Self-Organized and Resilient Distribution of Decisions over Dynamic Multi-Agent Systems

Pierre Rust[1,2] Gauthier Picard[1] Fano Ramparany[2]

[1]MINES Saint-Étienne, CNRS
Lab Hubert Curien UMR 5516

[2]Orange Labs

Distributed Decision-making over Dynamic Multi-Agent Systems

Decisions

- Constraints Optimization Problem
- Decisions $\equiv$ variables

Distributed

Dynamic

Distributed Decision-making over Dynamic Multi-Agent Systems

Decisions

- Constraints Optimization Problem
- Decisions $\equiv$ variables

Distributed

- Multi-agents
- DCOP
- **Efficient** distribution of the decisions

Dynamic

Distributed Decision-making over Dynamic Multi-Agent Systems

Decisions

- Constraints Optimization Problem
- Decisions $\equiv$ variables

Distributed

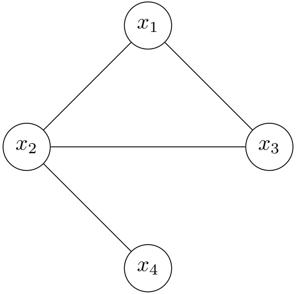
- Multi-agents
- DCOP
- **Efficient** distribution of the decisions

Dynamic

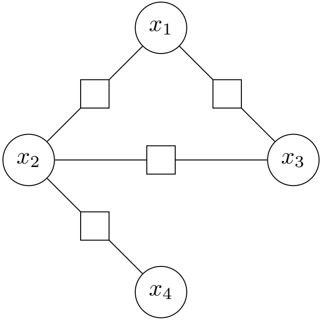
- Agents leave / join the system
- Decisions must be preserved
- Decisions must be migrated

Distribution of decision

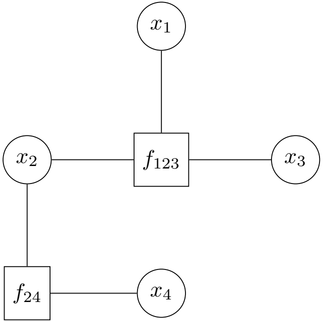
- DCOP $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu \rangle$
- Several graph representations
- Nodes in the graph = computations
- Distribute computation on agents



(a) Simple constraint graph



(b) Factor graph

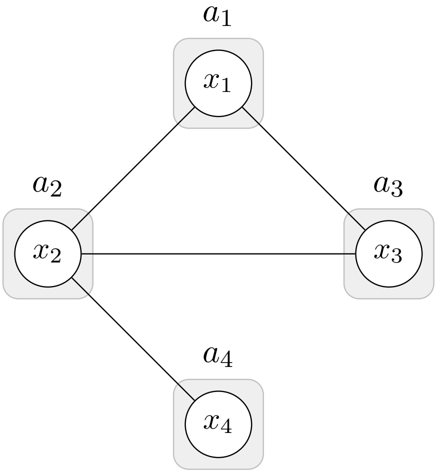


(c) Factor graph

Distributing computations

Computations

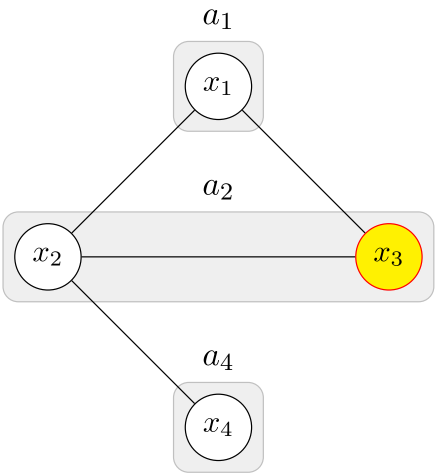
- **belong** to an agent :
  "natural" link,
  problem characteristics



Distributing computations

Computations

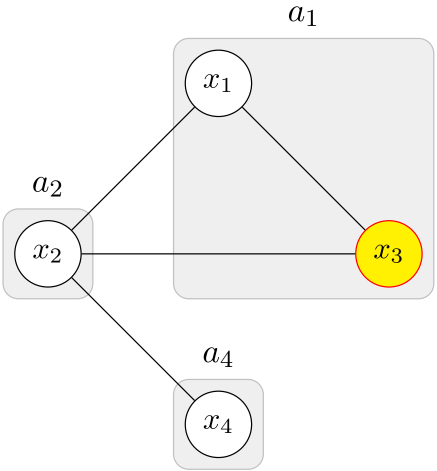
- **belong** to an agent
- **shared** decisions :
  modeling artifact, with no obvious agent relation (e.g. distributed meeting scheduling)



Distributing computations

Computations

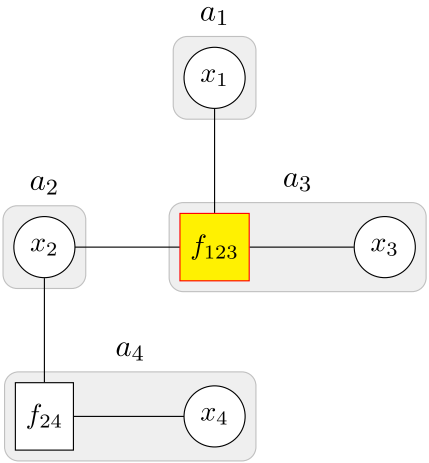
- **belong** to an agent
- **shared** decisions :
  modeling artifact, with no obvious agent relation (e.g. distributed meeting scheduling)



Distributing computations

Computations

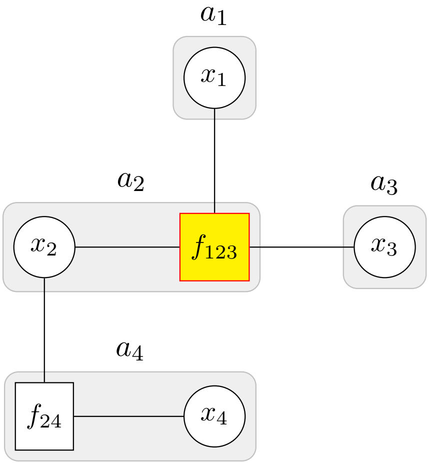
- **belong** to an agent
- **shared** decisions
- **factors**, in a factor graph:
  not representing a decision variable



Distributing computations

Computations

- **belong** to an agent
- **shared** decisions
- **factors**, in a factor graph:
  not representing a decision variable



Distributing computations

Distribution impacts the system characteristics

- speed
- communication load
- hosting costs / preferences

Optimal distribution

- problem dependent
- optimization problem: find the best distribution for your problem criteria
- determining the optimal distribution $\equiv$ graph partitioning
  NP-hard in general [BOULLE, 2004]

Optimal distribution definition

Generic definition

- Meet agents' capacity limit & computation footprint

$$\forall a_m \in \mathbf{A}, \quad \sum_{x_i \in D} \mathbf{weigh}(x_i) \cdot x_i^m \leq \mathbf{cap}(a_m) \tag{1}$$

- Minimize communication load
- Minimize hosting costs

Optimal distribution definition

Generic definition

- Meet agents' capacity limit
- Minimize communication load :
  with different communication costs for different edges

$$\underset{x_i^m}{\textbf{minimize}} \quad \sum_{(i,j) \in D} \sum_{(m,n) \in \mathbf{A}^2} \mathbf{com}(i,j,m,n) \cdot \alpha_{ij}^{mn} \tag{1}$$

- Minimize hosting costs

Optimal distribution definition

Generic definition

- Meet agents' capacity limit
- Minimize communication load
- Minimize hosting costs :
  can be used to model preferences, operational costs, etc.

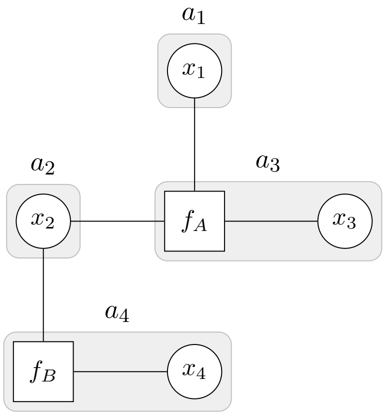
$$\underset{x_i^m}{\textbf{minimize}} \sum_{(x_i, a_m) \in X \times \mathbf{A}} x_i^m \cdot \mathbf{host}(a_m, x_i) \tag{1}$$

Distributing Computations

Optimal distribution

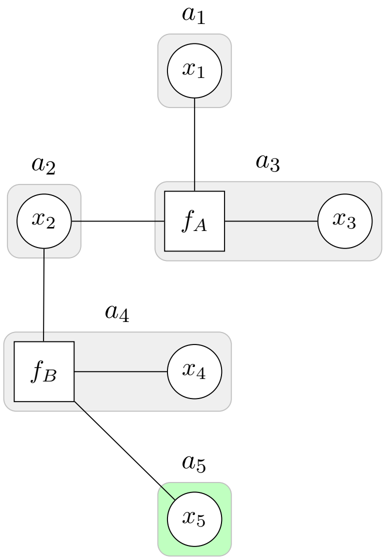
- NP-hard, but can be solved with branch-and-cut
  LP solvers are very good at this
- Useful to bootstrap a system
- Yet, only possible for relatively small instances
- When not solvable, still gives us a metrics to compare heuristics

Dynamic and Open System



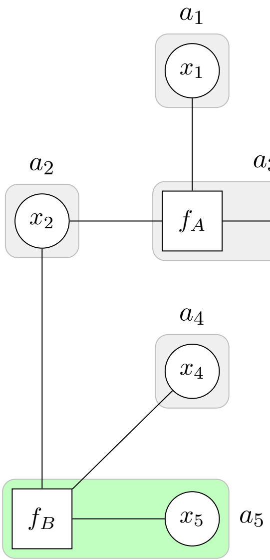
Dynamic and Open System

- New agents may **join** the system
  - Use the extra help / computing power ?
  - Migrate computations ?



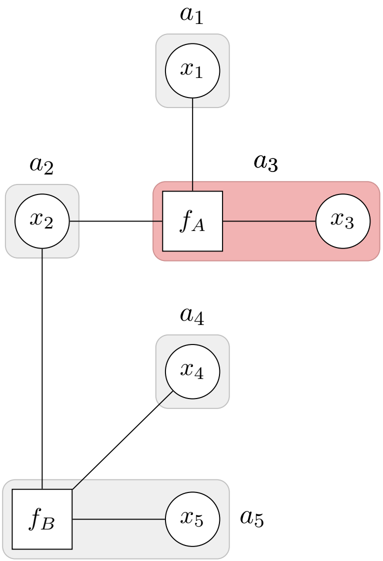
Dynamic and Open System

- New agents may **join** the system
  - Use the extra help / computing power ?
  - Migrate computations ?



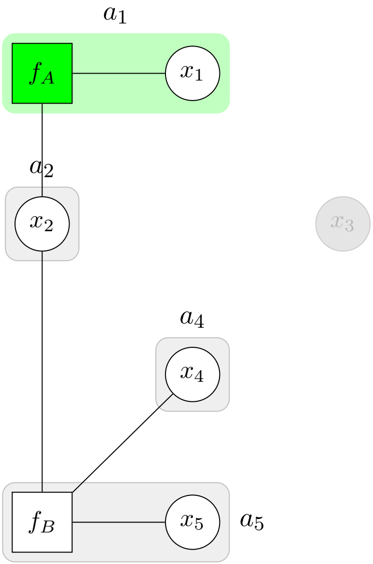
Dynamic and Open System

- New agents may **join** the system
- Agents may **leave** the system at any time



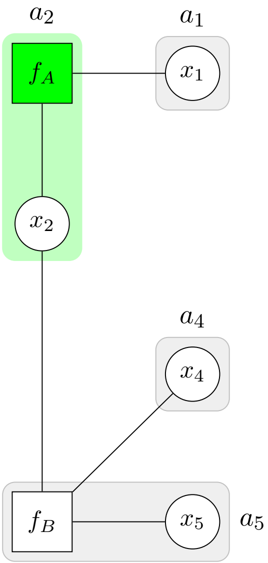
Dynamic and Open System

- New agents may **join** the system
- Agents may **leave** the system at any time
  - How to ensure that the system still works as expected ?
  - Migrate computation to remaining agents



Dynamic and Open System

- New agents may **join** the system
- Agents may **leave** the system at any time
  - How to ensure that the system still works as expected ?
  - Migrate computation to remaining agents



k-resilience

Definition (k-resilience)

Given a set of agents $\mathbf{A}$, a set of computations $\mathbf{X}$, and a distribution μ, the system is k**-resilient** if for any subset $F \subset \mathbf{A}, |F| \leq k$, a new distribution $\mu' : X \rightarrow \mathbf{A} \setminus F$ exists.

Implementation

- Having decisions' definition available : **replication** of computations
- Migrate orphaned computations : **selection** of candidate

Replication for k-resilience

Replica placement

- Replicate computations on k agents
- Respect agents' capacity
- Optimize for communication and hosting costs

Optimal Replication ?

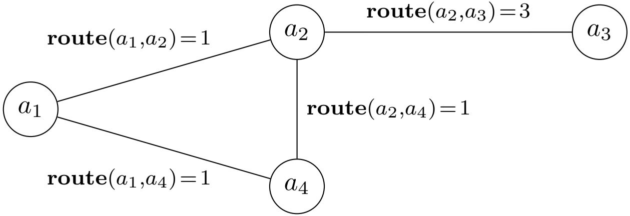
- Huge problem space $\equiv$ quadratic multiple knapsack problem (QMKP) [Saraç and Sipahioglu, 2014], NP-hard.
- No clear definition of what would be optimal !

Replication for k-resilience (cont.)

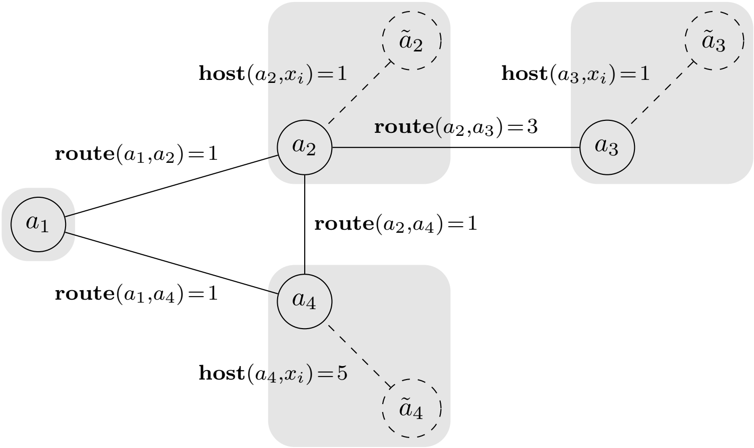
DPRM Heuristic

- Use the computation graph: communication costs
- Add extra nodes to account for hosting costs
- Use Iterative Lengthening / Uniform Cost Search on the graph
- Distributed implementation
- Initiated by each agent, for each of its computations to replicate

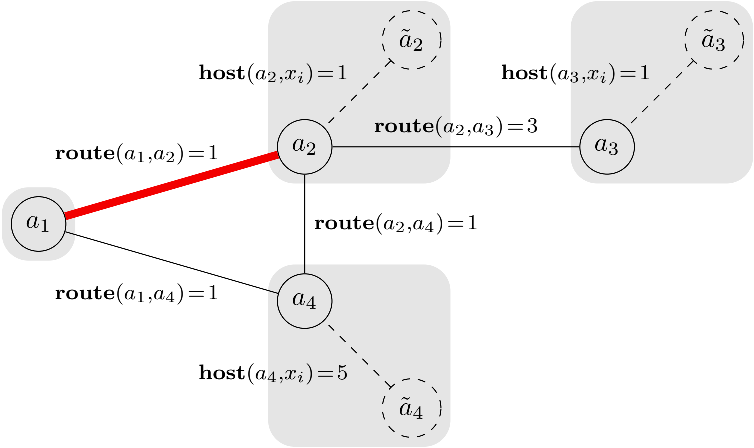
Distributed Replica Placement Method



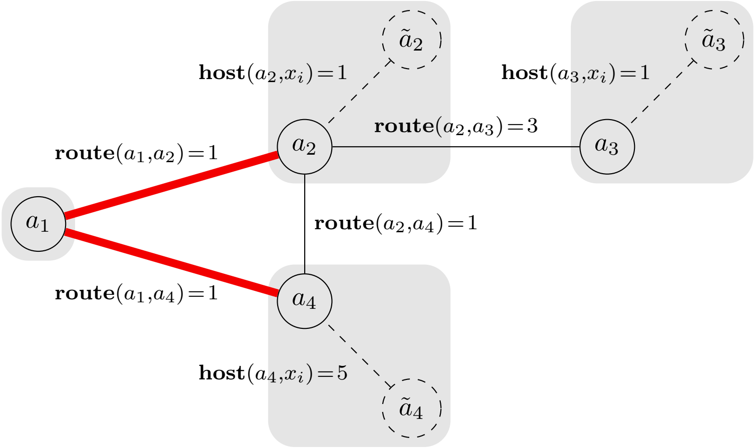
Distributed Replica Placement Method



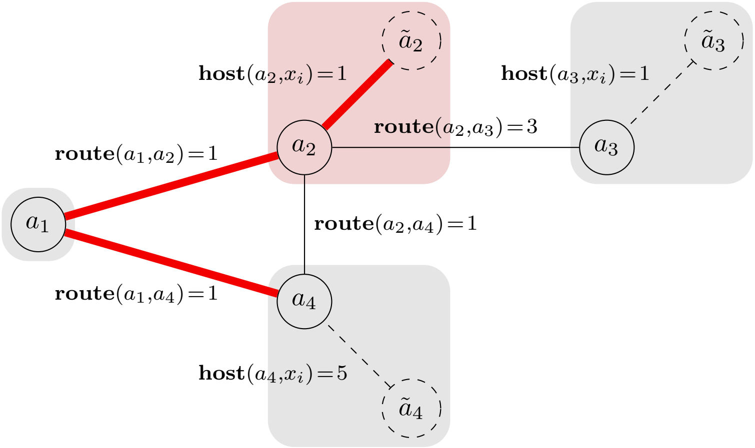
Distributed Replica Placement Method



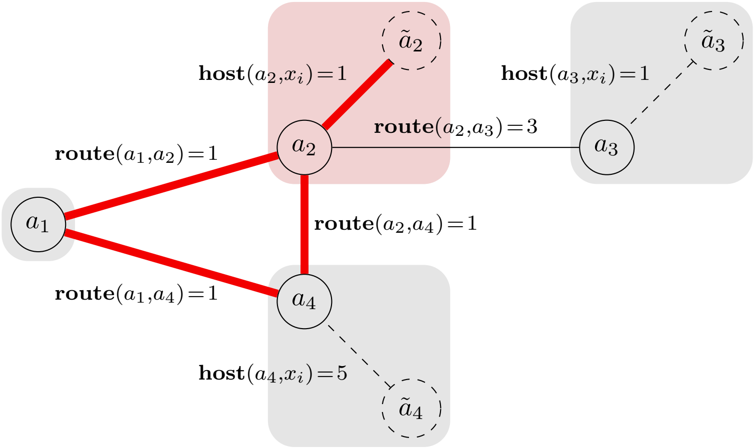
Distributed Replica Placement Method



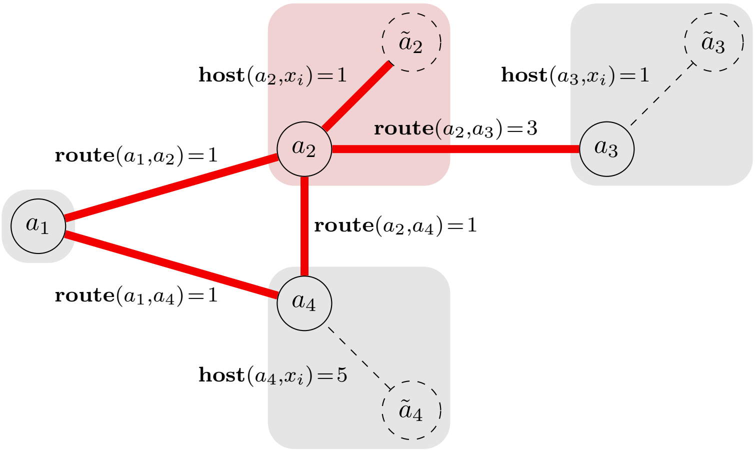
Distributed Replica Placement Method



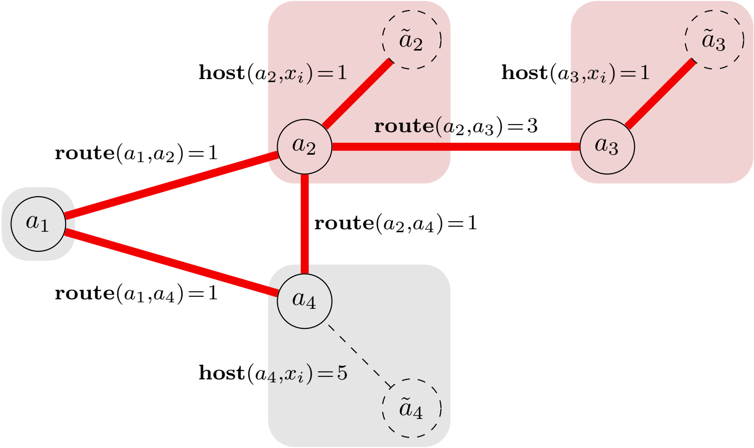
Distributed Replica Placement Method



Distributed Replica Placement Method



Distributed Replica Placement Method



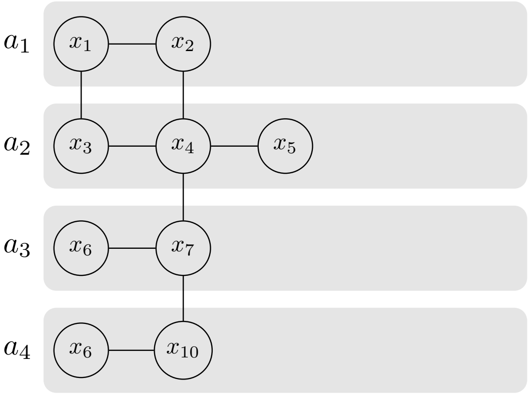
Distributed repair - selecting candidates

Repairing by migrating computations

- Orphaned computations X_c: hosted on a departed agent
- Candidate agents A_c:
  agents possessing replicas of orphaned computation ($\leq k$ for each computation)
- Select exactly one candidate agent for each orphaned computation
- Respect the agent's capacity
- Select the candidate that minimizes communication and hosting costs

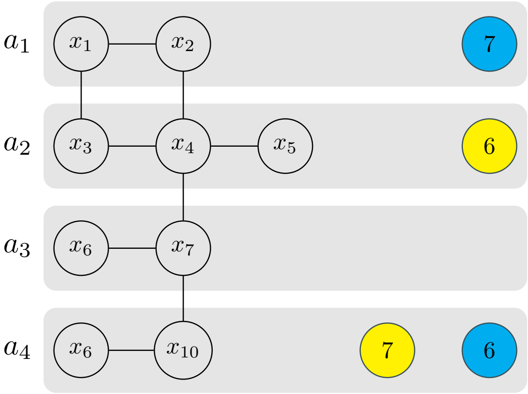
Like the initial distribution problem, but on a very restricted subset of the graph

Distributed repair - selecting candidates



DCOP with 9 computations distributed on 4 agents

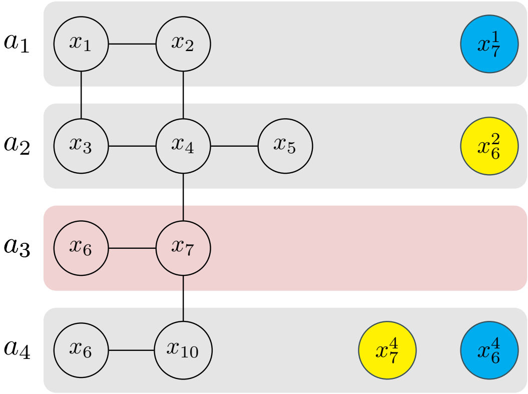
Distributed repair - selecting candidates



DCOP with 9 computations distributed on 4 agents

- Replicas for x_6 hosted on a_2, a_4
- Replicas for x_7 hosted on a_1, a_4

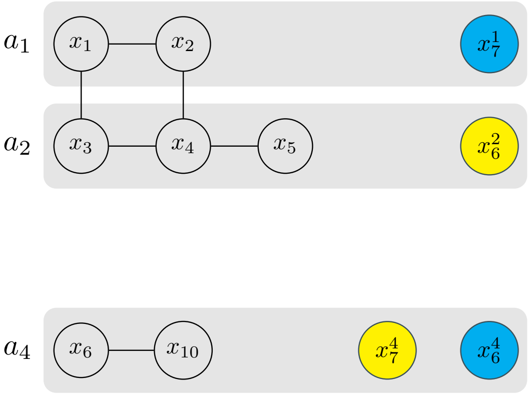
Distributed repair - selecting candidates



Agent a_3 leaves the system

- x_6 and x_7 must be moved
- Candidate agents :
  x_6 : { a_2, a_4 }
  x_7 : { a_1, a_4 }
- Binary variables for each replica : x_i^m
- Model the selection as an optimization problem

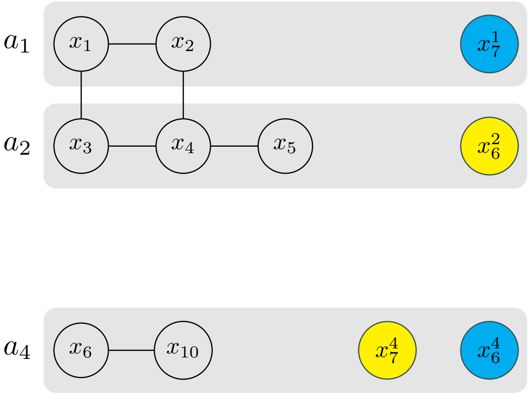
Distributed repair - selecting candidates



Model the selection as an optimization problem
All orphaned computation must be hosted:

$$\sum_{a_m \in A_c^i} x_i^m = 1 \tag{2}$$

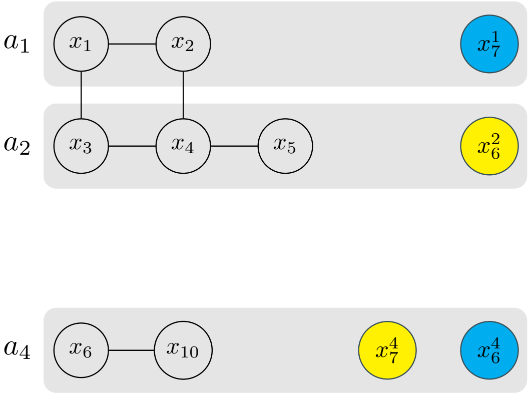
Distributed repair - selecting candidates



Capacity constraints

$$\sum_{x_i \in X_c^m} \mathbf{weigh}(x_i) \cdot x_i^m + \sum_{x_j \in \mu^{-1}(a_m) \setminus X_c} \mathbf{weigh}(x_j) \leq \mathbf{cap}(a_m) \quad (2)$$

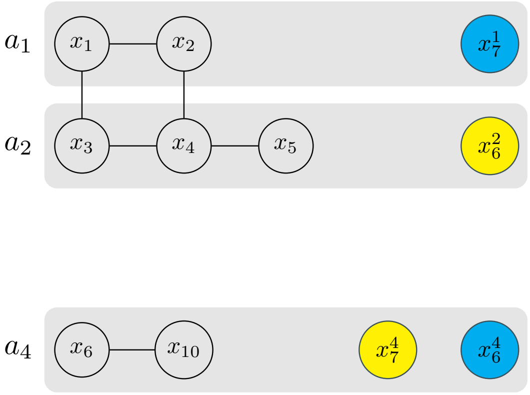
Distributed repair - selecting candidates



Minimize hosting costs

$$\sum_{x_i \in X_c^m} \mathbf{host}(a_m, x_i) \cdot x_i^m \tag{2}$$

Distributed repair - selecting candidates



Minimize communication constraints

$$\sum_{(x_i,x_j)\in X_c^m \times N_i \setminus X_c} x_i^m \cdot \mathbf{com}(i,j,m,\mu^{-1}(x_j)) + \sum_{(x_i,x_j)\in X_c^m \times N_i \cap X_c} x_i^m \cdot \sum_{a_n \in A_c^j} x_j^n \cdot \mathbf{com}(i,j,m,n)) \quad (2)$$

Solving the selection problem

Optimization problem

- The candidate selection is modeled as a DCOP
- We use a DCOP to repair the distribution of the original DCOP !
  - original DCOP : variables = decisions for our problem
  - repair DCOP : variables = candidate selection

Resolution

- MGM2
- fast, lightweight, monotonous
- good behavior with soft / hard constraints
- no issue with distribution in that case

Experimental results - IoT-like setup

Problem

- 100 variables, Domain size 10
- Constraints graph: scale-free graph (Barabási–Albert), binary constraints
- Uniform random cost functions

Infrastructure

- 100 agents
- Hosting costs: x_i prefers a_i random
- Route costs: respect the scale free distribution of the graph

Disturbance scenario

- every 30 seconds, 3 agents are removed

The original problem is solved using maxsum, the repair problems with MGM-2
Generate 20 instances, 5 run each
Solve with and without disturbance

Experimental results

Average cost of the solution

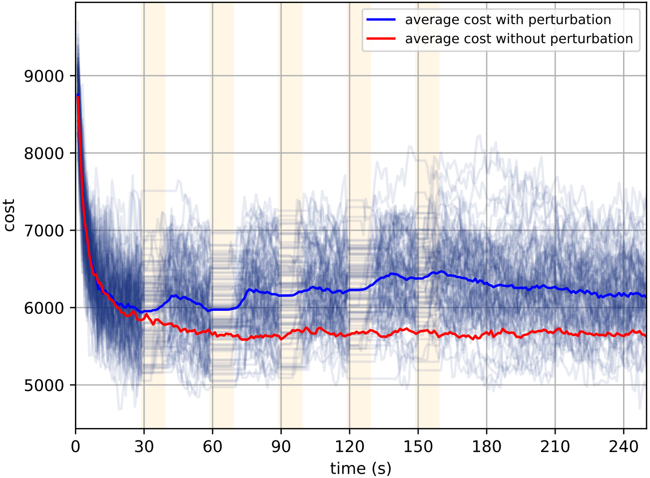


Figure: Average cost of MaxSum solution at runtime, on scale-free DCOPs, with (blue) and without perturbation (red).

Experimental results

Average cost of the distribution

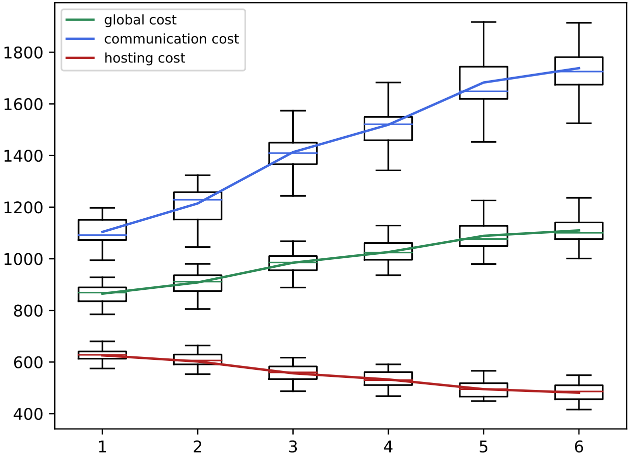


Figure: Cost of computation distribution after each event

Conclusion

Summary

- Definition of an optimal distribution of computations on a set of agents
- Distributed algorithm for computation replication (stateless)
- Distributed repair method, based on a DCOP

Future work

- Relax the requirements on the DCOP algorithm used for the original problem
- Test with other algorithms (only max-sum at the moment)
- More experimentations with other problem domain
  - more graph topologies (grid, random, etc.)
  - other types of problem (meeting scheduling, target tracking, etc)
  - non-DCOP computation graph

Self-Organized and Resilient Distribution of Decisions over Dynamic Multi-Agent Systems

Pierre Rust[1,2] Gauthier Picard[1] Fano Ramparany[2]

[1]MINES Saint-Étienne, CNRS
Lab Hubert Curien UMR 5516

[2]Orange Labs