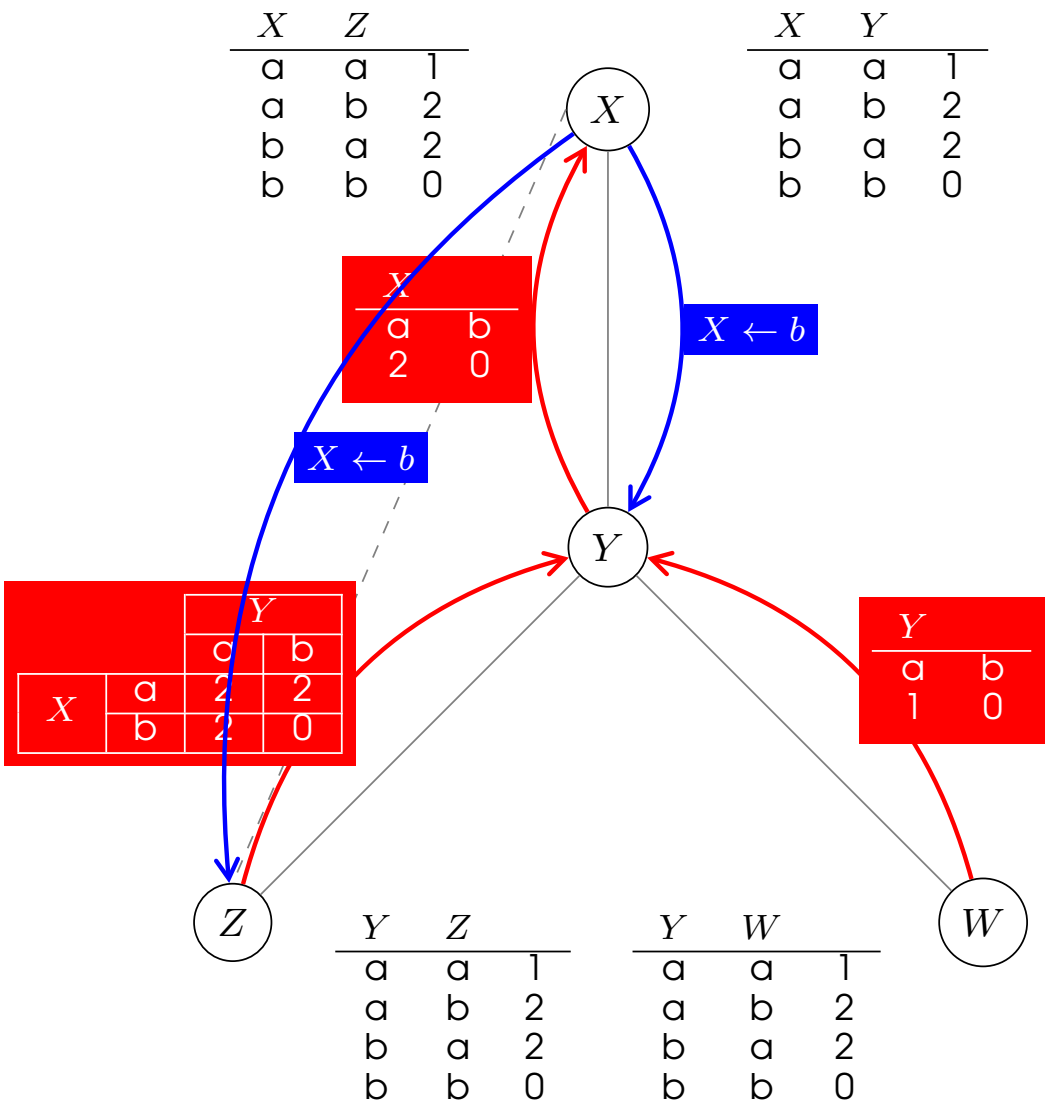
DPOP for any DCOP



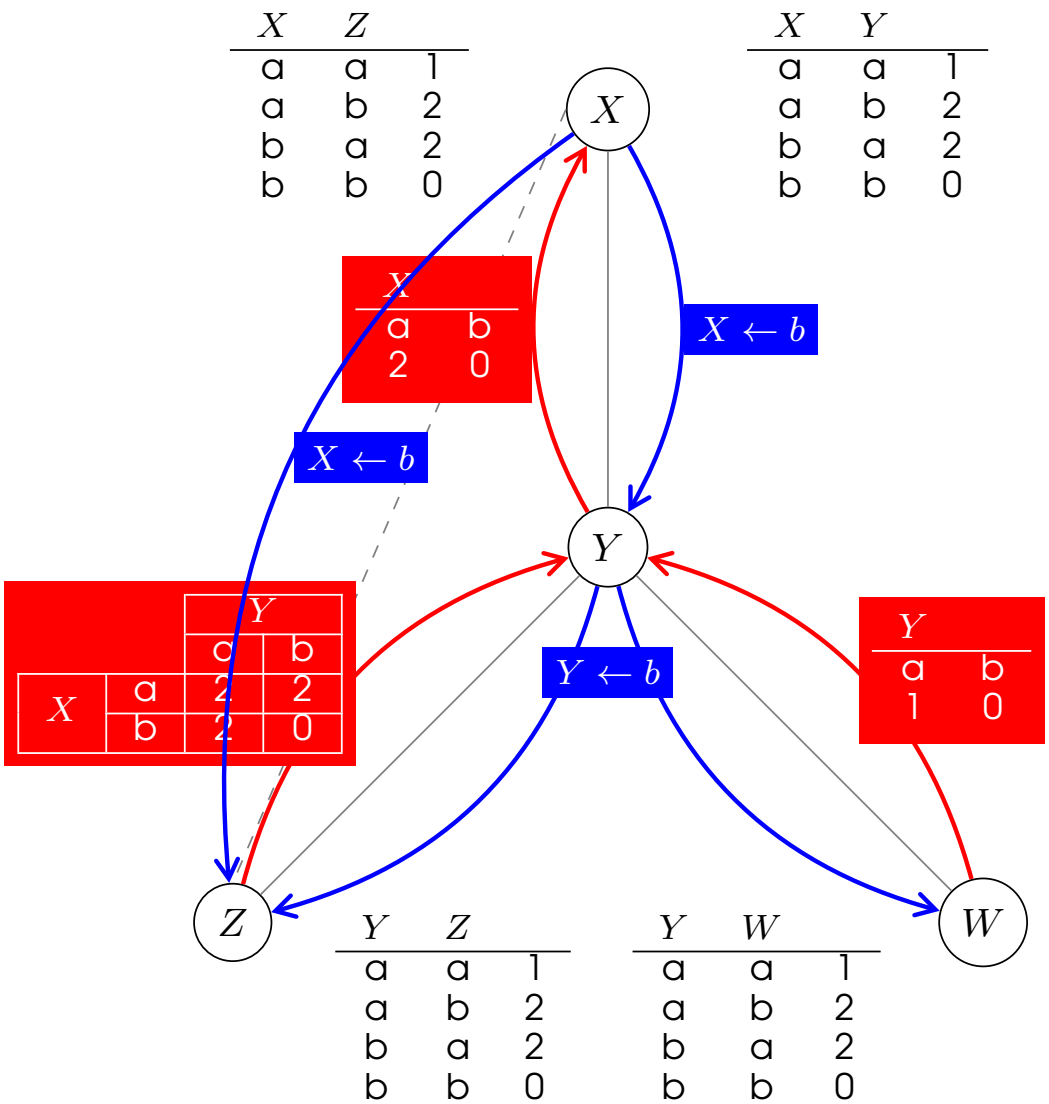
DPOP for any DCOP



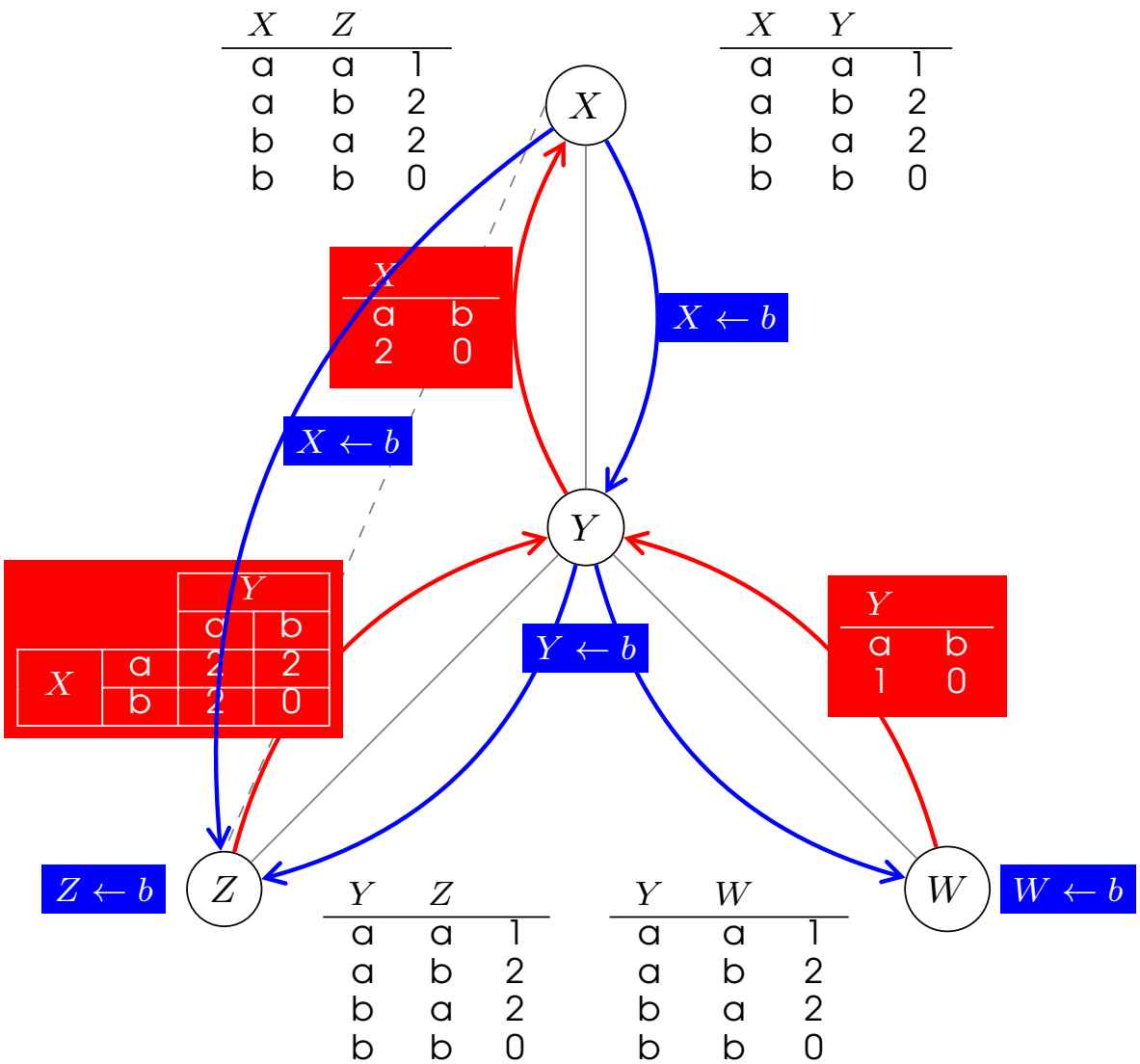
DPOP for any DCOP



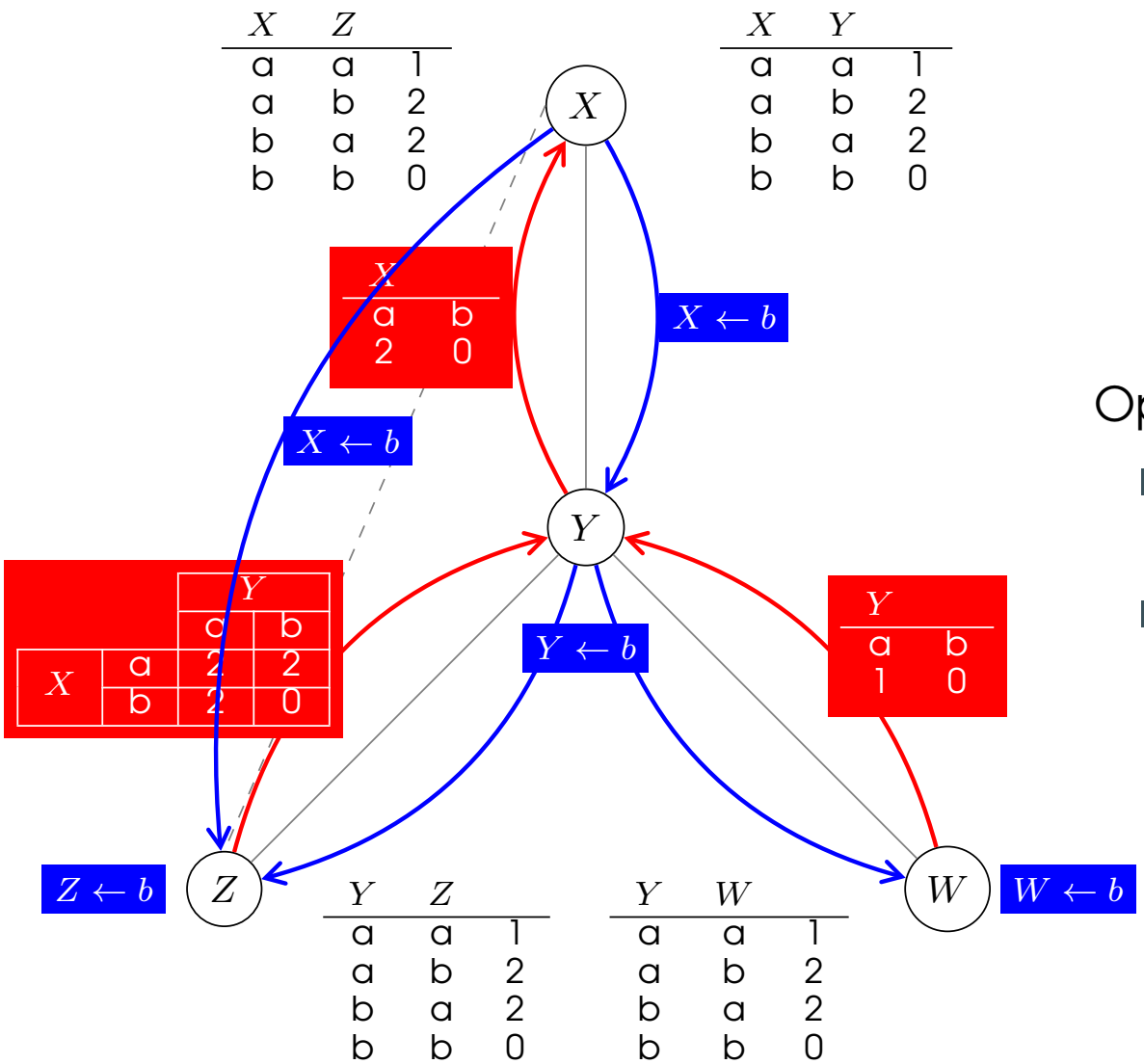
DPOP for any DCOP



DPOP for any DCOP



DPOP for any DCOP



Optimal solution:

- linear number of messages
- message size: exponential

GDL-based approaches

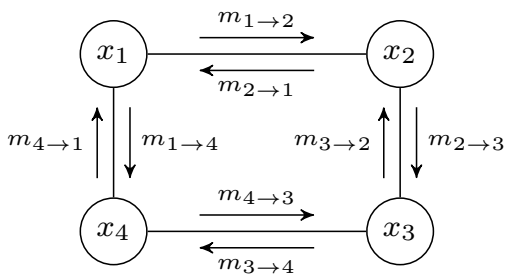
- Generalized Distributive Law (AJI and MCELIECE, 2000)
  - Unifying framework for inference in Graphical models
  - Builds on basic mathematical properties of semi-rings
  - Widely used in Info theory, Statistical physics, Probabilistic models
- Max-sum
  - DCOP settings: maximise social welfare

	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.	$\{0,1\}$	$(\mathrm{OR},0)$	$(\mathrm{AND},1)$	Boolean
10.	2^S	$(\cup,\emptyset)$	$(\cap,S)$	
11.	Λ	$(\vee,0)$	$(\wedge,1)$	
12.	Λ	$(\wedge,1)$	$(\vee,0)$.	

Max-Sum

(FARINELLI et al., 2008)

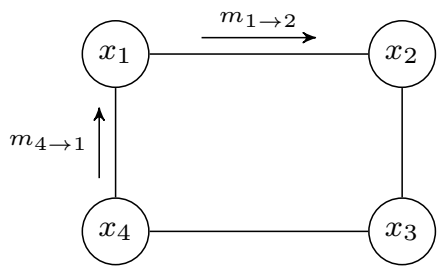
Agents iteratively computes local functions that depend only on the variable they control



Max-Sum

(FARINELLI et al., 2008)

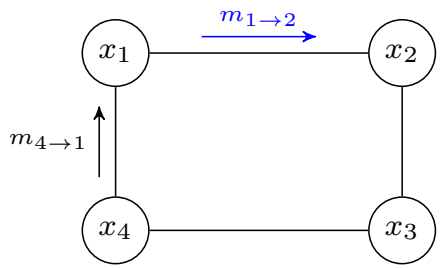
Agents iteratively computes local functions that depend only on the variable they control



Max-Sum

(FARINELLI et al., 2008)

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

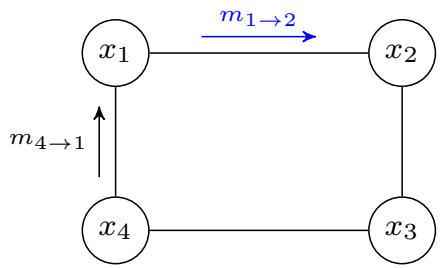


$$m_{1\to2}(x_2) = max_{x_1}(F_{12}(x_1, x_2) + m_{4\to1}(x_1))$$

Max-Sum

(FARINELLI et al., 2008)

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



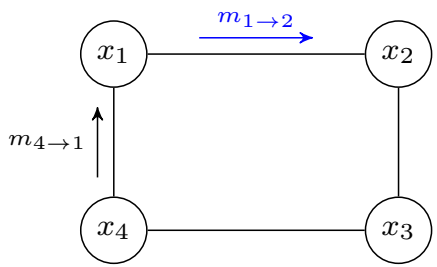
Shared constraint

$$m_{1\to2}(x_2) = max_{x_1}(\quad + m_{4\to1}(x_1))$$

Max-Sum

(FARINELLI et al., 2008)

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



Shared constraint

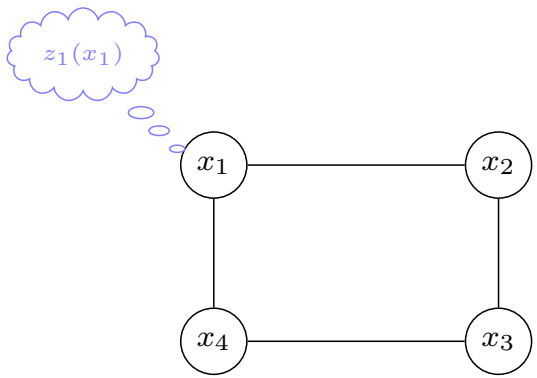
$$m_{1\to2}(x_2) = max_{x_1}(\quad + \quad)$$

All incoming messages except x_2

Max-Sum

(FARINELLI et al., 2008)

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



Shared constraint

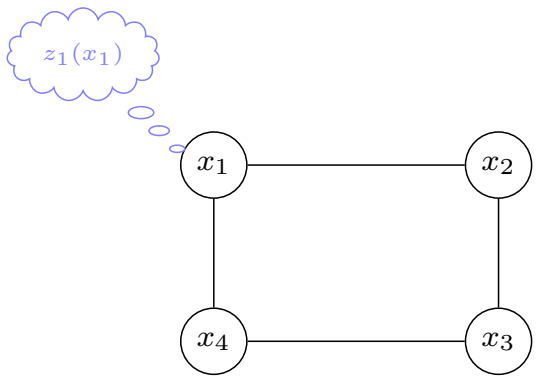
$$m_{1\to 2}(x_2) = max_{x_1}(\quad + \quad)$$

All incoming messages except x_2

Max-Sum

(FARINELLI et al., 2008)

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



Shared constraint

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

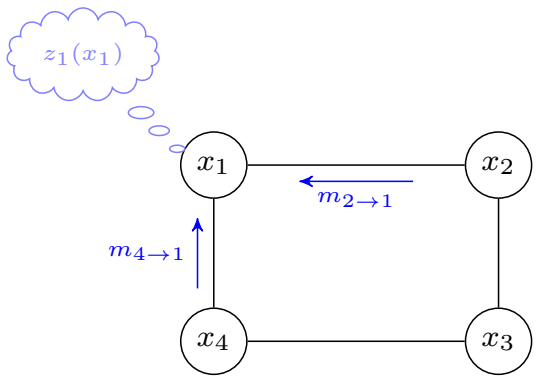
All incoming messages except x_2

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

Max-Sum

(FARINELLI et al., 2008)

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



Shared constraint

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

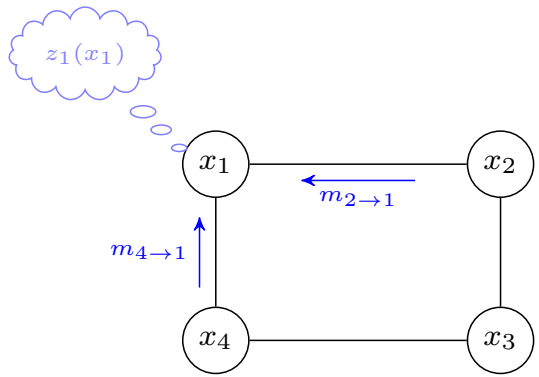
All incoming messages except x_2

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

Max-Sum

(FARINELLI et al., 2008)

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

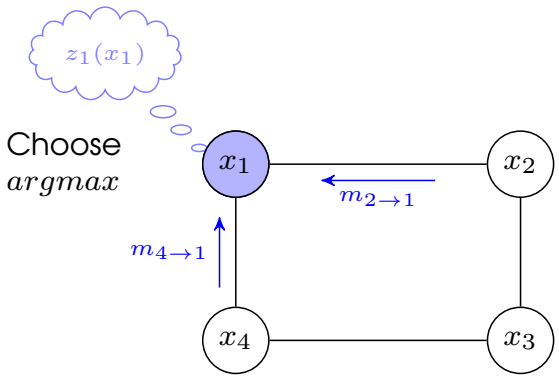
$$z_1(x_1) = \quad$$

All incoming messages

Max-Sum

(FARINELLI et al., 2008)

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



Shared constraint

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

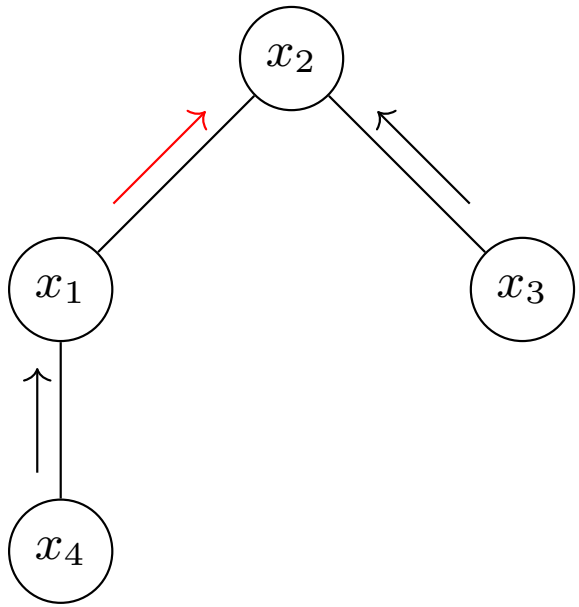
All incoming messages except x_2

$$z_1(x_1) =$$

All incoming messages

Max-Sum on acyclic graphs

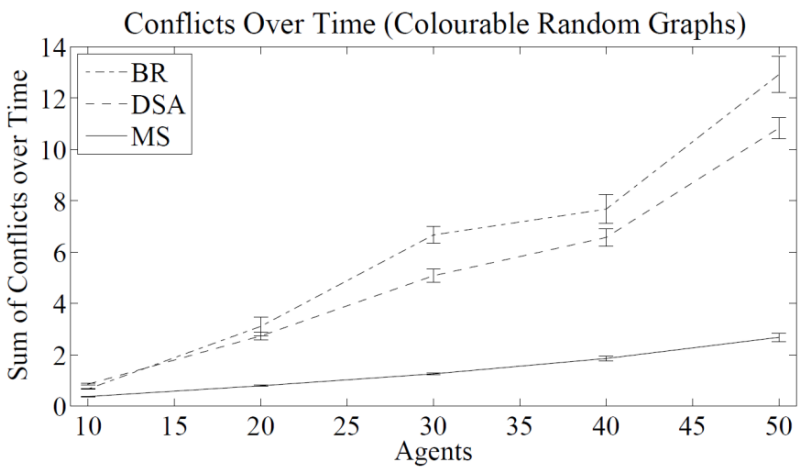
- Max-sum Optimal on acyclic graphs
  - Different branches are independent
  - Each agent can build a correct estimation of its contribution to the global problem (z functions)
- Message equations very similar to Util messages in DPOP
  - Sum messages from children and shared constraint
  - Maximize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

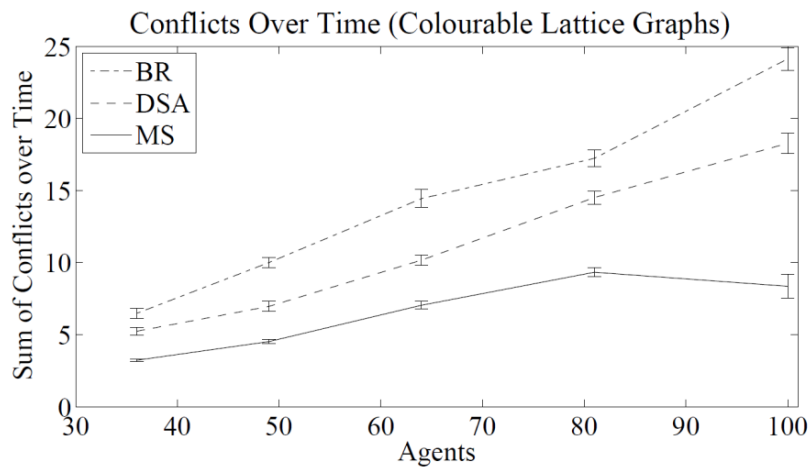


$$m_{1\to 2}(x_2) = max_{x_1}(F_{12}(x_1, x_2) + m_{4\to 1}(x_1))$$

Max-Sum Performance

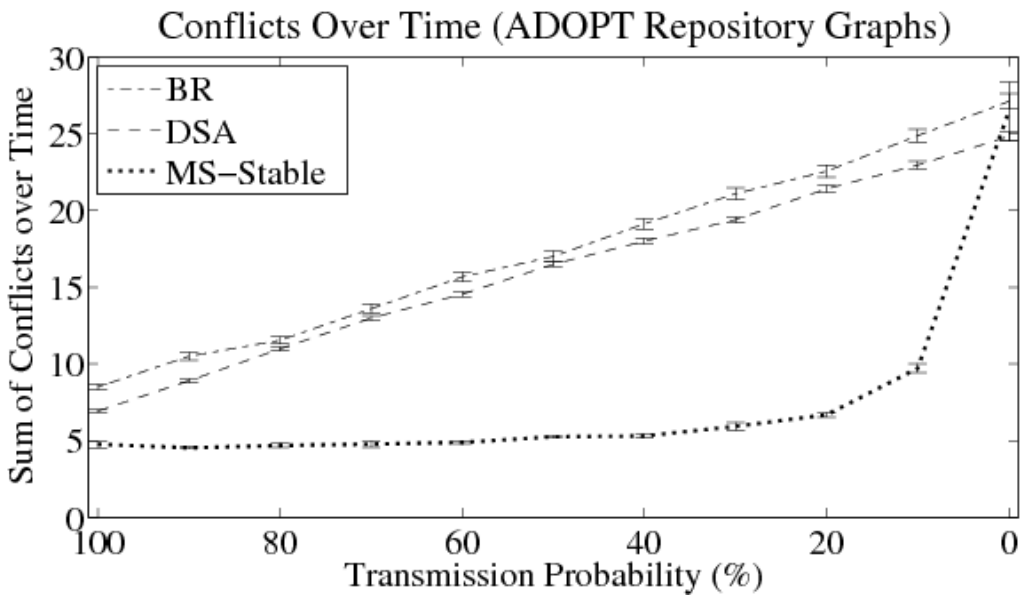
- Good performance on loopy networks (FARINELLI et al., 2008)
  - When it converges very good results
    - Interesting results when only one cycle (WEISS, 2000)
  - We could remove cycle but pay an exponential price (see DPOP)





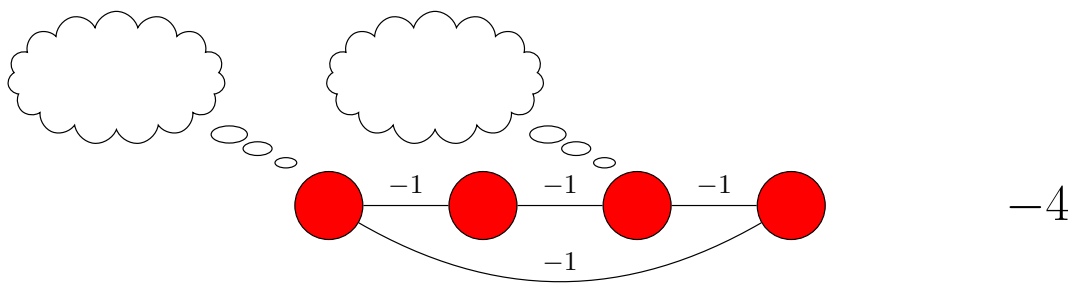
Max-Sum for low power devices

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



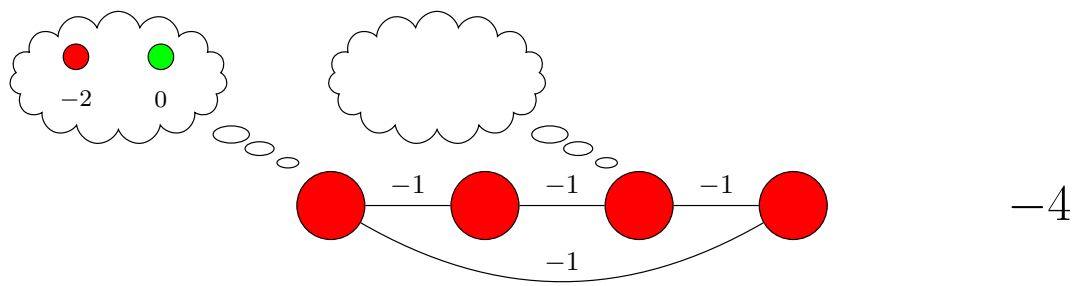
Local Greedy Approaches

- 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



Local Greedy Approaches

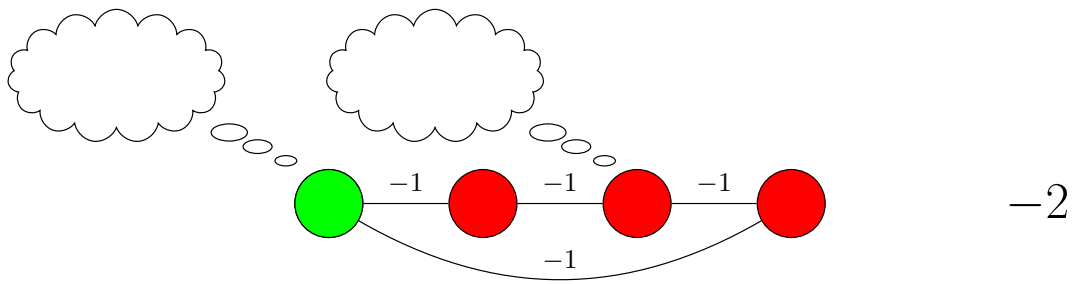
- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one



-4

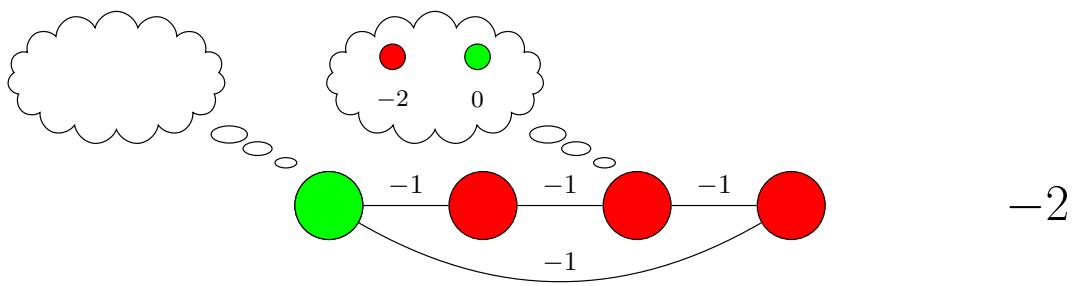
Local Greedy Approaches

- 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



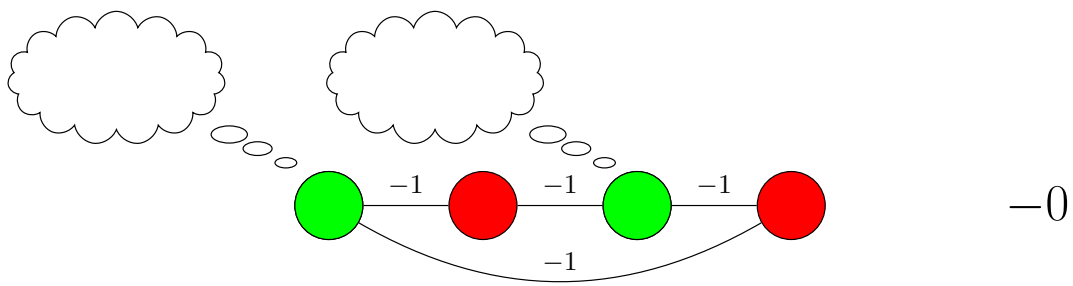
Local Greedy Approaches

- 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



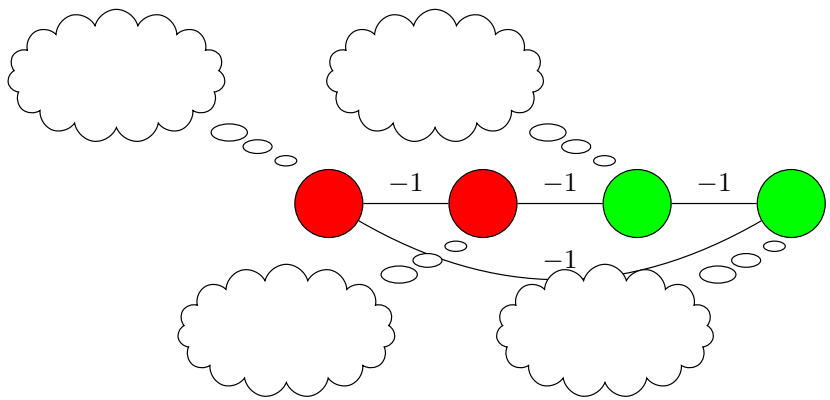
Local Greedy Approaches

- 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



Local Greedy Approaches

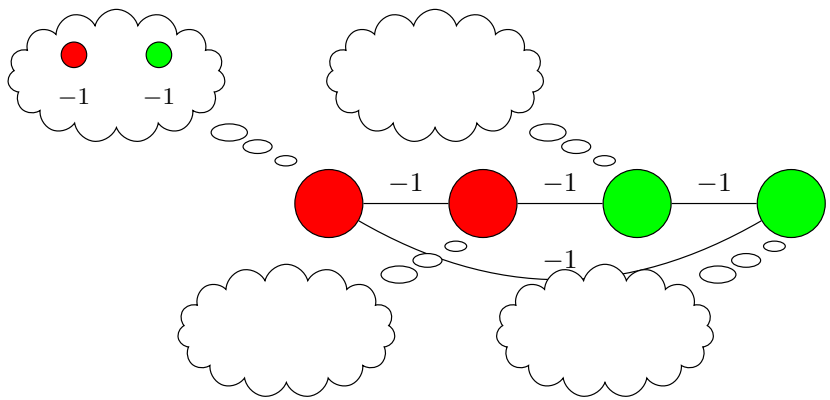
- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing



-2

Local Greedy Approaches

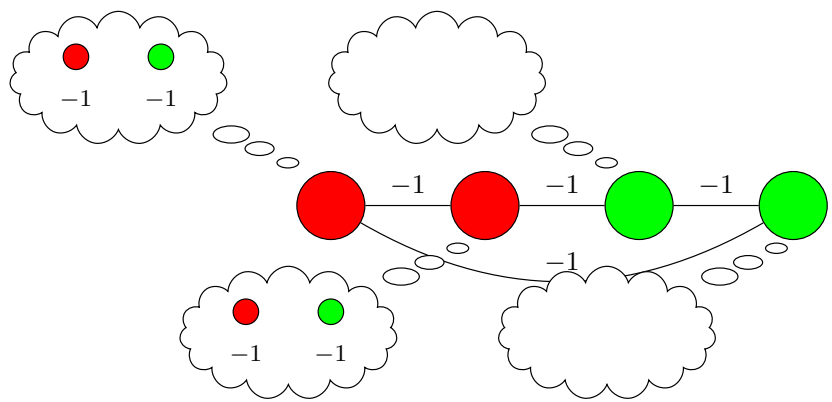
- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing



-2

Local Greedy Approaches

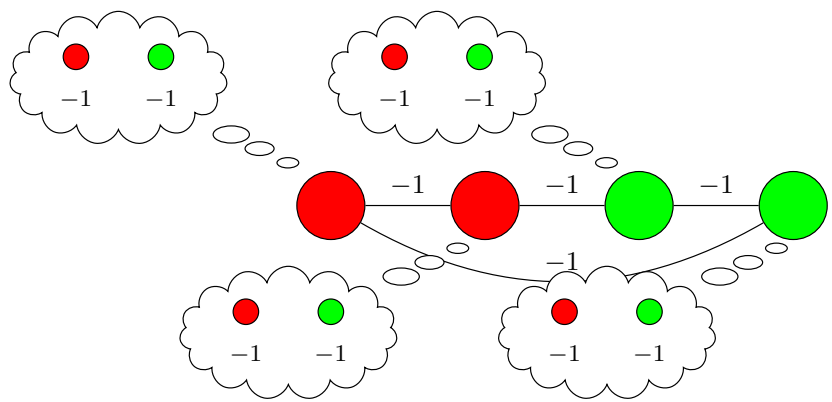
- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing



-2

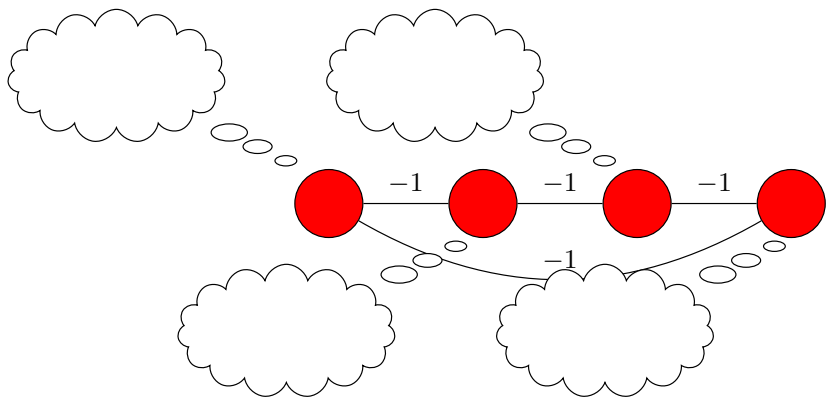
Local Greedy Approaches

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



Distributed Local Greedy approaches

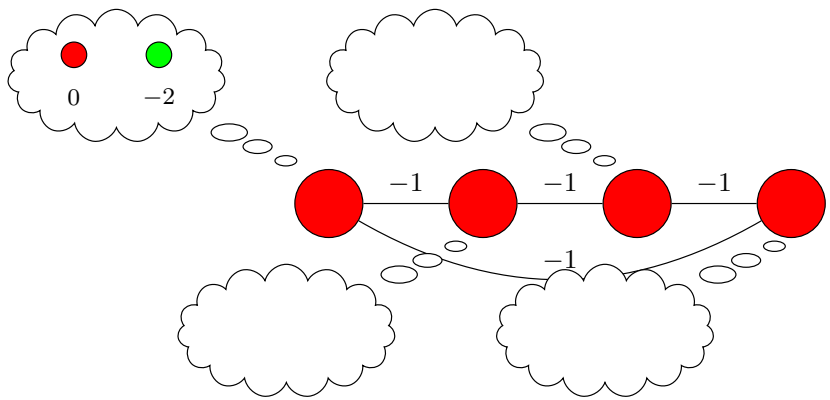
- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination



-4

Distributed Local Greedy approaches

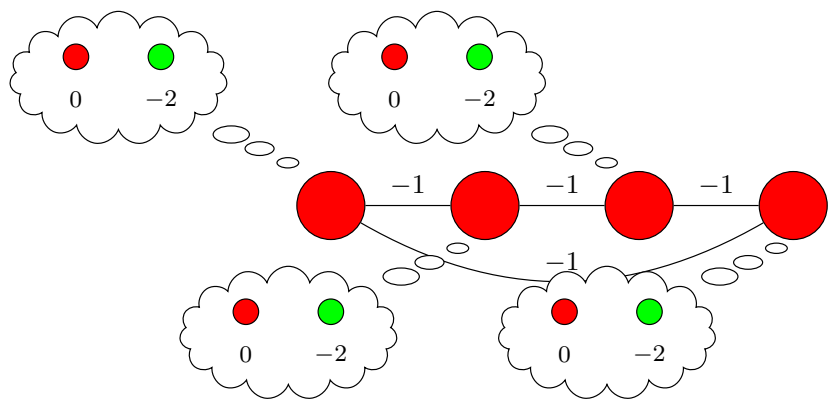
- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination



-4

Distributed Local Greedy approaches

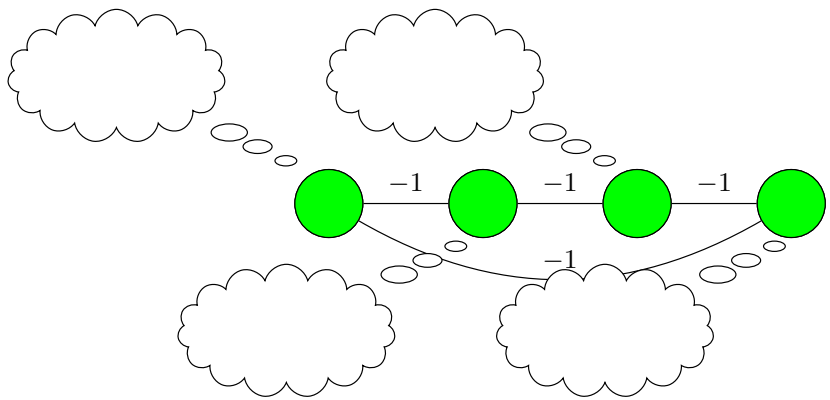
- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination



-4

Distributed Local Greedy approaches

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



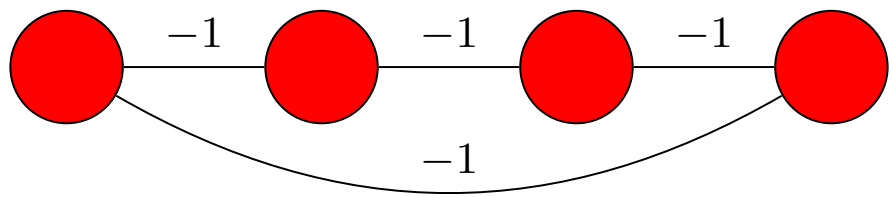
-4

Distributed Stochastic Search Algorithm (DSA)

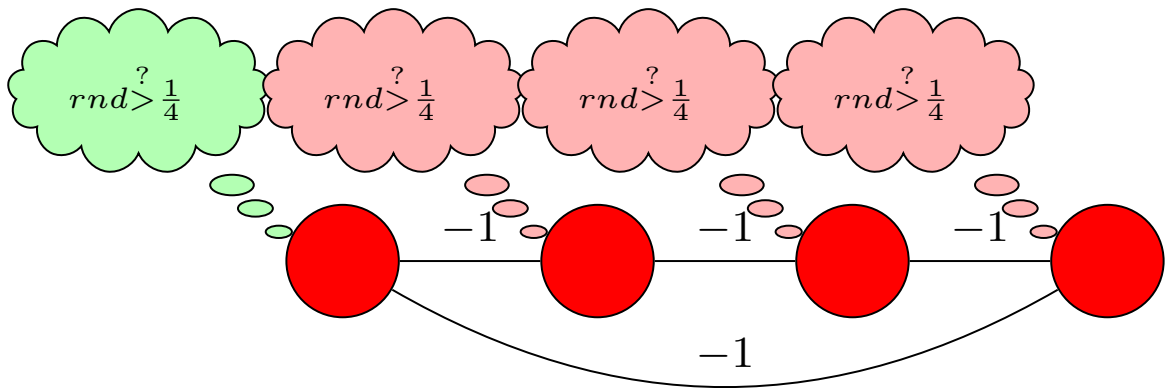
(ZHANG et al., 2005)

- 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

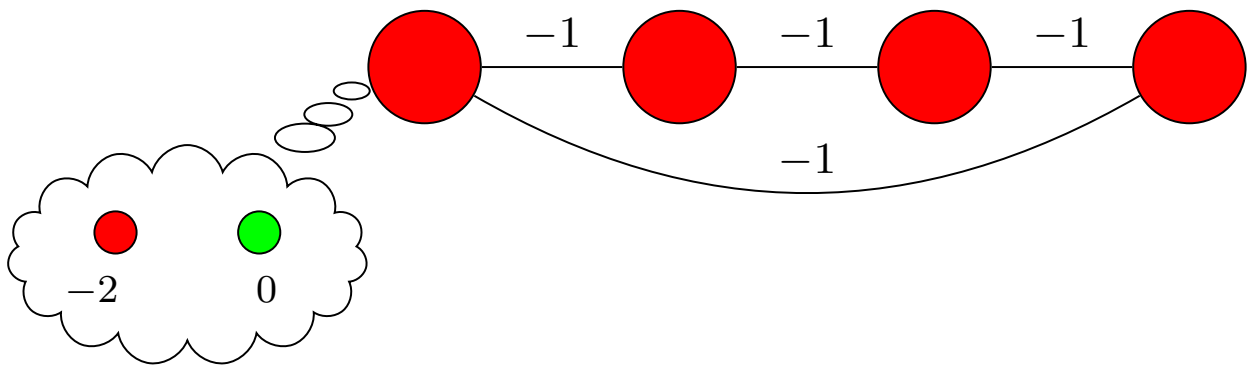
DSA-1: Execution Example



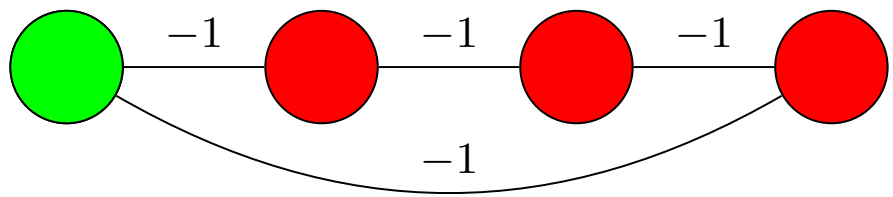
DSA-1: Execution Example



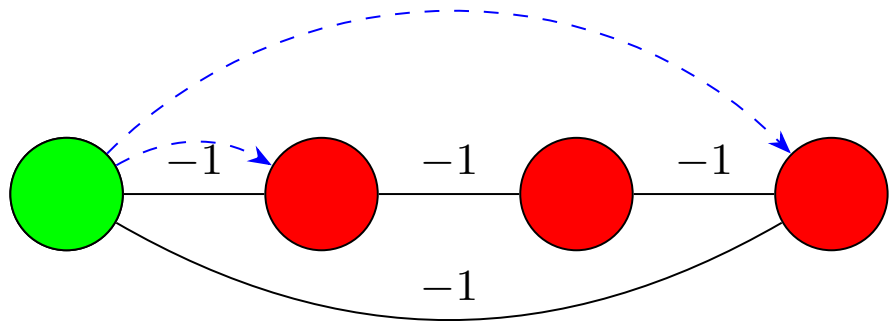
DSA-1: Execution Example



DSA-1: Execution Example



DSA-1: Execution Example



DSA-1: Discussion

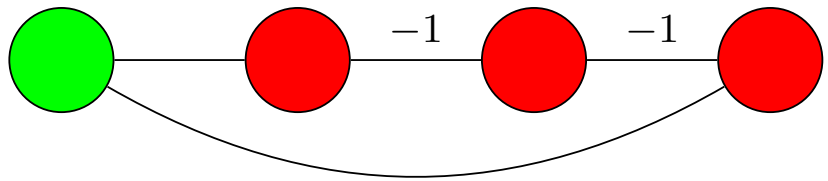
- Extremely "cheap" (computation/communication)
- Good performance in various domains
  - e.g. target tracking (Fitzpatrick and Meertens, 2003; Zhang et al., 2003)
  - Shows an anytime property (not guaranteed)
  - Benchmarking technique for coordination
- Problems
  - Activation probablity must be tuned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

Maximum Gain Message (MGM-1)

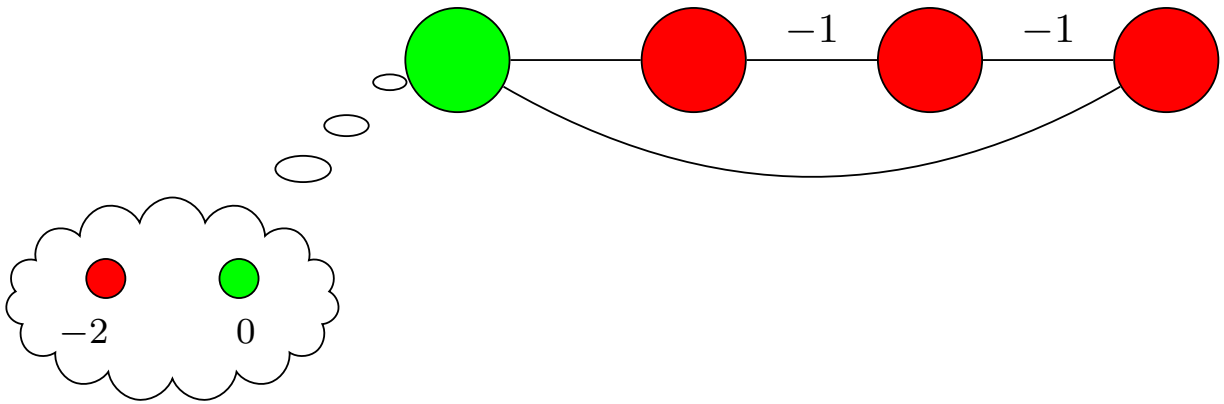
(MAHESWARAN et al., 2004)

- 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)

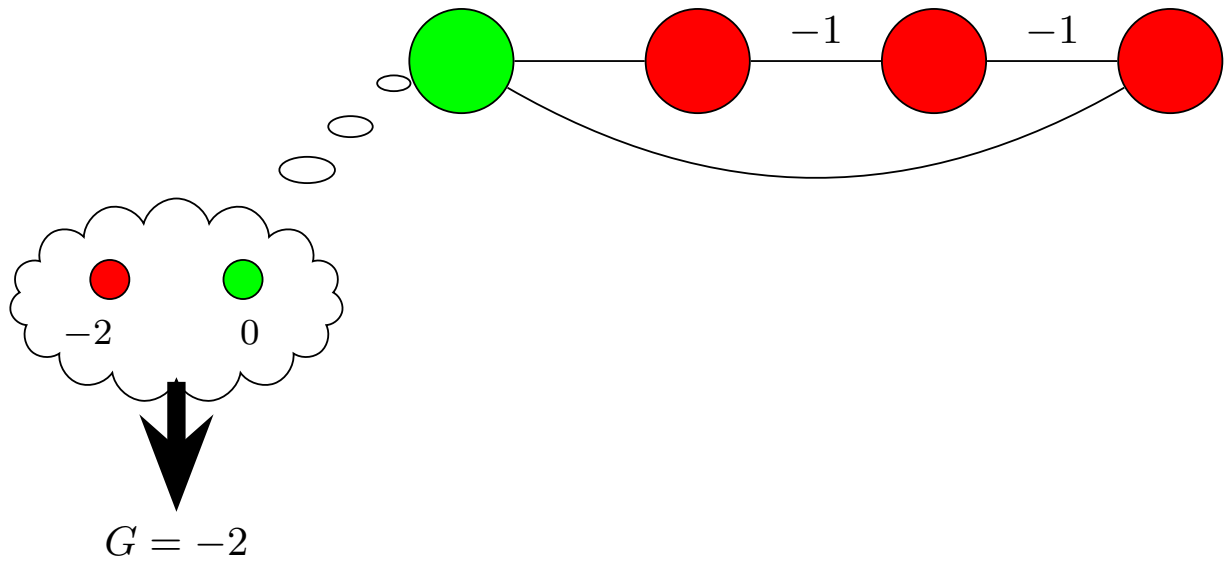
MGM-1: Example



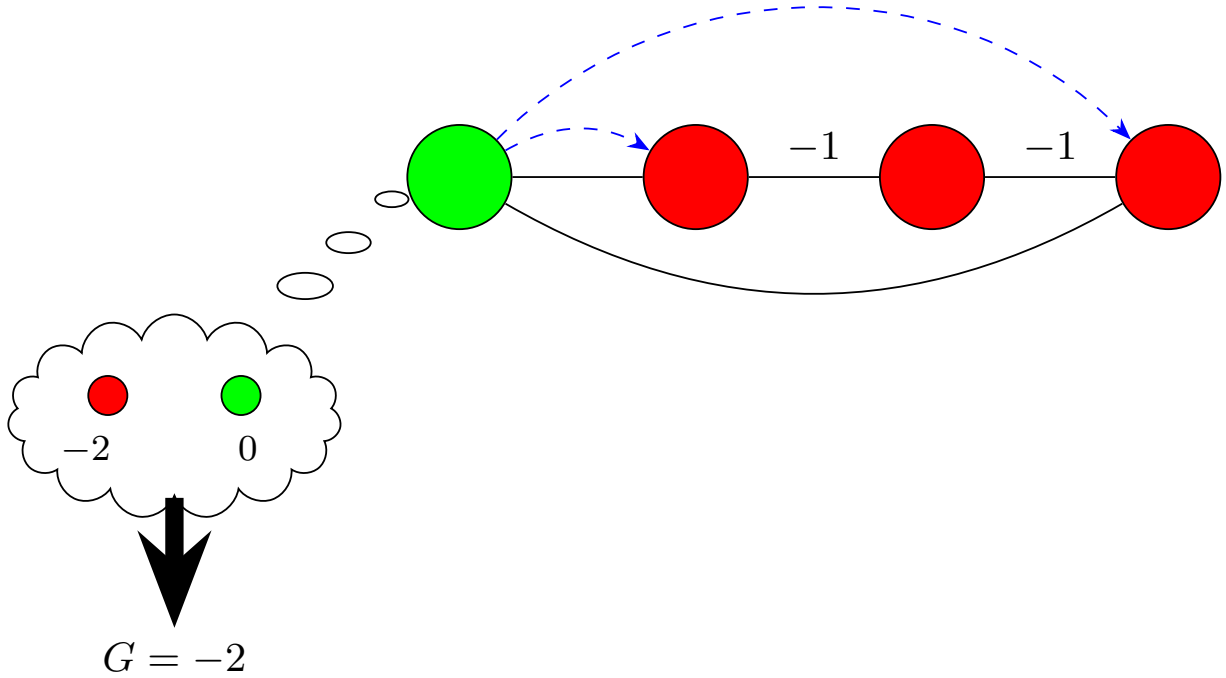
MGM-1: Example



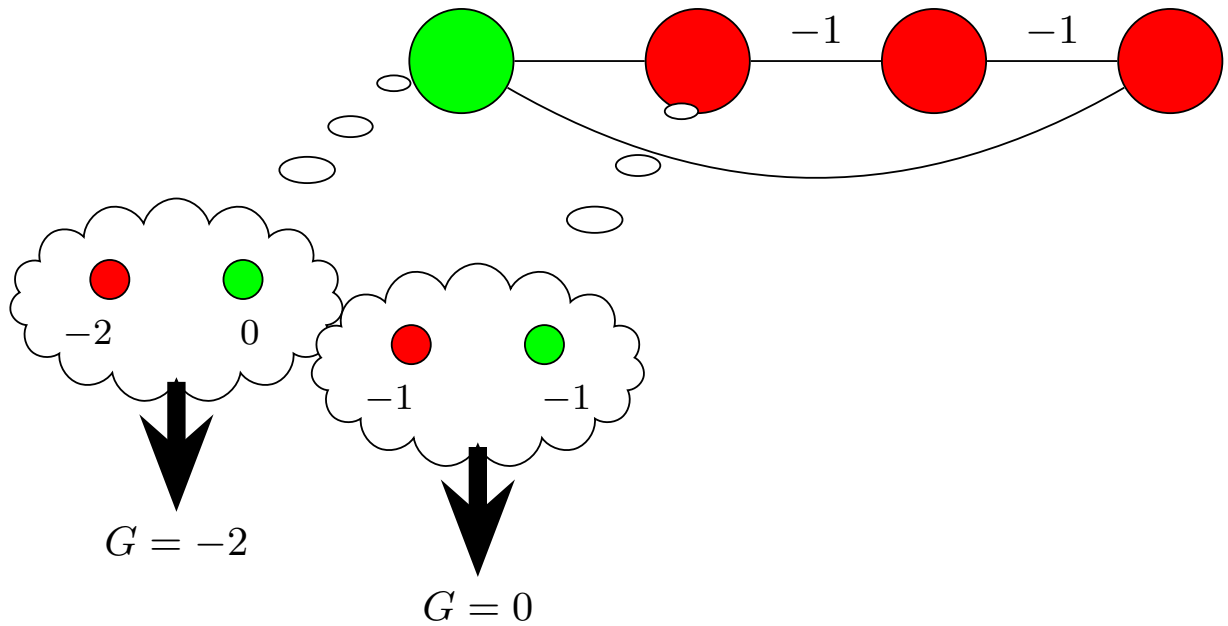
MGM-1: Example



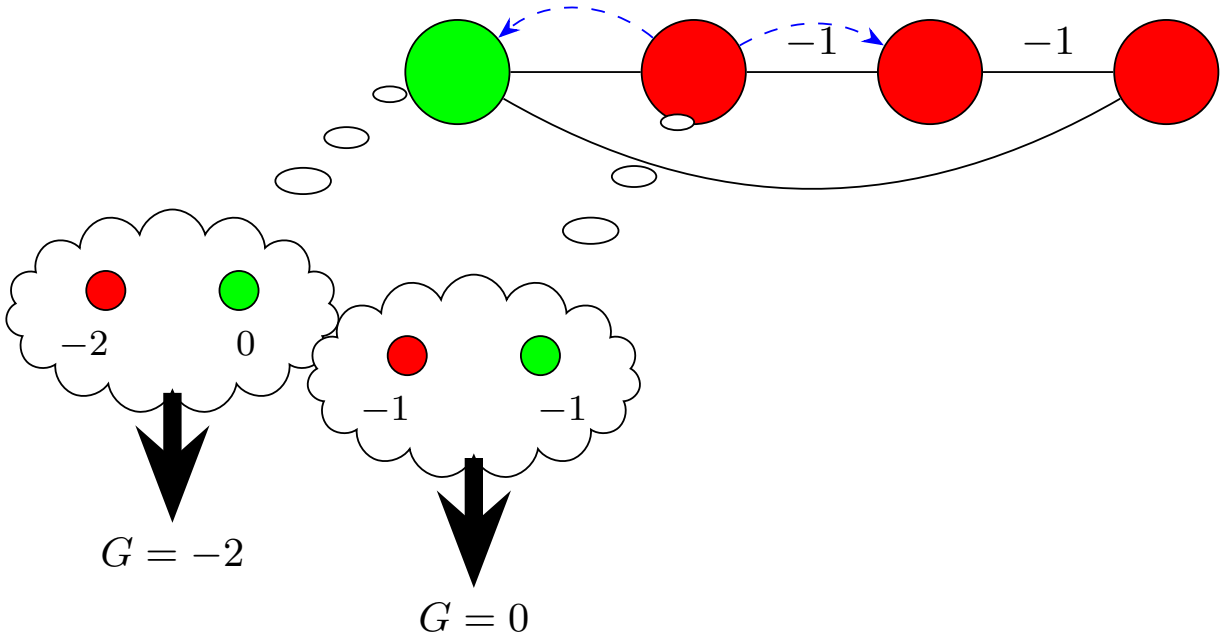
MGM-1: Example



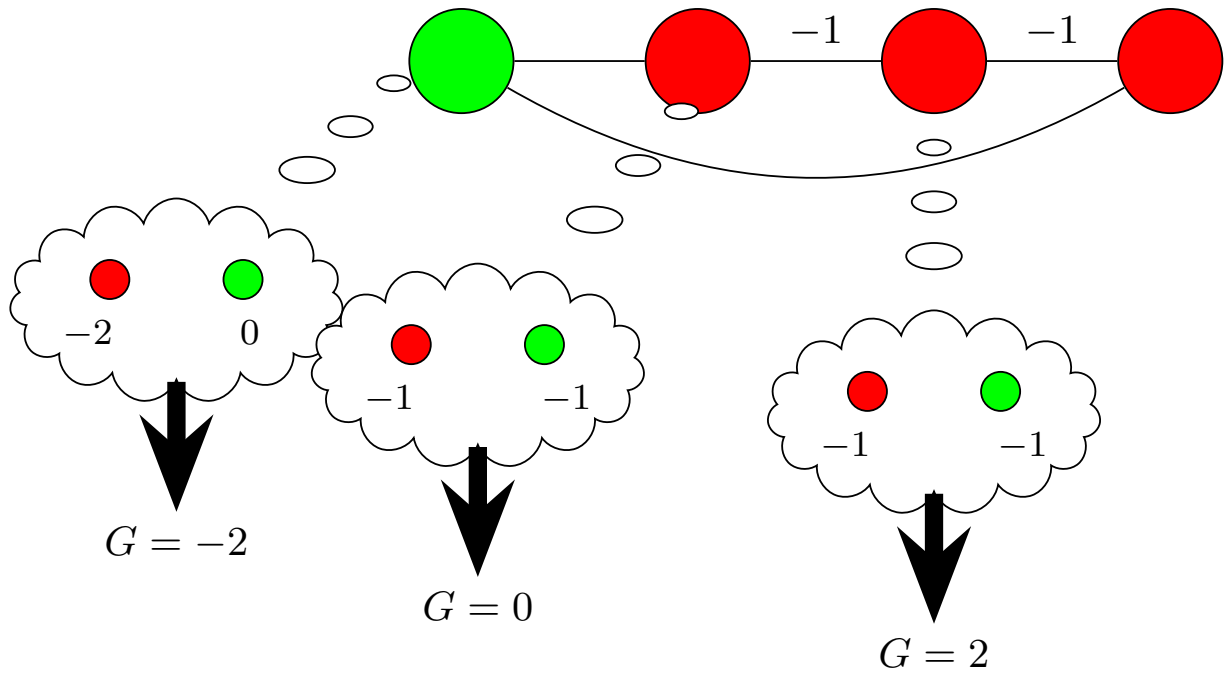
MGM-1: Example



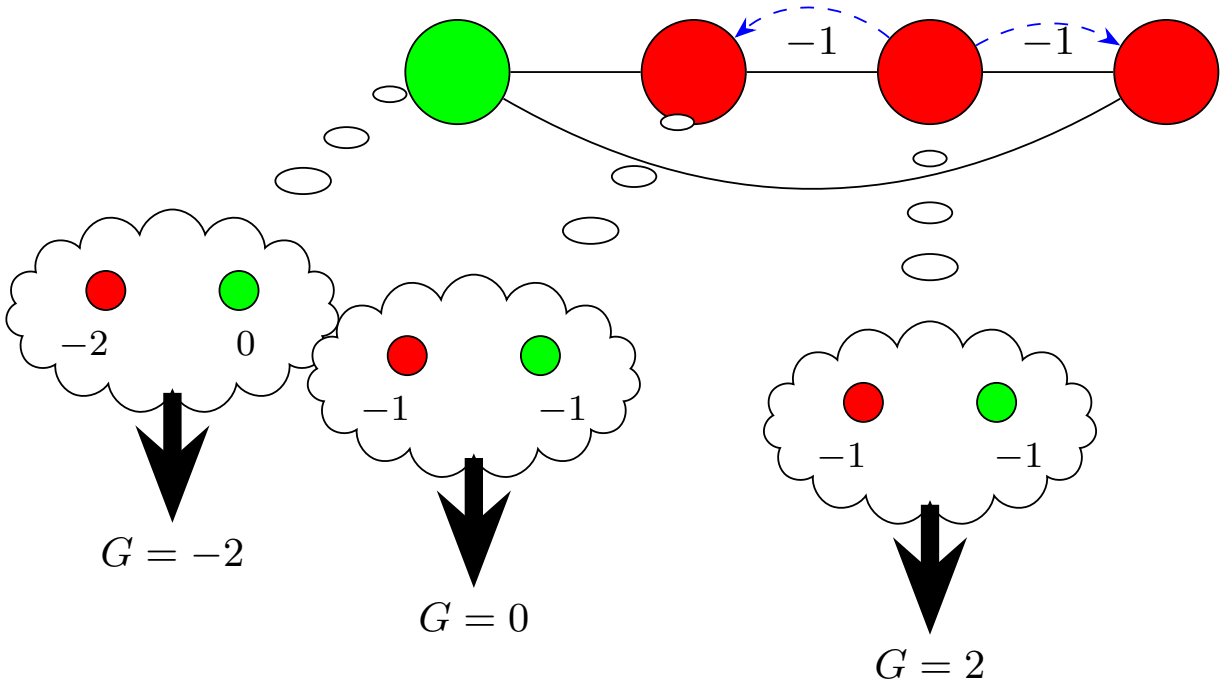
MGM-1: Example



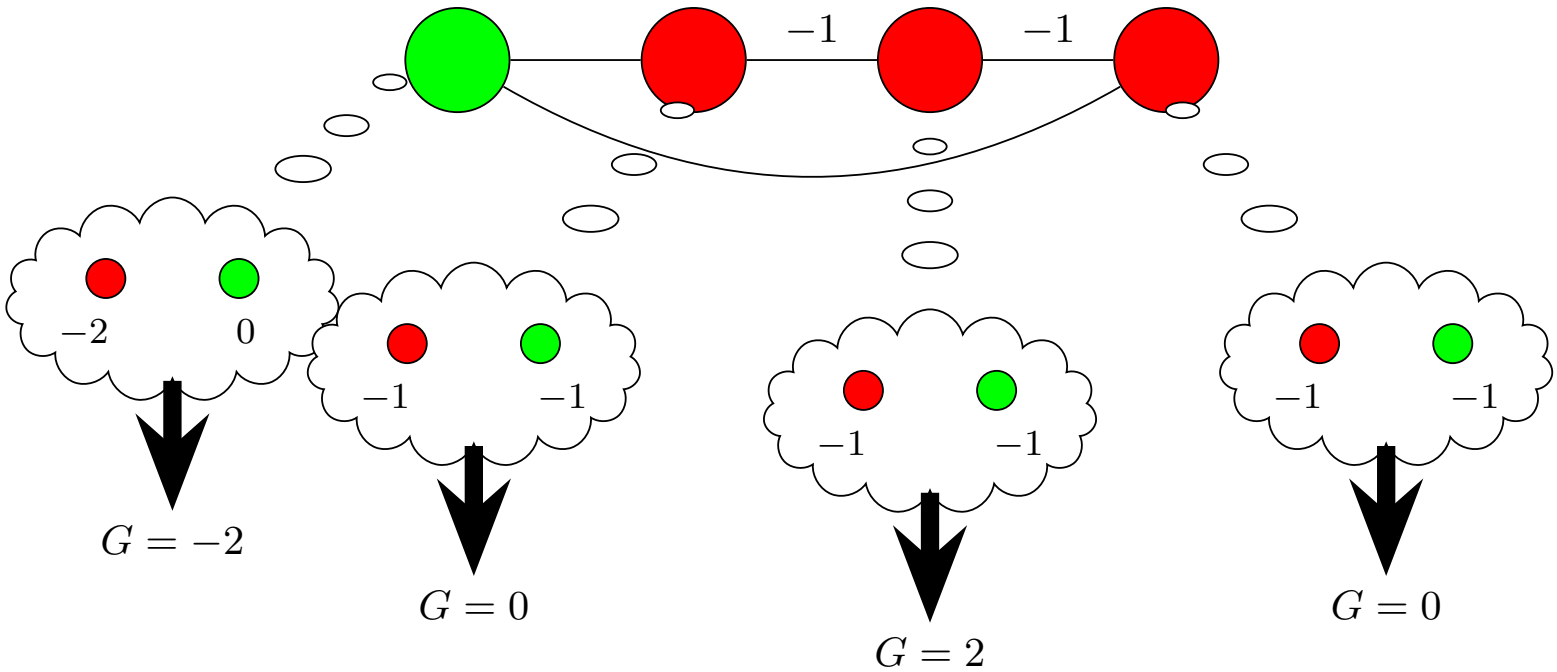
MGM-1: Example



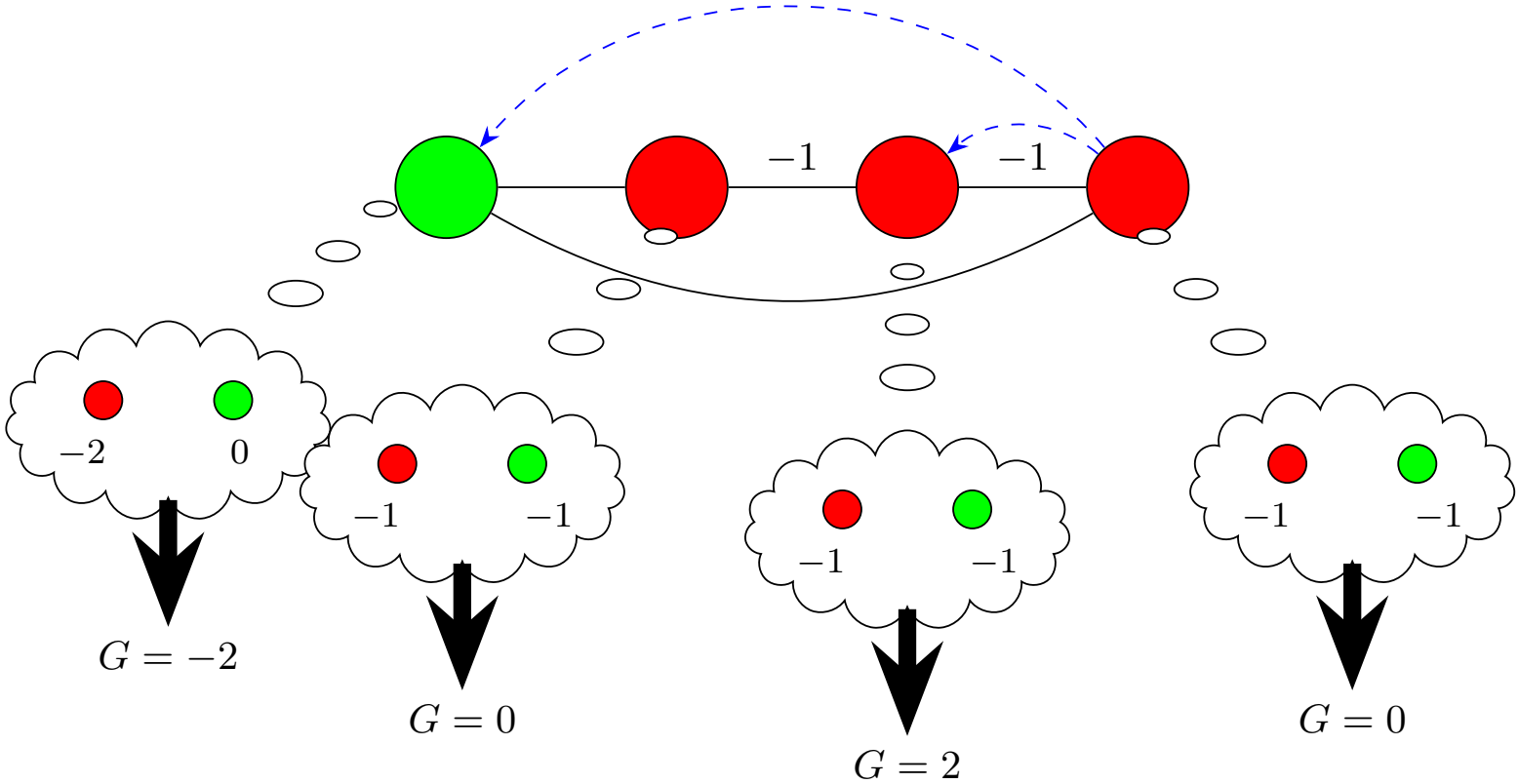
MGM-1: Example



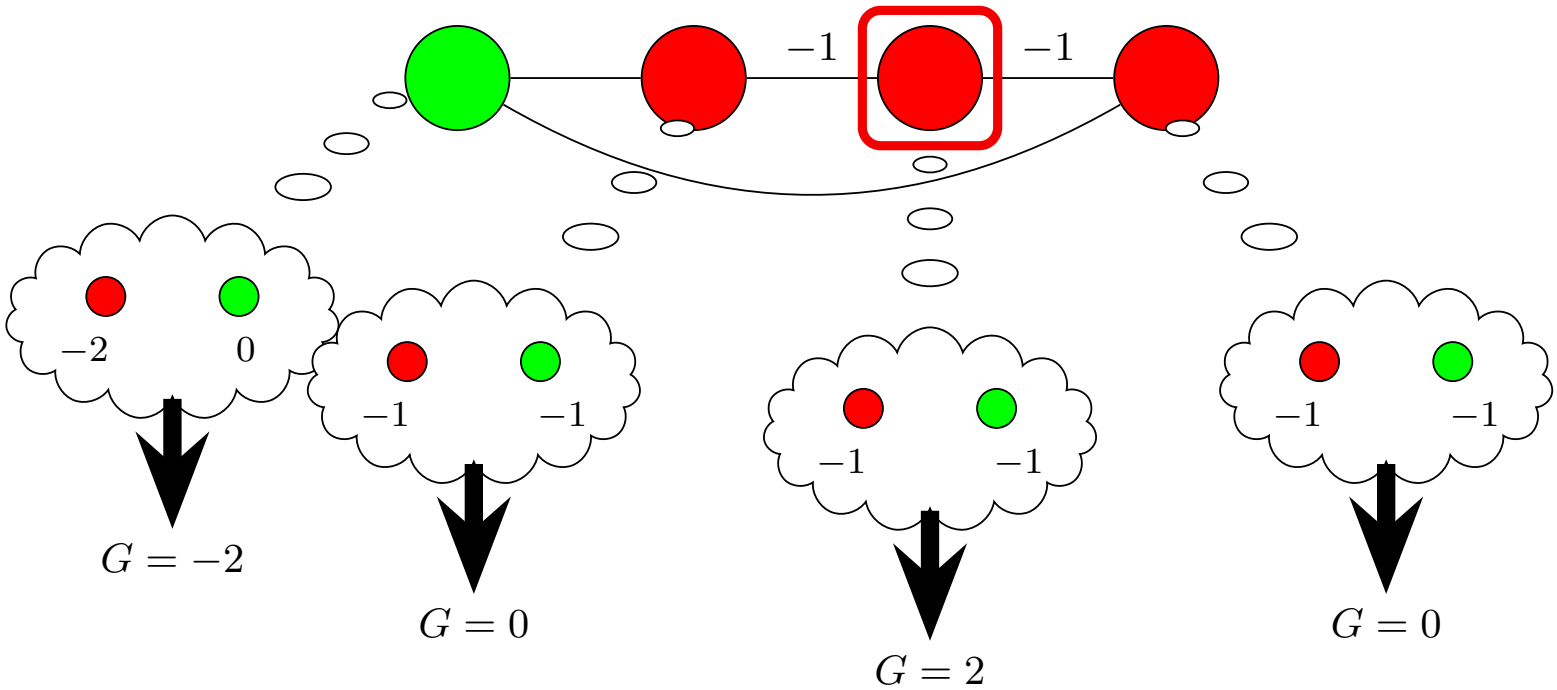
MGM-1: Example



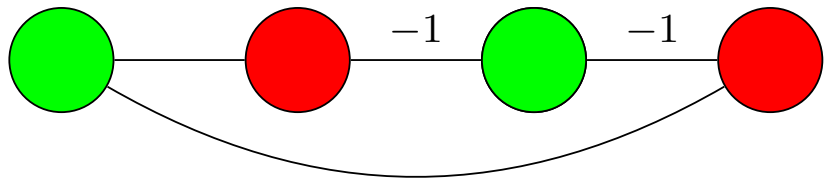
MGM-1: Example



MGM-1: Example



MGM-1: Example



Menu

Hands on PyDCOP I

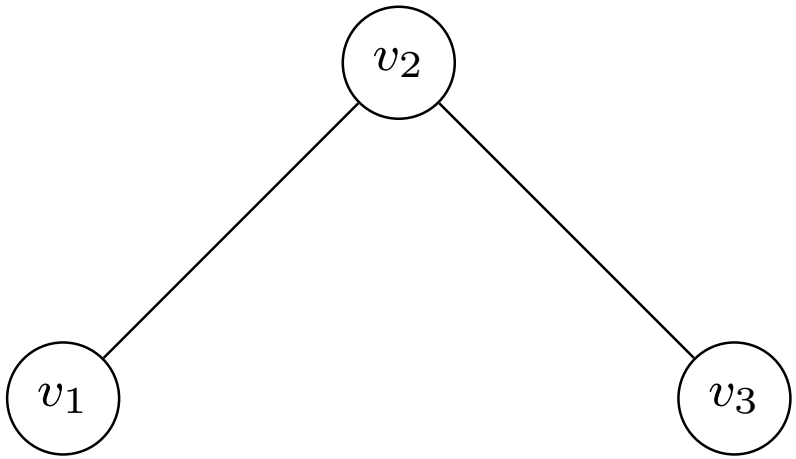
- Install VirtualBox
- Import the pyDCOP Virtual Machine (`https://bit.ly/2JUrrKP`)
  - It's a Debian image with everything preinstalled:
  - python3, pyDCOP, matplotlib, glpk, etc.
- Alternatively, follow

  `https://pydcop.readthedocs.io/en/latest/installation.html`

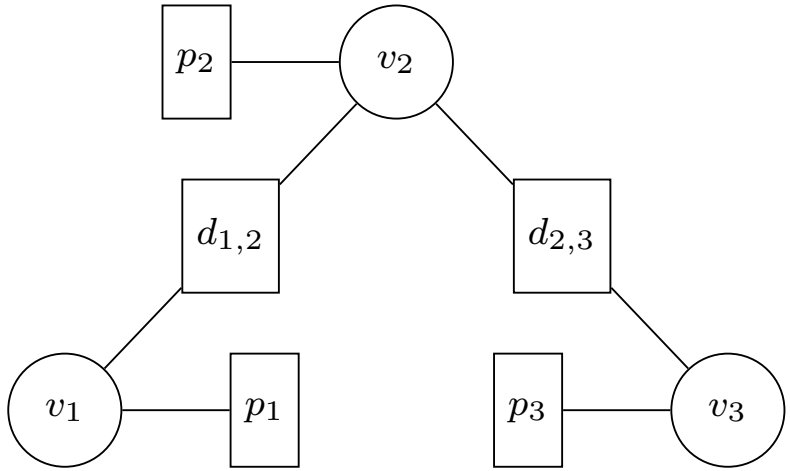
1. `https://pydcop.readthedocs.io/en/latest/tutorials/getting_started.html`
2. `https://pydcop.readthedocs.io/en/latest/tutorials/analysing_results.html`

Hands on PyDCOP I

DCOP - Graph Coloring



(a) constraints graph



(b) factor graph

- **Objective:** minimize
- **Domain:** 2 colors R and B
- **Variables:** V_1, V_2, V_3
- **Constraints:** neighbors must have different colors + preferences
- **Agents:** 3 agents

Yaml representation

Hands on PyDCOP I

Solving the Graph Coloring DCOP

Command:

```
$ pydcop solve --algo dpop graph_coloring.yaml
```

Output:

```
...
"assignment": {
  "v1": "R",
  "v2": "G",
  "v3": "R"
},
"cost": -0.1,
...
```

With other algorithms:

```
$ pydcop --timeout 2 solve --algo dsa graph_coloring.yaml
$ pydcop solve --algo mgm --algo_params stop_cycle:20 \
       graph_coloring.yaml
```

Hands on PyDCOP I

Results

Full results :


```
{
  "agt_metrics": {
    ...
  },
  "assignment": {
    "v1": "R",
    "v2": "G",
    "v3": "R"
  },
  "cost": -0.1,
  "cycle": 20,
  "msg_count": 158,
  "msg_size": 158,
  "status": "FINISHED",
  "time": 0.032010296999942 51,
  "violation": 0
}
```

Hands on PyDCOP I

Logs

Simple:

use `-v 0..3`

```
$ pydcop -v 2 solve --algo dpop graph_coloring.yaml
```

Precise :

use `-log <log.conf>`

```
$ pydcop --log log.conf -t 1 solve --algo dsa graph_coloring.yaml
```

Now, look at `algo.log`

Hands on PyDCOP I

Run-time metrics

periodic: "--collect_on period --period <p>"

```
$ pydcop --log log.conf -t 10 solve \
   --collect_on period --period <p>
   --algo dsa graph_coloring.yaml
```

cycle: "--collect_on cycle_change"

Only supported with synchronous algorithms !

```
$ pydcop solve --algo mgm --algo_params stop_cycle:20 \
    --collect_on cycle_change --run_metric ./metrics.csv \
    graph_coloring_50.yaml
```

value: "--collect_on value_change"

```
$ pydcop -t 5 solve --algo mgm --collect_on value_change \
    --run_metric ./metrics_on_value.csv \
    graph_coloring_50.yaml
```

Hands on PyDCOP I

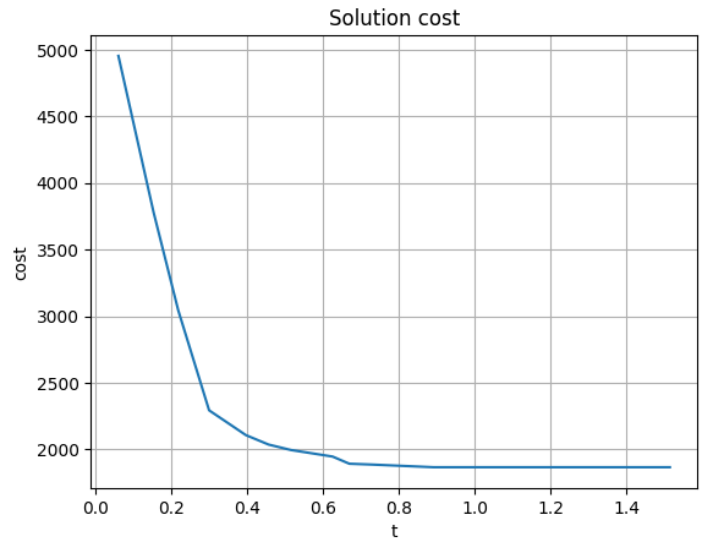
Run-time metrics

With a bigger graph coloring problem

```
$ pydcop solve --algo mgm --algo_params stop_cycle:20 \
           --collect_on cycle_change \
           --run_metric ./metrics.csv \
           graph_coloring_50.yaml
```

Plotting with matplotlib

```
$ python3 plot_cost.py ./metrics.csv
```



Menu

DCOP Framework

Focus on Some Solution Methods

Hands on PyDCOP I

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP II

Dynamic DCOPs

Conclusion

SECP model

Smart Environment Configuration Problem **(RUST et al., 2016)**

- Example of applying DCOPs to a "real" problem
- Coordinate objects in the building
- Model
  - objects
  - relations between objects and environment
  - user objectives and requirements
- Formulate the problem as an optimization problem

SECP model

Smart Environment Configuration Problem **(RUST et al., 2016)**

Focus on smart lighting use cases

- **Objects:** anything that can produce light: light bulbs, windows with rolling shutter, etc.
- **User preferences:** having a predefined luminosity level in a room, under some conditions
- **Energy efficiency**

Linking objects and user preferences:

- How to model the luminosity in a room ? **variable**
- How to model the dependency between the light sources and the luminosity ? **function / constraint**

SECP model

Example application to ambient intelligence scenario

- **Actuators**
  - Connected light bulbs, TV, Rolling shutters, ...
- **Sensors**
  - Presence detector, Luminosity Sensor, etc.
- **Physical Dependency Models**
  - E.g. Living-room light model
- **User Preferences**
  - Expressed as rules :

IF	presence_living_room	$=$	1
AND	light_sensor_living_room	$<$	60
THEN	light_level_living_room	$\leftarrow$	60
AND	shutter_living_room	$\leftarrow$	0

SECP model

Example application to ambient intelligence scenario

- **Actuators**
  - *Decision* variable x_i, domain $\mathcal{D}_{x_i}$
  - Cost function $c_i : \mathcal{D}_{x_i} \to \mathbb{R}$
- **Sensors**
  - *Read-only* variable s_l, domain $\mathcal{D}_{s_l}$
- **Physical Dependency Models** $\langle y_j, \phi_j \rangle$
  - Give the expected state of the environment from a set of actuator-variables influencing this model
  - Variable y_j representing the *expected* state of the environment
  - Function $\phi_j : \prod_{\varsigma \in \sigma(\phi_j)} \mathcal{D}_\varsigma \to \mathcal{D}_{y_j}$
- **User Preferences**
  - Utility function u_k
  - Distance from the current expected state to the target state of the environment

Formulating SECP as a DCOP

Multi-objective optimization problem

$$\min_{x_i \in \nu(\mathfrak{A})} \sum_{i \in \mathfrak{A}} c_i \quad \text{and} \quad \max_{\substack{x_i \in \nu(\mathfrak{A}) \\ y_j \in \nu(\Phi)}} \sum_{k \in \mathfrak{R}} u_k$$

$$\textbf{s.t.} \;\; \phi_j(x_j^1, \ldots, x_j^{\overline{\phi_j}}) = y_j \quad \forall y_j \in \nu(\Phi)$$

Then mono-objective DCOP formulation

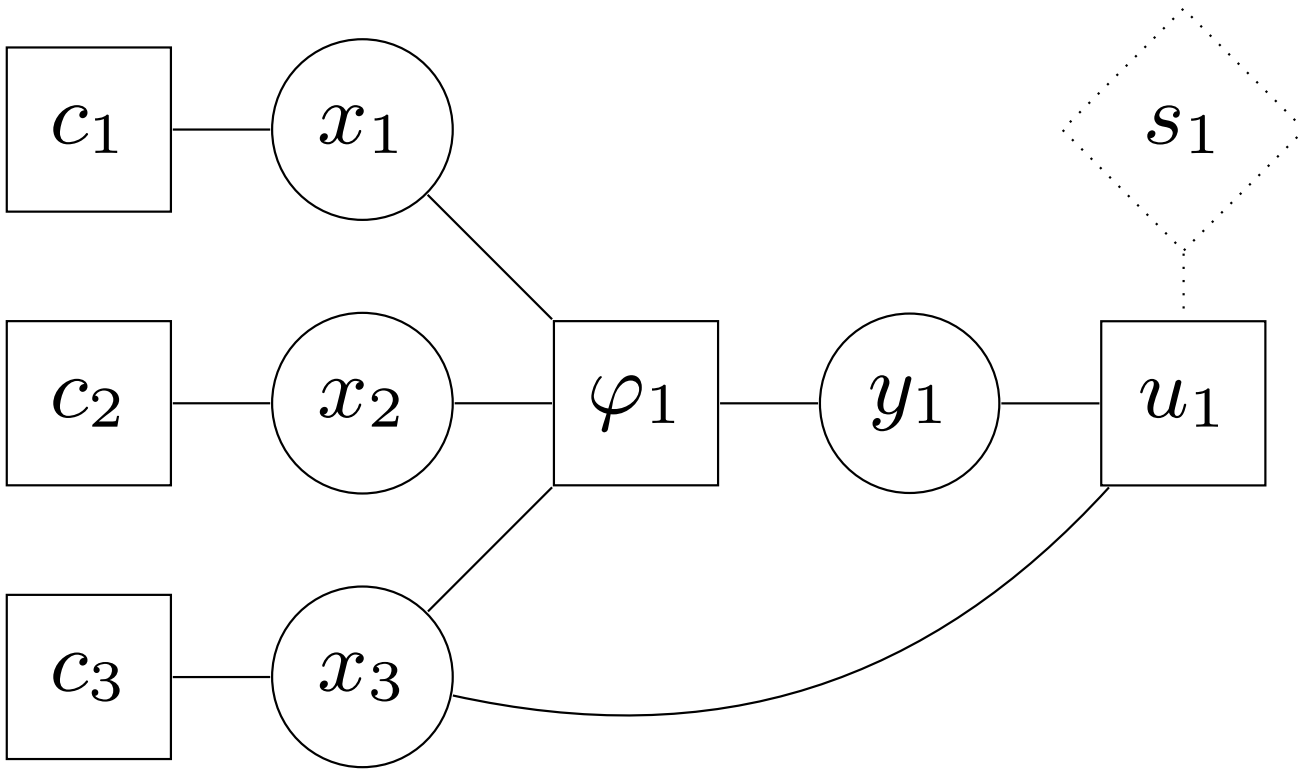
$$\max_{\substack{x_i \in \nu(\mathfrak{A}) \\ y_j \in \nu(\Phi)}} \omega_u \sum_{k \in \mathfrak{R}} u_k - \omega_c \sum_{i \in \mathfrak{A}} c_i + \sum_{\varphi_j \in \mathbf{\Phi}} \varphi_j$$

with reformulation of hard constraints ϕ_j into soft ones:

$$\varphi_j(x_j^1, \ldots, x_j^{|\sigma(\phi_j)|}, y_j) = \begin{cases} 0 & \text{if } \phi_j(x_j^1, \ldots, x_j^{|\sigma(\phi_j)|}) = y_j \\ -\infty & \text{otherwise} \end{cases}$$

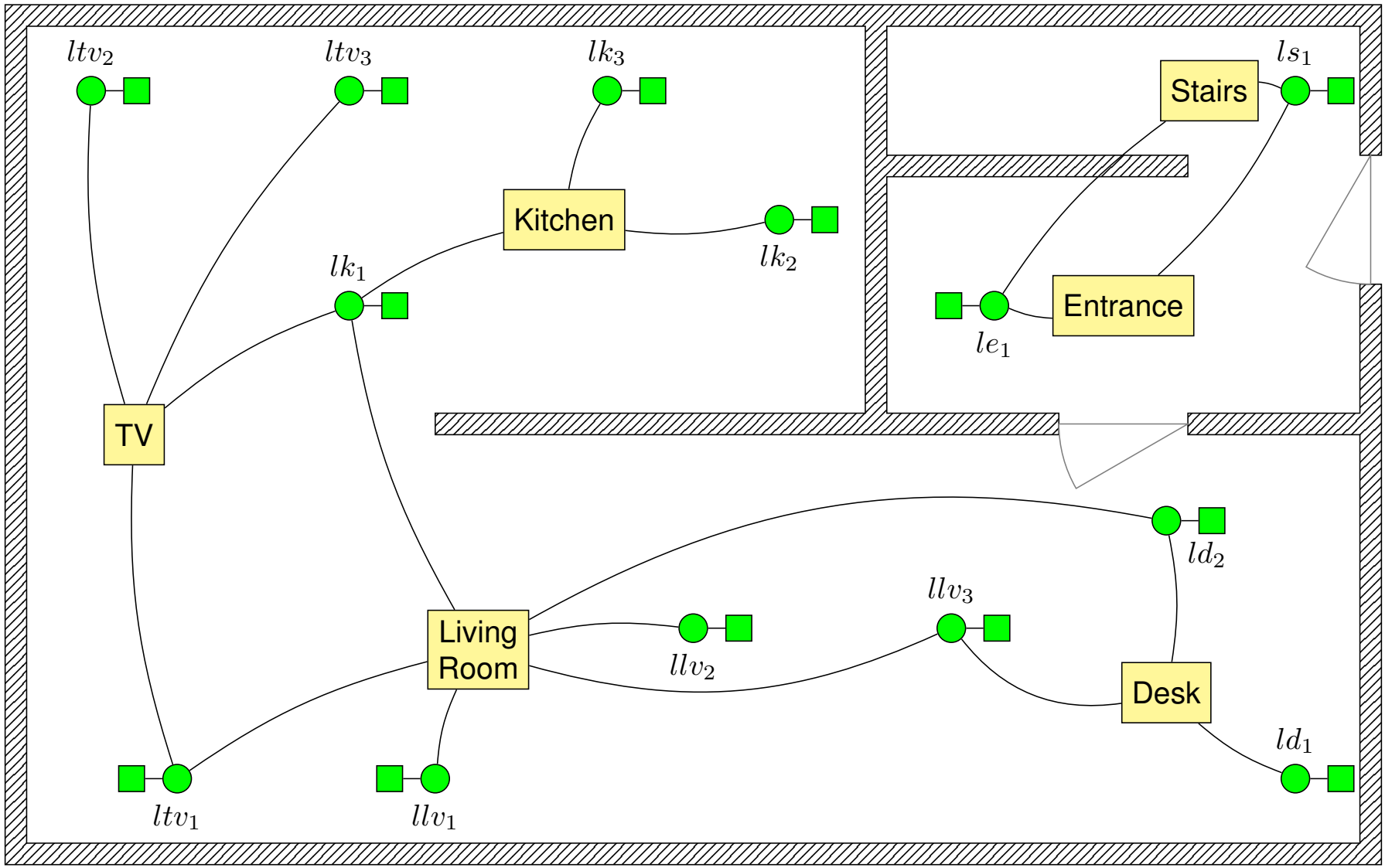
Formulating SECP as a DCOP

Representing a DCOP as a factor graph



SECP Factor Graph

in a house (without rules)



Menu

Distribution of computations

Allocating computations to agents

- DCOP: $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu \rangle$
- μ : function mapping variables to their associated agent

Why is distribution needed ?

Common assumptions:

- computation $\equiv$ variable
- each agent controls exactly one variable (bijection)
- binary constraints

Real distributed problems:

- agents must be hosted on real devices
- the set of devices might be given by the problem
- for some variables the relation with an agent is obvious, but not always

Distribution of computations

Allocating computations to agents

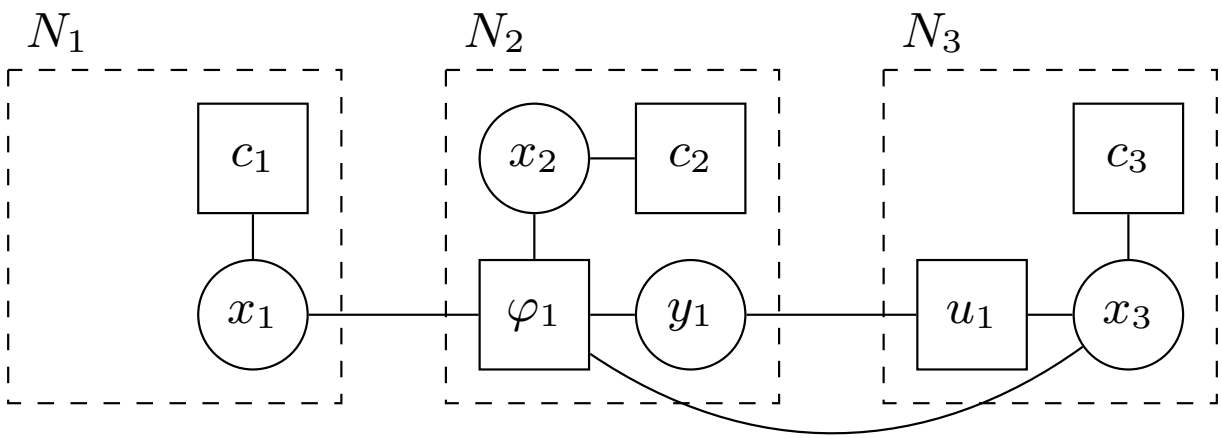
- Distributing computations
  - computations depends on the graph model
  - variables and / or factors

- Distribution impacts the system characteristics :
  - speed,
  - communication load,
  - hosting costs, etc.

Distribution of computations

Allocating computations to agents

- Simple heuristic
  - No computation on sleepy devices (sensors)
  - Computation should be close the the impacted variables
  - Spread the computation load amongst agents
- How good is it ?



Distribution of computations

Optimal definition

Optimal distribution ?

- Problem dependent
- Optimization problem :
  find the best distribution, for your problem's criteria
- Optimal distribution $\equiv$ graph partitioning,
  NP-hard in general (BOULLE, 2004)

Distribution of computations

Better definition

SECP distribution problem

- Devices have limited memory
- Communication is expensive and has limited bandwidth
- Variable related to an actuator are hosted by it
- Objective : **minimize overall communication between agents**

Optimization problem : define an ILP for it !

Binary ILP for computation distribution

- x_i^k , binary variables that map computations to agents and $\alpha_{ij}^{mn} = x_i^m \cdot f_j^n$

$$\forall x_i \in X, \quad \sum_{a_m \in \mathbf{A}} x_i^m = 1 \tag{1}$$

- Message's size between variable x_i and factor f_j: $msg(i,j)$

$$\underset{x_i^m}{\textbf{minimize}} \quad \sum_{(i,j)\in D} \sum_{(m,n)\in \mathbf{A^2}} \mathbf{msg}(i,j) \cdot \alpha_{ij}^{mn} \tag{2}$$

- Memory footprint of a computation: $\mathbf{weight}(e)$, and memory capacity for a device: $\mathbf{cap}(a_k)$

$$\forall a_m \in \mathbf{A}, \quad \sum_{x_i \in D} \mathbf{weight}(x_i) \cdot x_i^m \leq \mathbf{cap}(a_m) \tag{3}$$

- and a few linearization constraints

Binary ILP for computation distribution

More generic case:

- Add route cost: $\mathbf{com}(i, j, m, n)$

$$\forall x_i, x_j \in \mathbf{X}, \forall a_m, a_n \in \mathbf{A},$$

$$\mathbf{com}(i, j, m, n) = \begin{cases} \mathbf{msg}(i, j) \cdot \mathbf{route}(m, n) & \text{if } (i, j) \in D, m \neq n \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

$$\underset{x_i^m}{\textbf{minimize}} \sum_{(i,j)\in D} \sum_{(m,n)\in \mathbf{A}^2} \mathbf{com}(i, j, m, n) \cdot \alpha_{ij}^{mn} \quad (5)$$

- Add hosting costs : $\mathbf{host}(a_m, x_i)$

$$\underset{x_i^m}{\textbf{minimize}} \sum_{(x_i,a_m)\in X\times \mathbf{A}} x_i^m \cdot \mathbf{host}(a_m, x_i) \quad (6)$$

Binary ILP for computation distribution

$$\underset{x_i^m}{\textbf{minimize}} \quad \omega_{\mathbf{com}} \cdot \sum_{(i,j)\in D} \sum_{(m,n)\in \mathbf{A}^2} \mathbf{com}(i,j,m,n) \cdot \alpha_{ij}^{mn}$$

$$+ \quad \omega_{\mathbf{host}} \cdot \sum_{(x_i,a_m)\in X\times \mathbf{A}} x_i^m \cdot \mathbf{host}(a_m, x_i) \tag{7}$$

subject to

$$\forall a_m \in \mathbf{A}, \quad \sum_{x_i \in D} \mathbf{weight}(x_i) \cdot x_i^m \leq \mathbf{cap}(a_m) \tag{8}$$

$$\forall x_i \in X, \quad \sum_{a_m \in \mathbf{A}} x_i^m = 1 \tag{9}$$

$$\forall x_i \in X, \quad \alpha_{ij}^{mn} \leq x_i^m \tag{10}$$

$$\forall x_j \in X, \quad \alpha_{ij}^{mn} \leq x_j^m \tag{11}$$

$$\forall x_i, x_j \in X, a_m \in A, \quad \alpha_{ij}^{mn} \geq x_i^m + x_j^n - 1 \tag{12}$$

Solving the ILP for computation deployment

- NP-hard, but can be solved with branch-and-cut
  LP solvers are very good at this
- Yet, only possible for small instances
- Gives us a reference for optimality: benchmarking
- When not solvable, still gives us a metrics to compare heuristics

Menu

Hands on PyDCOP II

1. https://pydcop.readthedocs.io/en/latest/tutorials/deploying_on_machines.html
2. https://pydcop.readthedocs.io/en/latest/usage/cli/run.html
3. https://pydcop.readthedocs.io/en/latest/usage/cli/distribute.html

Hands on PyDCOP II

SECP

A Very simple SECP: single room

- 3 light bulbs, 1 model and 3 rules
- `/tutorials/hands-on_2/single_room.yaml`
- Solve with

```
pydcop --log log.conf -t 10 solve
  --algo maxsum --algo_params damping:0.8
  --dist adhoc single_room.yaml
```

- Result : `"cost": 702.3000000000004, ...`
  - not that good ...
  - Look at the yaml definition
  - the rules contradict each other !
- Change the yaml definition
  - comment out rules to keep only one active
  - could be done with *'read-only'* variables
  - solve it again

Hands on PyDCOP II

SECP - Running on several machines

- We used `solve`
  - great for testing
  - everything run locally, in the same process
- Launching several agents:
  - One agent for each light bulb `a1`, `a2` and `a3` (change port for each agent)

```
pydcop -v3 agent -n a1 -p 9001
   --orchestrator 127.0.0.1:9000
```

  - an orchestrator

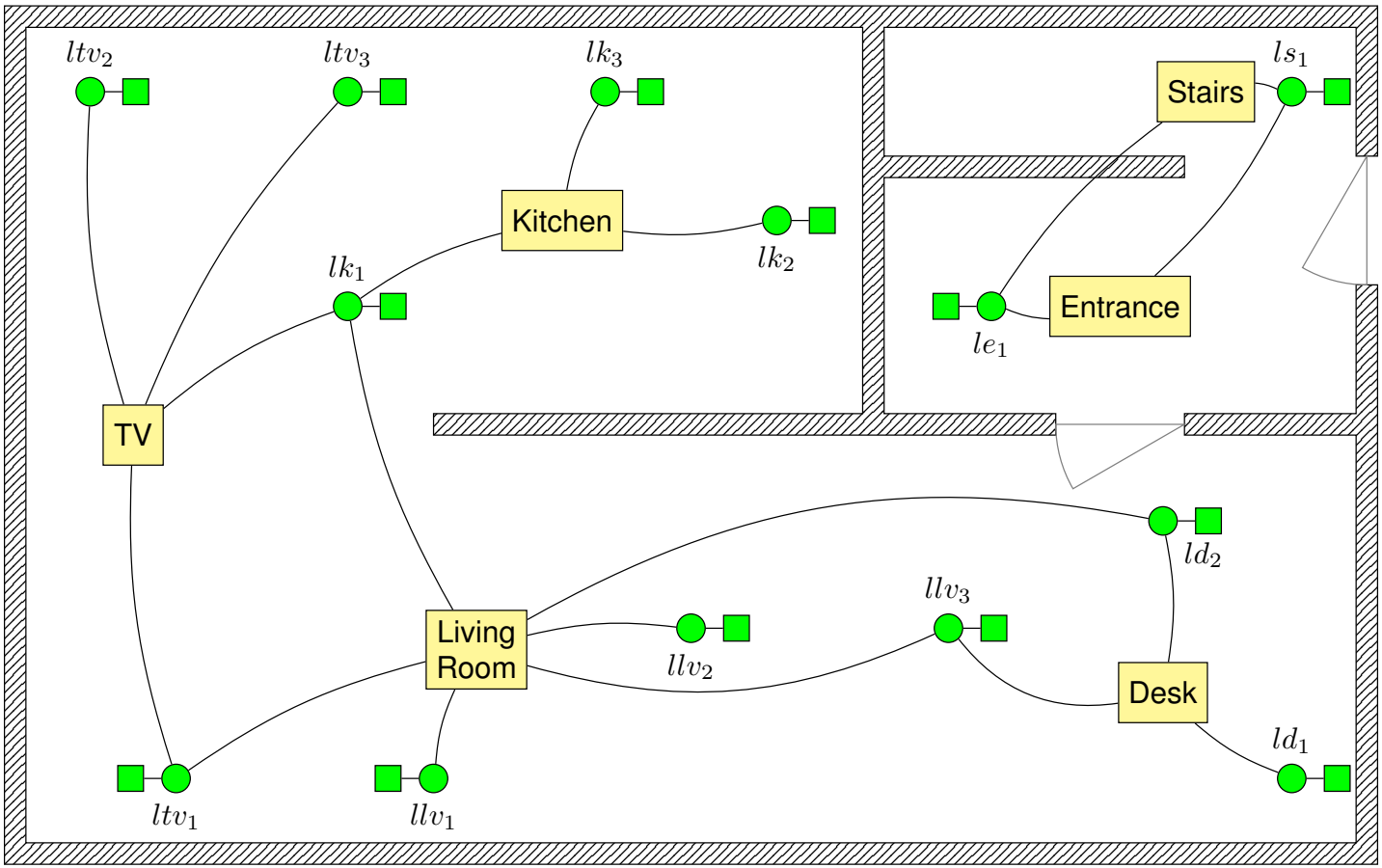
```
pydcop --log log.conf -t 10 orchestrator \
  --algo maxsum --algo_params damping:0.8 \
  --dist adhoc single_room.yaml
```

  - run the agents on different Virtual machines, different computers

Hands on PyDCOP II

A bigger SECP

in `/tutorials/hands-on_2/SimpleHouse.yml`
13 light bulbs, 6 models



Hands on PyDCOP II

Distributing a SECP

```
$ pydcop --output dist_house_fg_ilp.yaml distribute -d ilp_compref \
        -a maxsum SimpleHouse.yml
```

Need to specify the algorithm

- computation graph
- weight of computation's
- size of computations' messages

On such a small system, we can compute the optimal distribution !

Hands on PyDCOP II

Distributing a SECP

```
cost: 8725.0
distribution:
  a_d1: [mv_desk, mc_desk, l_d1, r_work, mc_livingroom, mv_livingroom]
  a_d2: [l_d2]
  a_e1: [mv_entry, r_entry, mv_stairs, l_e1, mc_entry, mc_stairs]
  a_e2: [l_e2]
  a_k1: [l_k1]
  a_k2: [l_k2]
  a_k3: [l_k3]
  a_lv1: [l_lv1]
  a_lv2: [mc_kitchen, l_lv2]
  a_lv3: [l_lv3]
  a_tv1: [l_tv1]
  a_tv2: [l_tv2]
  a_tv3: [r_lunch, l_tv3, mv_tv, r_cooking, r_homecinema, mc_tv, mv_kitchen]
inputs:
  algo: maxsum
  dcop: [SimpleHouse.yml]
  dist_algo: ilp_compref
  graph: factor_graph
```

Menu

SECP and DCOP

So far we have:

- Designed a model for SECP
- Formulated this model as a DCOP
- Distributed the computation of the DCOP on devices / agents (bootstrap)
- Run our system to get an auto-configuration of the devices

But what happens
in dynamic environments ?
if objects appear and disappear?

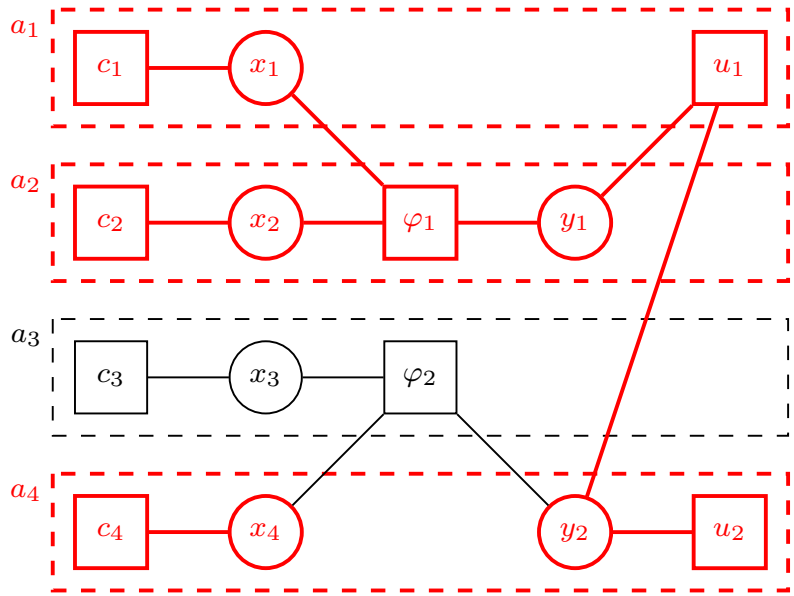
SECP is a dynamic problem

Dynamics in the infrastructure

- Devices can disappear
- New devices can be added to the system

At runtime..

- No powerful device available to solve the ILP
- The deployment must be repaired: self-adaptation
- Only consider a portion of the factor graph: the neighborhood



k-resilience

Dynamics in the infrastruture

Definition (k-resiliency)

k-resiliency is the capacity for a system to repair itself and operate correctly even in the case of the disappearance of up to k agents

- Two parts:
  - Do not loose the definition of the computations: **replication**
  - Migrate the orphaned computations to another agent: **selection** / activation
- Apply to any graph of computations, not only DCOP

Replication of computations

Replica distribution

- For each computation, place k replica on k other agents
  replica $equiv$ definition of the computation
- Must be distributed !
- Optimal replication ? impact the set of available agents when repairing
  which criteria ? too hard (quadratic multiple knapsack problem)...

Distributed Replica Placement Method (DRPM)

- Heuristic : place replica on agents close (network) the active computation, while respecting capacity
- Distributed version of iterative lengthening (aka uniform cost search based on path costs)

Replication of computations

iterative lengthening on route and hosting costs

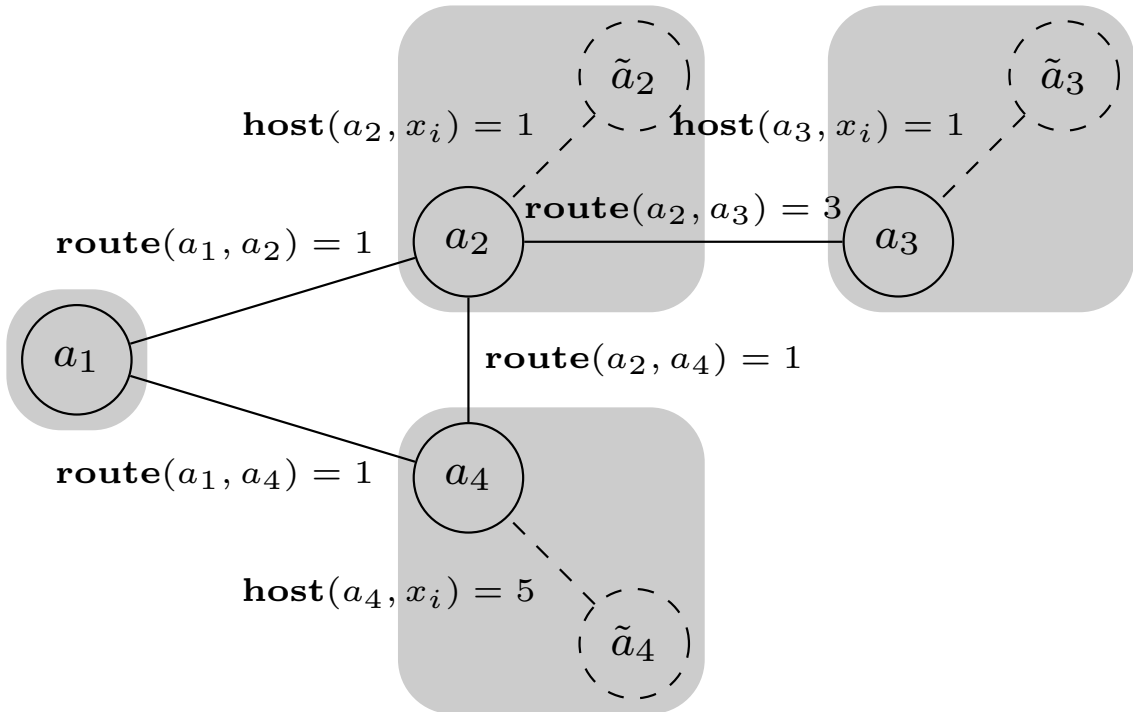


Figure: A sample **route+host**–graph with 4 agents (in gray), where a_1 search for hosting computation x_i. For $k = 2$, DRMP places a replica on a_2 (cost of $1 + 1 = 2$) and another on a_3 (cost of $1 + 3 + 1 = 5$) if enough capacity on these two agents, since the minimum cost path to host on a_4 is higher ($1 + 5 = 6$).

Migrating computations

Selecting an agent

Migrating a set of x_i computations X_c

- set of candidate agents A_c
- migrating the computation must not exceed agent's capacity
- for each computation, select the agent that minimize hosting and communication cost

Same optimization problem than for initial distribution, but on a subset of the of the graph

Distributed process !

Migrating computations

Selecting an agent

Distributed optimization problem $\Rightarrow$ let's use a DCOP!

- $\mathcal{A}$ is the set of candidate agents A_c
- $\mathcal{X}$ are the binary decision variables x_i^m
- $\mathcal{C}$ are the constraints ensuring that all computations are hosted, agent's capacities are respected and hosting and communication costs are minimized

Migrating computations

Selecting an agent

$$\sum_{a_m \in A_c^i} x_i^m = 1 \tag{13}$$

$$\sum_{x_i \in X_c^m} \mathbf{weight}(x_i) \cdot x_i^m + \sum_{x_j \in \mu^{-1}(a_m) \setminus X_c} \mathbf{weight}(x_j) \leq \mathbf{cap}(a_m) \tag{14}$$

$$\sum_{x_i \in X_c^m} \mathbf{host}(a_m, x_i) \cdot x_i^m \tag{15}$$

$$\begin{aligned} &\sum_{(x_i, x_j) \in X_c^m \times N_i \setminus X_c} x_i^m \cdot \mathbf{com}(i, j, m, \mu^{-1}(x_j)) \\ + &\sum_{(x_i, x_j) \in X_c^m \times N_i \cap X_c} x_i^m \cdot \sum_{a_n \in A_c^j} x_j^n \cdot \mathbf{com}(i, j, m, n)) \end{aligned} \tag{16}$$

Decentralized reparation

When agents are removed:

- computation to migrate = computation that where hosted on these agents
- candidate agents = remaining agents that posses a replica of these orphaned computation

Solving the migration DCOP

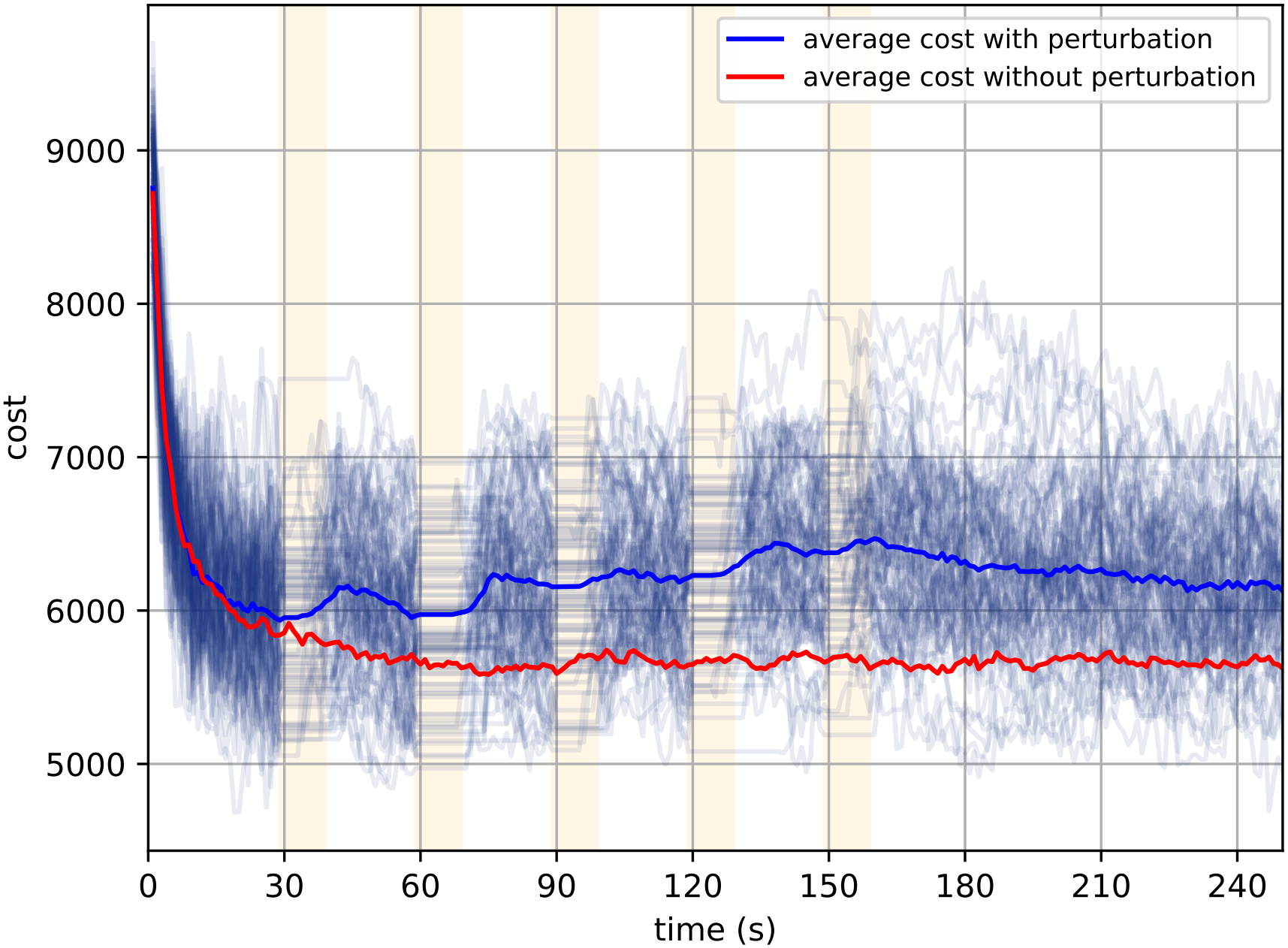
Which algorithm should we use ?

Criteria:

- lightweight
- fast (even if not optimal !)
- monotonic : mix of hard and soft constraints

MGM-2 : like MGM, with 2-coordination

Experimental results



Menu

To sum up

What we've seen today:

- Some generic concepts
  - How to model coordination problems using DCOP formalism
  - Some solution methods (complete and incomplete) to solve DCOP
- Some specificities of IoT-based apps
  - How to model a specific smart environment configuration problem as a DCOP
  - How to use PyDCOP to model, run, solve, and distribute DCOP
  - How to equip a system with resilience using replication and DCOP-based reparation

- Want to go deeper into DCOPs $\rightarrow$ OPTMAS-DCR workshop series (AAMAS/IJCAI), other tutorials at AAMAS/IJCAI

The End

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.

BOULLE, M. (2004). "Compact Mathematical Formulation for Graph Partitioning". In: *Optimization and Engineering* 5.3, pp. 315–333. ISSN: 1573-2924. DOI: 10.1023/B:OPTE.0000038889.84284.c7. URL: http://dx.doi.org/10.1023/B:OPTE.0000038889.84284.c7.

BÜRGER, M., G. NOTARSTEFANO, F. BULLO, and F. ALLGÖWER (2012). "A distributed simplex algorithm for degenerate linear programs and multi-agent assignments". In: *Automatica* 48.9, pp. 2298 –2304. ISSN: 0005-1098. DOI: http://dx.doi.org/10.1016/j.automatica.2012.06.040. URL: //www.sciencedirect.com/science/article/pii/S0005109812002956.

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.

FIORETTO, F., E. PONTELLI, and W. YEOH (2018). "Distributed Constraint Optimization Problems and Applications: A Survey". In: *Journal of Artificial Intelligence Research* 61, pp. 623–698.

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.

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.

References (cont.)

RUST, P., G. PICARD, and F. RAMPARANY (2016). "Using Message-passing DCOP Algorithms to Solve Energy-efficient Smart Environment Configuration Problems". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press.

— (2017). "On the Deployment of Factor Graph Elements to Operate Max-Sum in Dynamic Ambient Environments". In: *Autonomous Agents and Multiagent Systems – AAMAS 2017 Workshops, Best Papers, Sao Paulo, Brazil, May 8-12, 2017, Revised Selected Papers*. Ed. by G. SUKTHANKAR and J.A. RODRIGUEZ-AGUILAR. Vol. 10642. Lecture Notes in Artificial Intelligence (LNAI). Extended Version. Springer, pp. 116–137. DOI: 10.1007/978-3-319-71682-4_8.

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 (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.

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.