Distributed Constraint Reasoning 2

Michal Jakob

Agent Technology Center,
Dept. of Computer Science and Engineering,
FEE, Czech Technical University

AE4M36MAS Autumn 2013 - Lecture 10

Where are We?

Agent architectures (inc. BDI architecture)
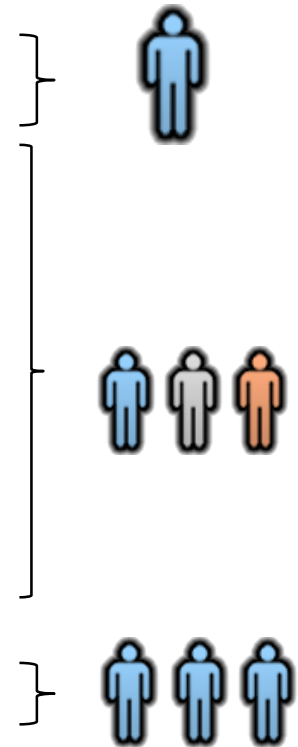
Logics for MAS

Non-cooperative game theory

Cooperative game theory

Auctions

Social choice

Distributed constraint reasoning

Formalism Review

Distributed Constraint Reasoning

Constraint Network

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

- $X = \{x_1, \ldots, x_n\}$ is a set of **variables**;

- $D = \{D_1, \ldots, D_n\}$ is a set of finite **variable domains**, which enumerate all possible values of the corresponding variables;

- $C = \{C_1, \ldots, C_m\}$ is a set of **constraints**; where a constraint C_i is defined on a subset of variables $S_i \subseteq X$ which comprise the **scope of the constraint**

  - $r_i = |S_i|$ is the **arity** of constraint i

Hard vs. Soft Constraints

Hard constraint C_i^h is a Boolean **predicate** P_i that defines **valid joint assignments** of variables in the scope

$$P_i: D_{i_1} \times \cdots \times D_{i_r} \rightarrow \{\mathrm{F}, \mathrm{T}\}$$

Soft constraint C_i^s is a **function** F_i that maps every possible joint assignment of all variables in the scope to a real value

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

We further assume F_i is non-negative

- non-restrictive, can always shift

Binary Constraint Networks

Binary constraint networks are those where each **constraint** (soft or hard) is defined **over two variables**.

Every constraint network can be **mapped to a binary** constraint network

- requires the addition of variables and constraints
- may add complexity to the model

Binary constraint networks can be represented by a **constraint graph**
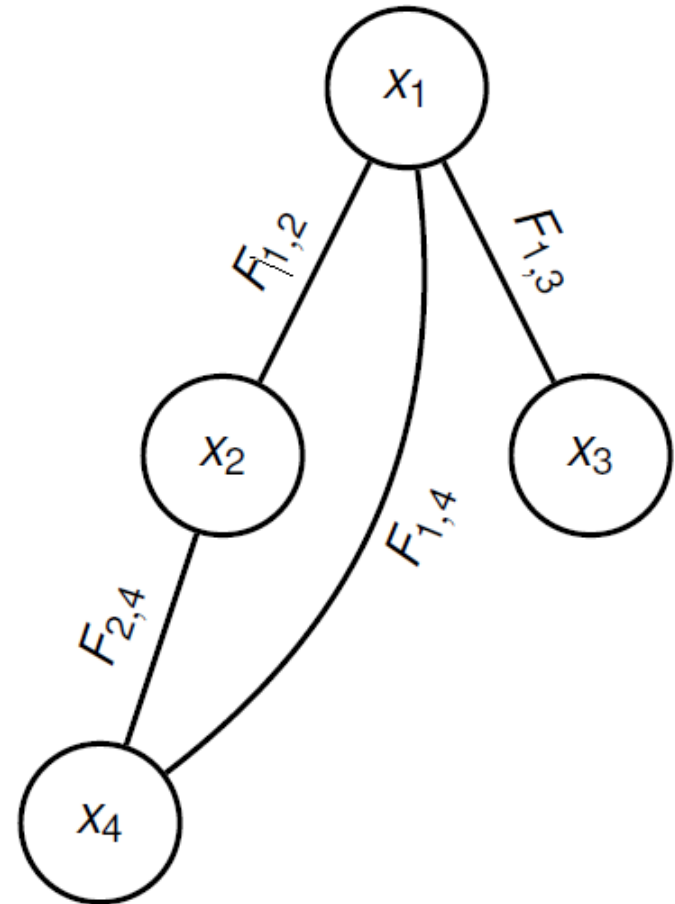
Distributed Constraint Reasoning Problem

A distributed constraint reasoning problem consists of a **constraint network** $\langle X, D, C \rangle$ and a **set of agents** $A = \{A_1, \ldots, A_k\}$ where each agent:

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

This lecture: Distributed Constraint Optimization Problems (DCOPs)

# Algorithms for Distributed Constraint Optimization
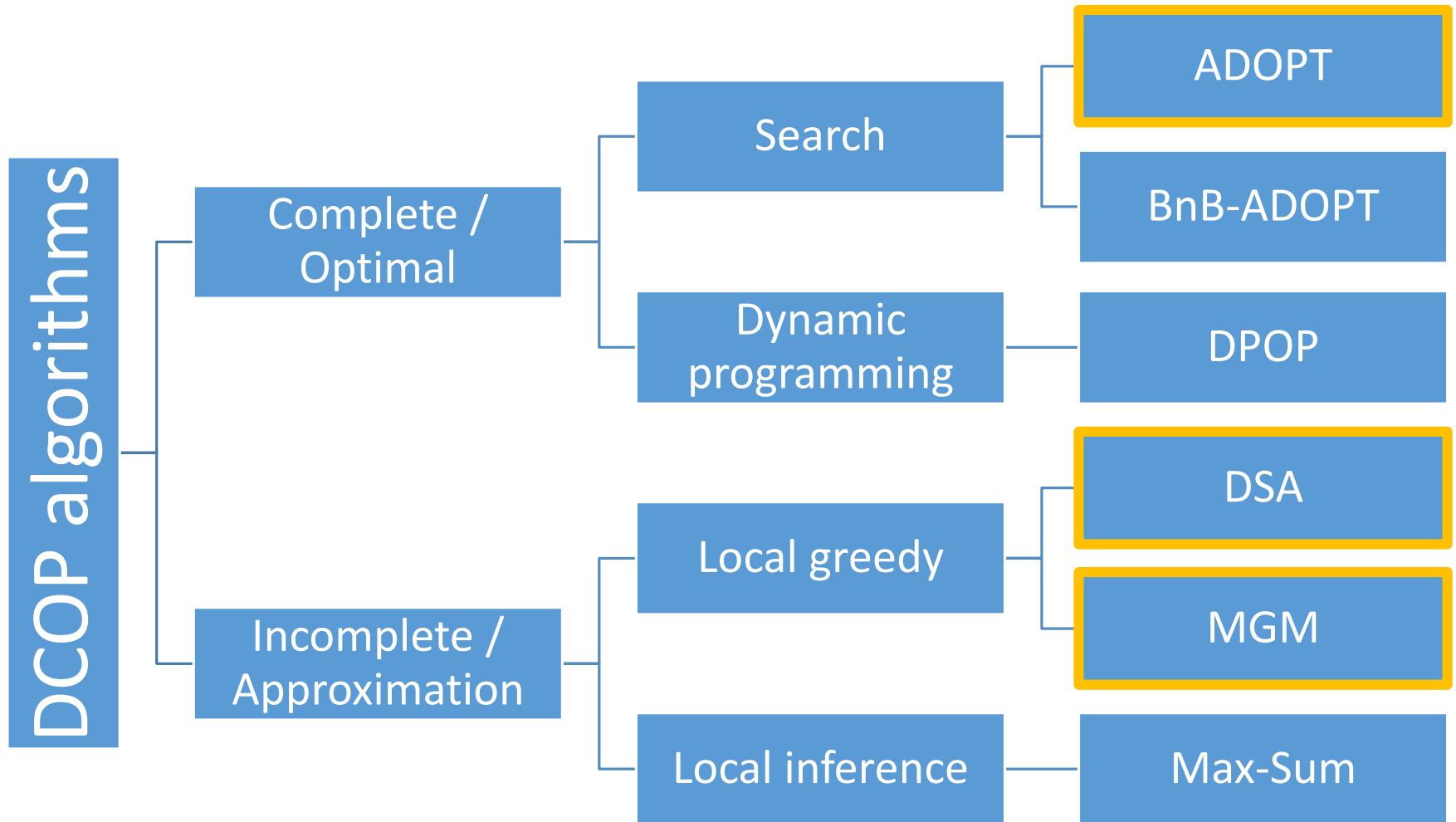
Complete Algorithms

Distributed Constraint Optimization

Synchronous BnB

Agents agree on an variable order and repeat:

1. send partial solution up to X_{k-1} to k-th agent.
2. k-th agent generates the next extension to this partial solution whose partial cost (i.e. lower bound) is not greater than the upper bound.
3. if the solution cannot be extended: $k \leftarrow k - 1$ (backtrack control to previous agent).
4. if solution can be extended consistently: update lower bound, $k \leftarrow k + 1$ (pass control to the next agent)
5. if $k > n$: stop → if lower-bound (now the total cost) < upper bound, then upper bound = lower bound; remember best so far assignment
6. if $k < 1$: stop → return best so far assignment.
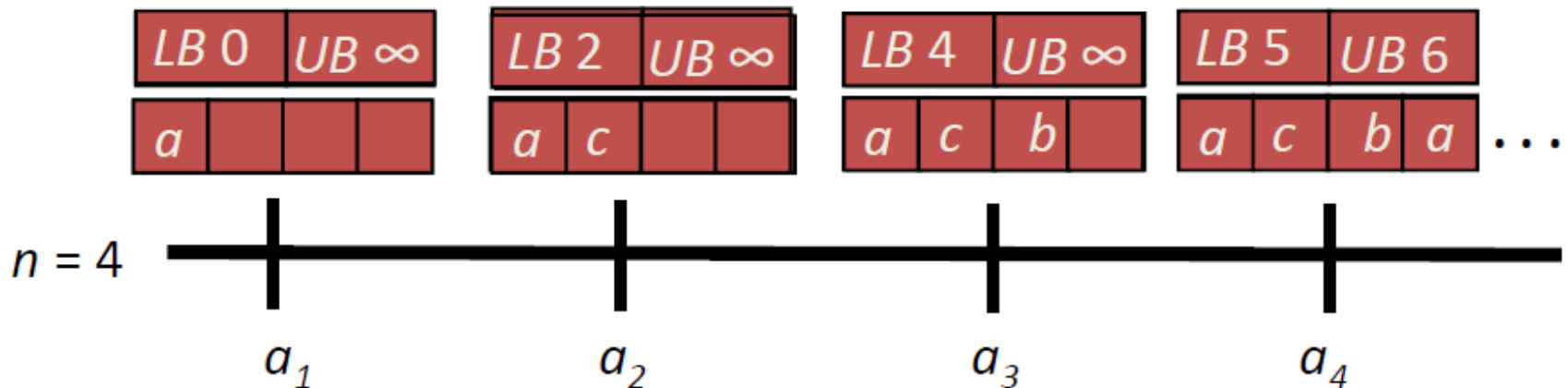
Asynchronous Backtracking: Assumptions

1. Agents communicate by **sending messages**

2. An agent can send messages to others, iff it knows their identifiers (**directed communication** / no broadcasting)

3. The **delay** transmitting a message is **finite** but random

4. For any pair of agents, messages are **delivered in the order** they were sent

5. Agents **know the constraints in which they are involved**, but not the other constraints

6. Each agent owns a **single variable** (agents = variables)

7. Constraints are **binary** (2 variables involved)

not essential, can be lifted

Decentralised Complete Algorithms

Search-based

- Uses distributed **search**
- Exchange individual **values**
- **Small messages** but
- . . . exponentially **many**
- Representative: **ADOPT**

Dynamic programming

- Uses distributed **inference**
- Exchange **constraints**
- **Few messages** but
- . . . exponentially **large**
- Representative: **DPOP**

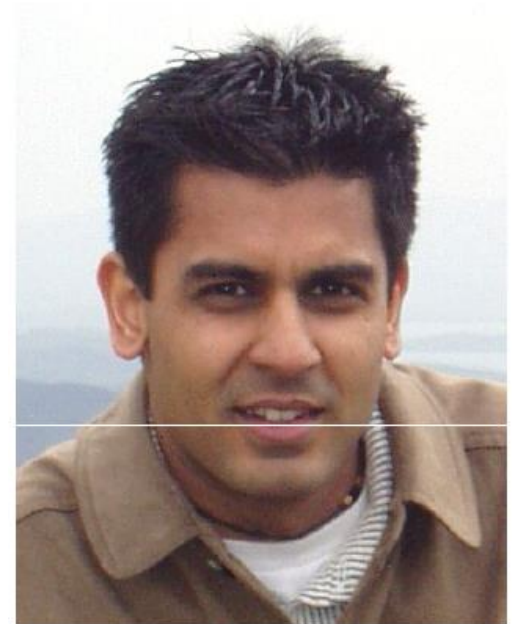# ADOPT*: Asynchronous Distributed OPTimization

First **asynchonous complete** algorithm for optimally solving DCOP

Distributed backtrack search using a "**opportunistic**" **best-first** strategy

- agents keep on choosing the **best** value based on the **current** available **information**

Backtrack thresholds used to speed up the search of previously explored solutions.

Termination conditions that check if the bound interval is less than a given valid error bound (0 if optimal)

Pragnesh Jay Modi and colleagues

*ADOPT: asynchronous distributed constraint optimization with quality guarantees; P. Jay Modi, W. M. Shen, M. Tambe, M. Yokoo, Artificial Intelligence, 2005

ADOPT Overview

Opportunistic best-first search strategy, i.e., each agent keeps on choosing the value with minimum lower bound.

- **Lower bounds** are more suitable for **asynchronous search**—a lower bound can be computed **without** necessarily having **accumulated global cost** information.

Each agent keeps a **lower and upper bound** on the cost for the **sub-problem below** it (given **assignments from above**) and on the **sub-problems for** each one of its **children**.

It then tells the children to **look for a solution** but **ignore** any **partial solution** whose **cost** is **above** the lower bound because it already knows that it can get that lower cost.

ADOPT: DFS Tree

ADOPT assumes that agents are arranged in a **depth-first search (DFS) tree**:

- split constraint graph into a **spanning tree** and **backedges**
- two constrained nodes must be in the same path to the root by tree links (same branch), i.e., backedges from a node go to the ancestors of the node

Every graph admits a DFS tree. A DFS can be constructed in **polynomial time** using a **distributed** algorithm.
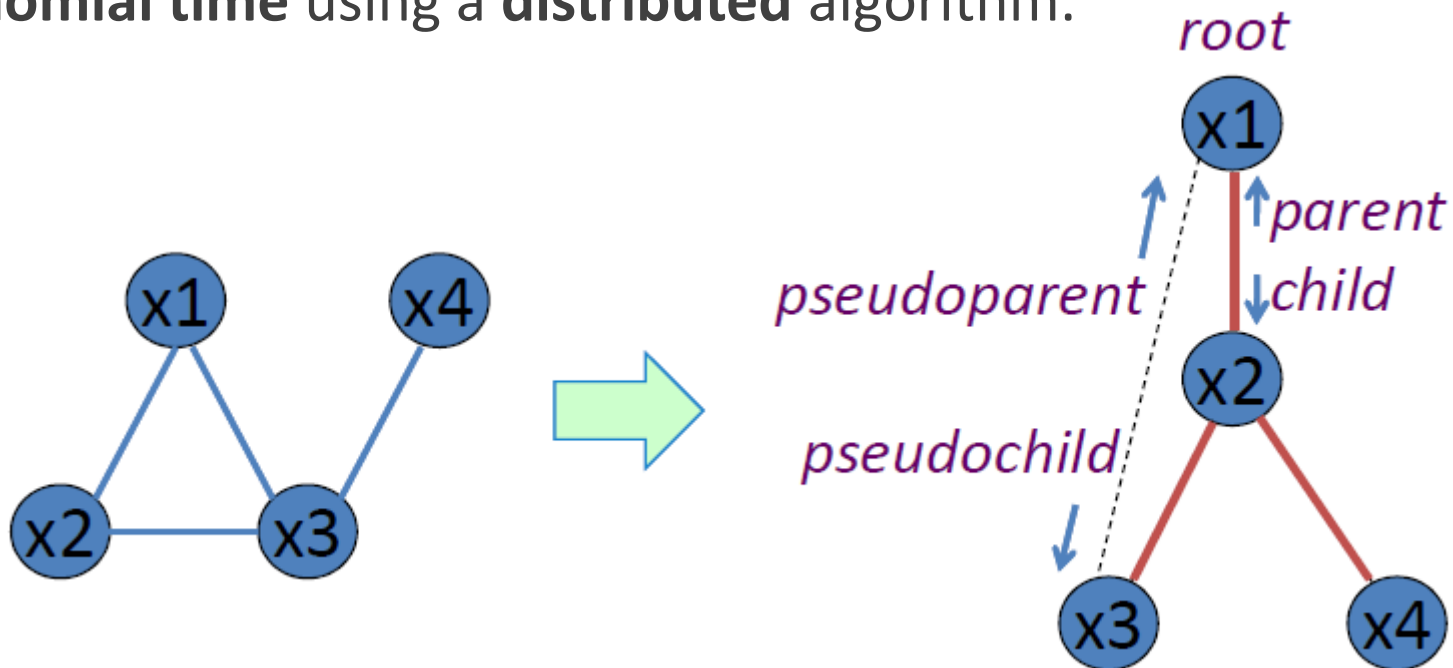
ADOPT: Messages

value$(parent \rightarrow children \cup pseudochildren, a)$:
parent informs its descendants that it has taken value a

cost$(child \rightarrow parent, lower\ bound, upper\ bound, context)$:
child informs parent of the best cost of its assignment; attached context to detect obsolescence;

threshold $(parent \rightarrow child, threshold)$:
minimum cost of solution in child is at least $threshold$

termination $(parent \rightarrow children)$: solution found, terminated

ADOPT: Data Structures

1. **Current context** (agent view):
   list (x_i, v) of values v of higher-priority agents x_i sharing a constrain with x_j

x_i

2. **Bounds:** for each x_j's value d and child x_k
   - lower bounds $\qquad lb(d, x_k)$
   - upper bounds $\qquad ub(d, x_k)$
   - thresholds $\qquad th(d, x_k)$
   - contexts $\qquad \mathcal{C}(d, x_k)$

3. **Threshold** th

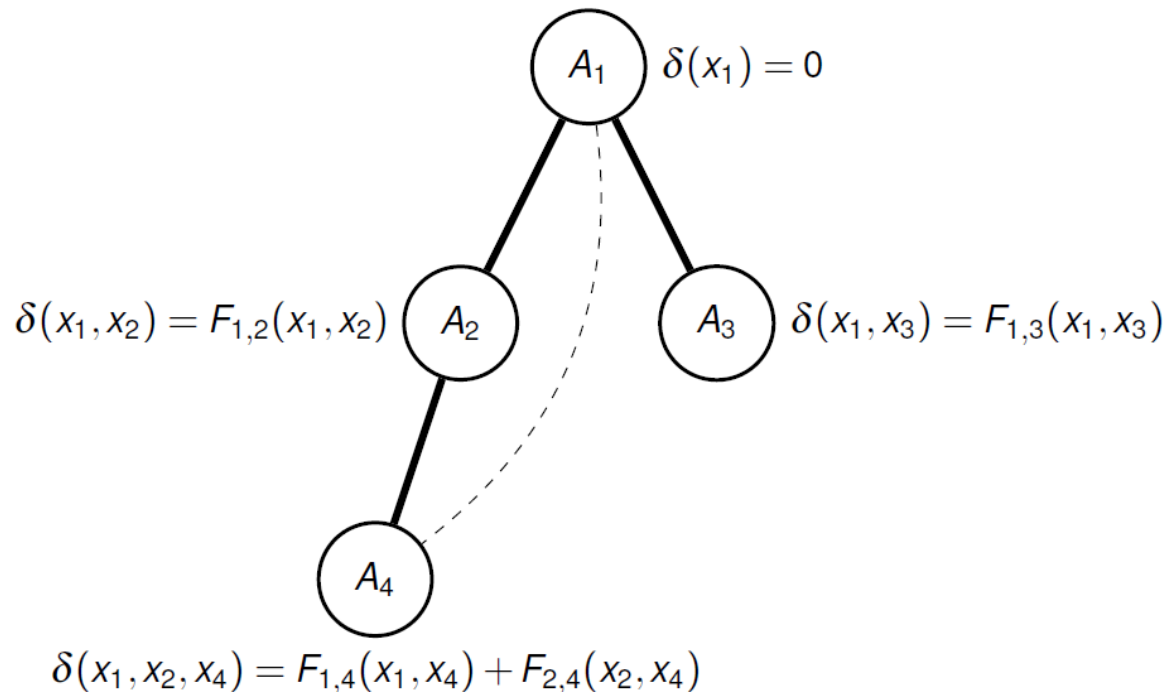Stored contexts must be **active**:
   - left-hand side is satisfied in the current context

If a children's x_k context becomes **obsolete**, it is **removed/reset**, i.e., $lb(., x_k), th(., x_k) \leftarrow 0, ub(., x_k) \leftarrow \infty$
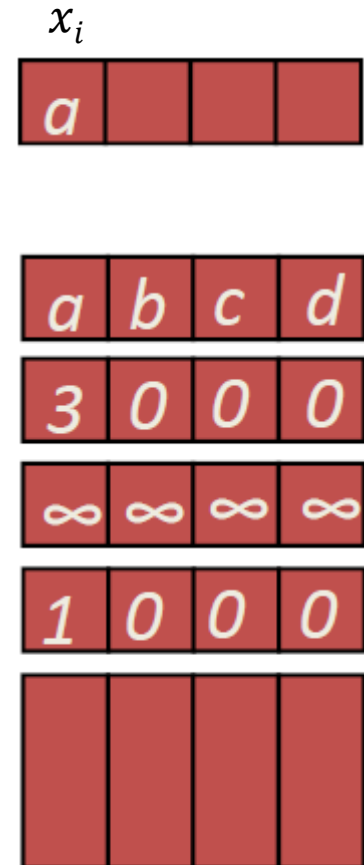
Local Cost Function

The **local cost function** $\delta(x_i)$ for an agent A_i is the **sum** of the values of **constraints** involving only **higher-level** neighbours in the DFS.

A_1 $\delta(x_1) = 0$

$\delta(x_1, x_2) = F_{1,2}(x_1, x_2)$ A_2 A_3 $\delta(x_1, x_3) = F_{1,3}(x_1, x_3)$

A_4

$\delta(x_1, x_2, x_4) = F_{1,4}(x_1, x_4) + F_{2,4}(x_2, x_4)$

partial cost in the current **context** $\mathcal{C}$: $\delta_j(d) = \sum_{(x_i, v) \in \mathcal{C}} F_{ij}(v, d)$

Key Idea: Opportunistic Best First

$$OPT_{x_j}(\mathcal{C}) =$$
$$\min_{d \in d_j}(\delta_j(d) + \sum_{x_k \in children(x_j)} OPT_{x_k}(\mathcal{C} \cup (x_j, d)))$$

i.e. the best value for x_j is a value minimizing the sum of x_j's **local cost** and the **lowest cost of children** under the context extended with the assignment

Bound Computation

OPT_{x_k} values are **incrementally bounded** using $[lb_k, ub_k]$ intervals propagated in **cost** messages

Lower bound computation:

- Each agent evaluates for each possible value of its variable: its **local cost** function with respect to the **current context** adding all the **compatible lower bound messages** received from its children.

- $LB_j(d) = \delta_j(d) + \sum_{x_k \in children(x_j)} lb(d, x_k)$

- $LB_j = \min_{d \in d_j} LB_j(d)$

Upper bound computation:

- $UB(d) = \delta(d) + \sum_{x_k \in children(x_j)} ub(d, x_k)$
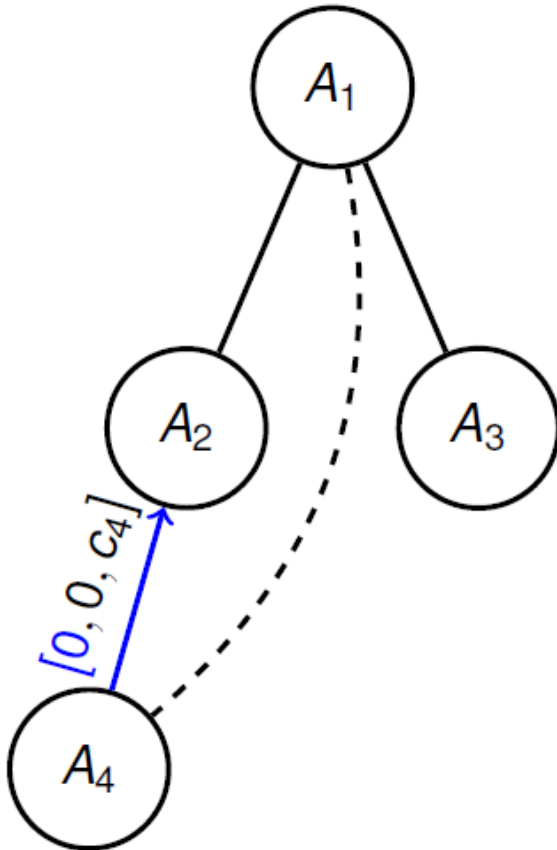
- $UB_j = \max_{d \in d_j} UB_j(d)$

Lower Bound Calculation Example
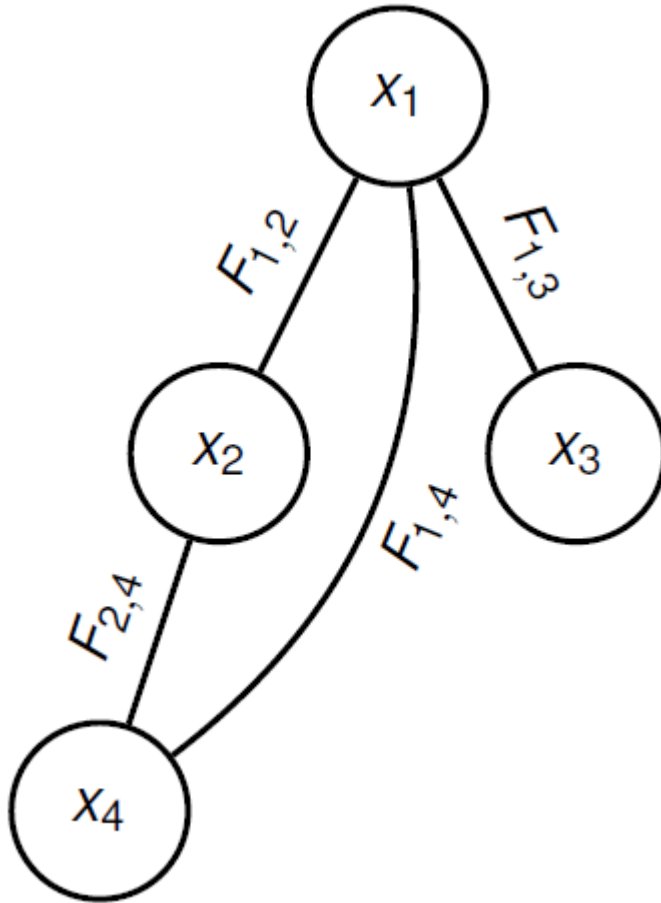
| $F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

Lower Bound Calculation Example

Consider the lower bound in the cost message sent by A_4:

- Recall that A_4 local cost function is:
  $$\delta(x_1, x_2, x_4) = F_{1,4}(x_1, x_4) + F_{2,4}(x_2, x_4)$$
- Restricted to the current context
  $c_4 = \{(x_1 = 0, x_2 = 0)\}$:
  $$\lambda(0, 0, x_4) = F_{1,4}(0, x_4) + F_{2,4}(0, x_4).$$
- For $x_4 = 0$:
  $$\lambda(0, 0, 0) = F_{1,4}(0, 0) + F_{2,4}(0, 0) = 2 + 2 = 4.$$
- For $x_4 = 1$:
  $$\lambda(0, 0, 1) = F_{1,4}(0, 1) + F_{2,4}(0, 1) = 0 + 0 = 0.$$

Then the minimum lower bound across variable values is **LB** $= 0$.

Lower Bound Calculation Example
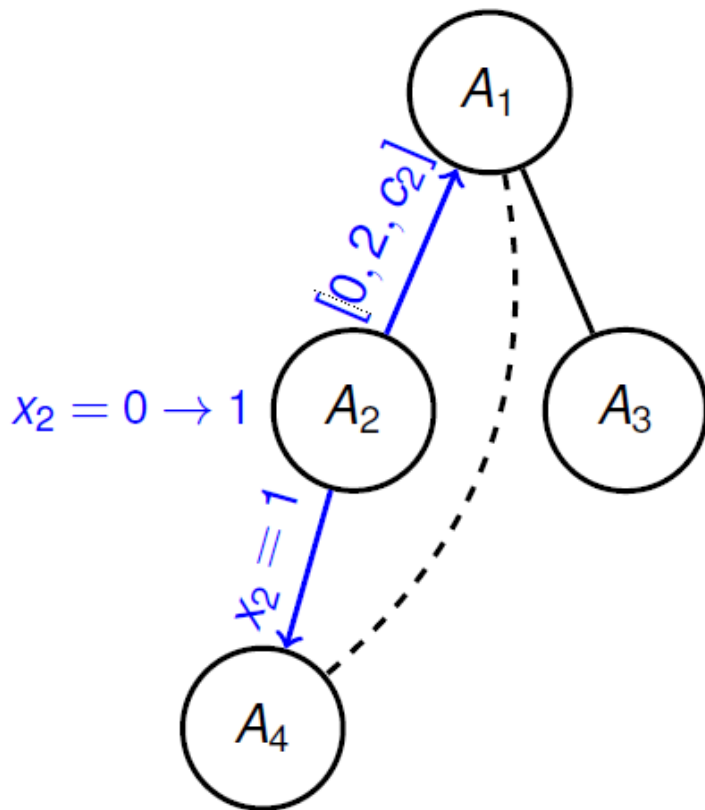
A_2 computes for each possible value of its variable its local function restricted to the current context $c_2 = \{(x_1 = 0)\}$ $(\lambda(0, x_2) = F_{1,2}(0, x_2))$ and adding lower bound message from A_4 (lb).

- For $x_2 = 0$: $LB(x_2 = 0) = \lambda(0, x_2 = 0) + lb(x_2 = 0) = 2 + 0 = 2$.

- For $x_2 = 1$: $LB(x_2 = 1) = \lambda(0, x_2 = 1) + 0 = 0 + 0 = 0$.

A_2 changes its value to $x_2 = 1$ with $LB = 0$.

ADOPT Operation
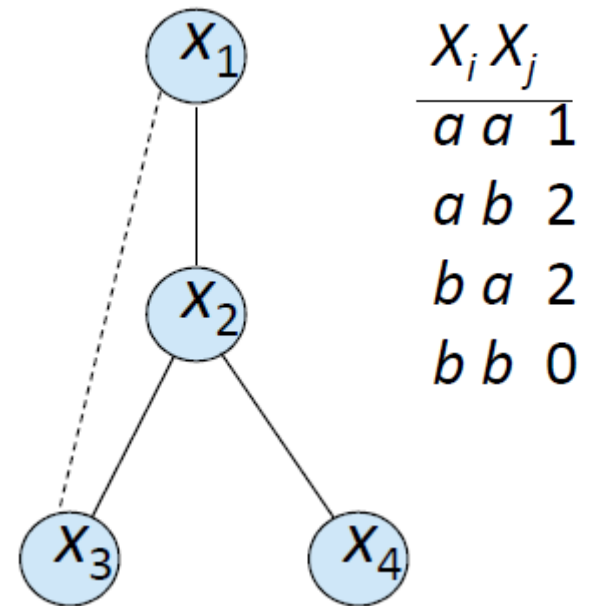
Each time an agent receives a message:

- Processes it:
  - can invalidate context
  - may take a new value minimizing its lower bound
- Sends **value** messages to its children and pseudochildren
- Sends a **cost** message to its parent
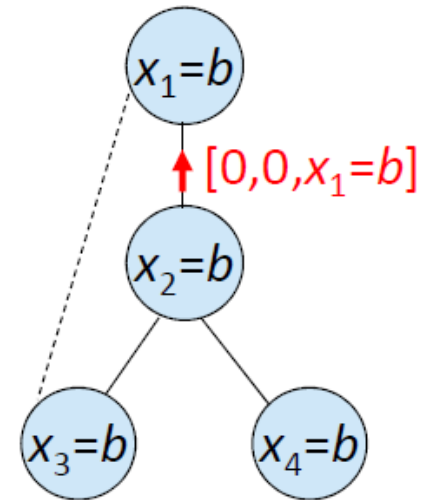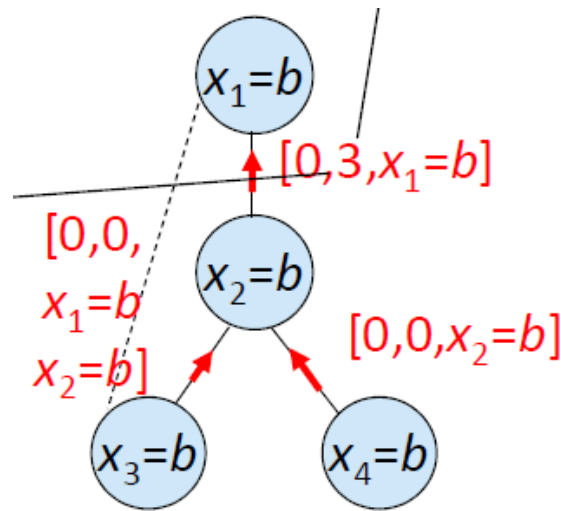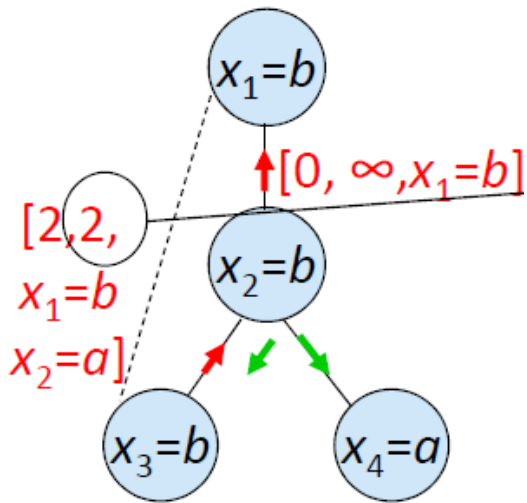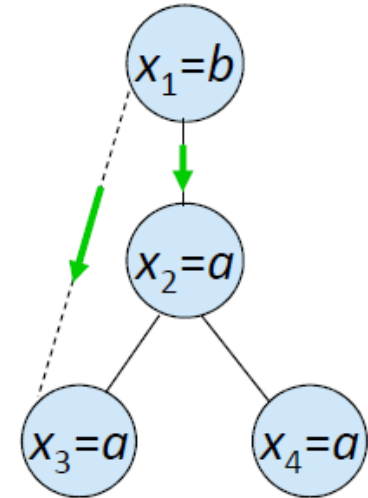- (**threshold** messages)

ADOPT: Example

4 Variables (4 agents) x_1, x_2, x_3 x_4 with
$D = \{a, b\}$

4 binary identical cost functions

Constraint graph:

X_i X_j	
a a	1
a b	2
b a	2
b b	0

ADOPT Example
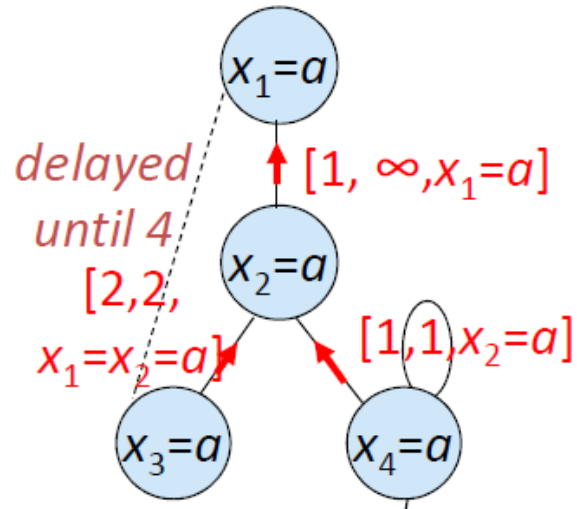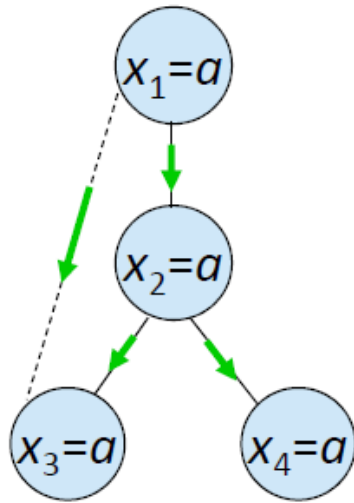
Backtrack Thresholds

The search strategy is based on lower bounds.

Problem
- Lower/upper bounds **only stored** for the **current context**
- **Values abandoned before** proven to be **suboptimal**

Reconstruction of Abandoned Solutions

A_1 changes its value and the context with $x_1 = 0$ is visited again.

- Reconstructing from **scratch** is **inefficient**
- **Remembering** solutions is **expensive**

Detailed cost information **lost** but stored at parent's node in an **aggregated** form.

Can be used for **effective reconstruction** of **abandoned** solutions.

$$x_1 = 0 \rightarrow 1 \rightarrow 0$$
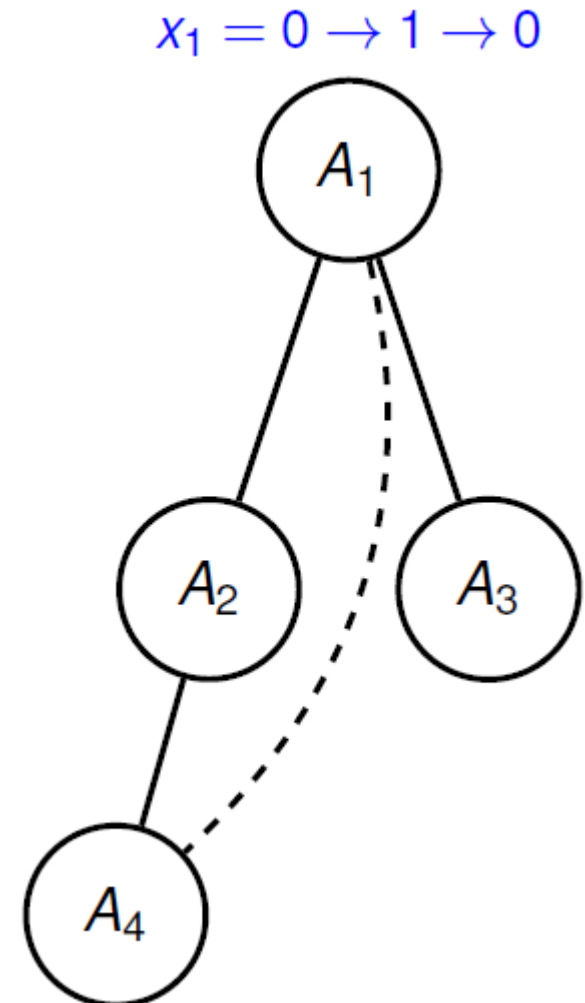
Backend Threshold

Backtrack thresholds: used to speed up the search of previously explored solutions.

- lower bound previously determined by children
- polynomial space

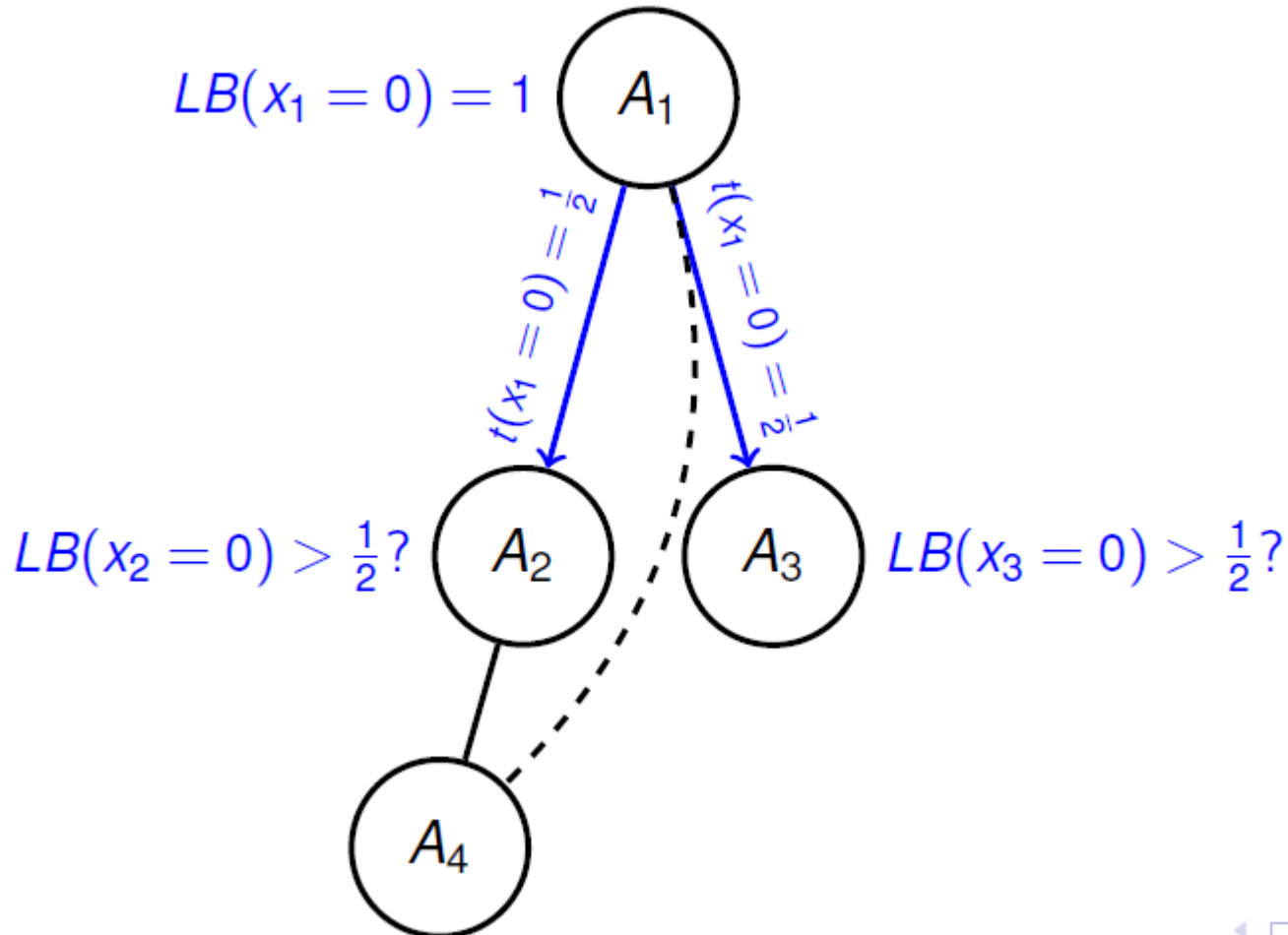Send by parents to a child as **allowance on solution cost**:

- child then **heuristically re-subdivides**, or allocates, the threshold among its own children.
- can be incorrect: **correct** for over-estimates over time as cost feedback is (re)received from the children.

Control backtracking to efficiently search

- **Key point:** do not change value until $LB(currentvalue) > threshold$, i.e., there is a strong reason to believe that current value is not the best (wait until having accumulated enough cost messages)

Backend Threshold: Example

A child agent will not change its variable value so long as **cost is less** than the **backtrack threshold** given to it by its parent.

$LB(x_1 = 0) = 1$ — A_1

$t(x_1 = 0) = \frac{1}{2}$

$t(x_1 = 0) = \frac{1}{2}$

$LB(x_2 = 0) > \frac{1}{2}?$ — A_2

A_3 — $LB(x_3 = 0) > \frac{1}{2}?$

A_4

Threshold Reballancing

Parent **distributes** the accumulated bound among children and **corrects subdivision** as feedback is received from children
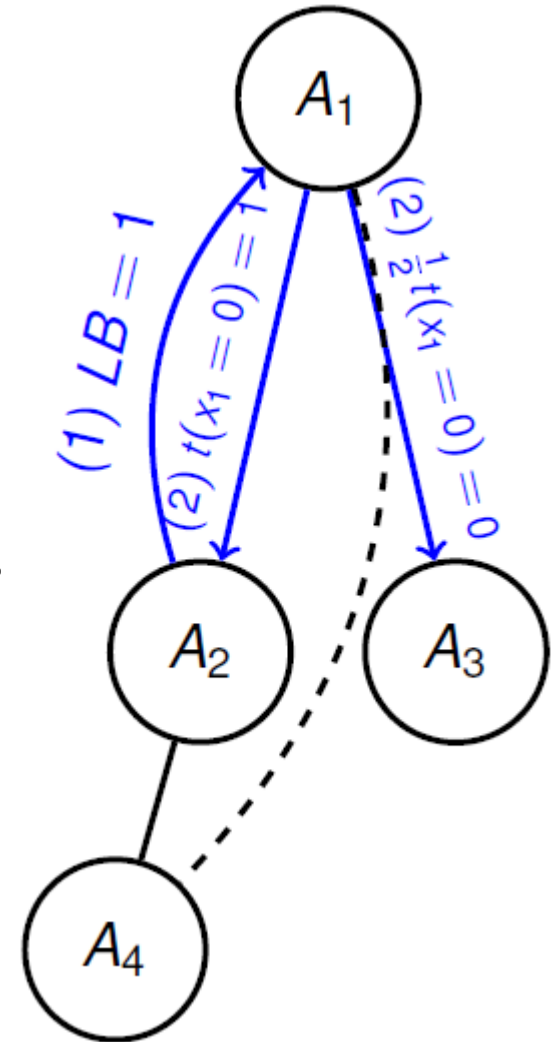
Maintain invariants:

- **allocation invariant**: the threshold on cost for x_j must equal the local cost of choosing d plus the sum of the thresholds allocated to x_j's children.

- **child threshold invariant**: The threshold allocated to child x_k by parent x_j cannot be less than the lower bound or greater than the upper bound reported by x_k to x_j .

Reballancing

When A_1 receives a new lower bound from A_2 **rebalances** thresholds.

A_1 **resends** threshold messages to A_2 and A_3.

Diagram showing A_1 at top connected to A_2 and A_3 below, with A_4 at the bottom. Blue edge labels: $(1)\ LB = 1$, $(2)\ t(x_1 = 0) = 1$, $(2)\ t(x_1 = 0) = 0$, $(2)\ \frac{1}{2}t(x_1 = 0) = 0$.
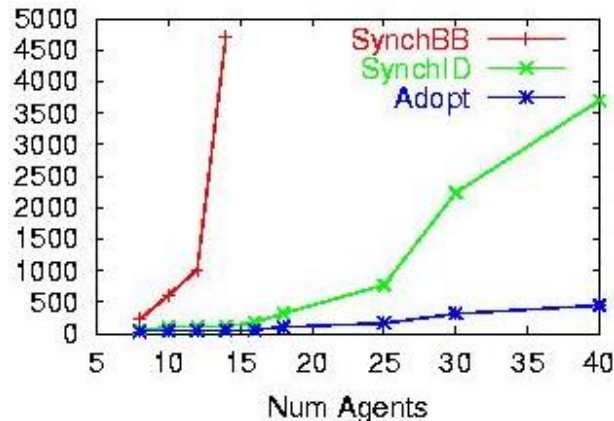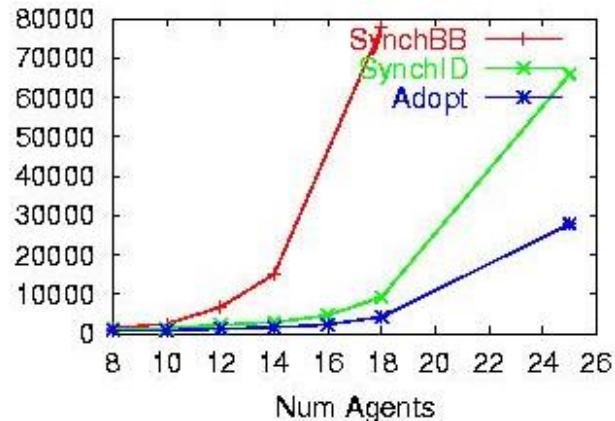
ADOPT Properties

For finite DCOPs with binary non-negative constraints, ADOPT is **guaranteed to terminate** with the **globally optimal solution.**
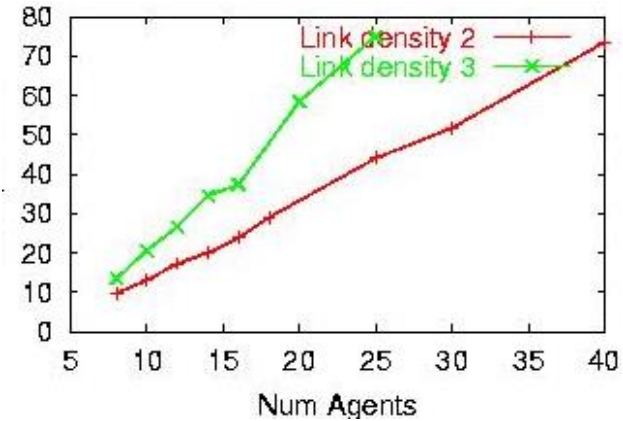
Performance on Graph Coloring

Avg. number of cycles, link density = 2

Avg. number of cycles, link density = 3

Avg. messages per cycle

- ADOPT's lower bound search method and parallelism yields significant efficiency gains.

- Sparse graphs (density 2) solved **optimally and efficiently** by ADOPT.

- Communication only grows linearly
  - thanks to the sparsity of constraint graph

ADOPT Approximation

ADOPT can be used for finding **suboptimal** solutions with **guaranteed error** bound b.

Terminate when **lower bound** at the **root** get within b of the **upper bound**.

Using error bound, **less** of the solution space **explored** ➔ADOPT is able to find a solution faster, thereby providing a method to **trade-off computation time** for **guaranteed** solution **quality**.

Adopt summary – Key Ideas

Optimal, **asynchronous** algorithm for DCOP
- **polynomial space** at each agent

Weak Backtracking
- **lower bound** based search method
- Parallel search in independent subtrees

Efficient reconstruction of abandoned solutions
- **backtrack** thresholds to control backtracking

Bounded error approximation
- sub-optimal solutions **faster**
- **bound** on worst-case performance

Approximation Algorithms

Distributed Constraint Optimization

Why Approximate Algorithms

Optimality in practical applications often not achievable

Approximate algorithms

- sacrifice optimality in favour of computational and communication efficiency
- well-suited for **large-scale** distributed applications:
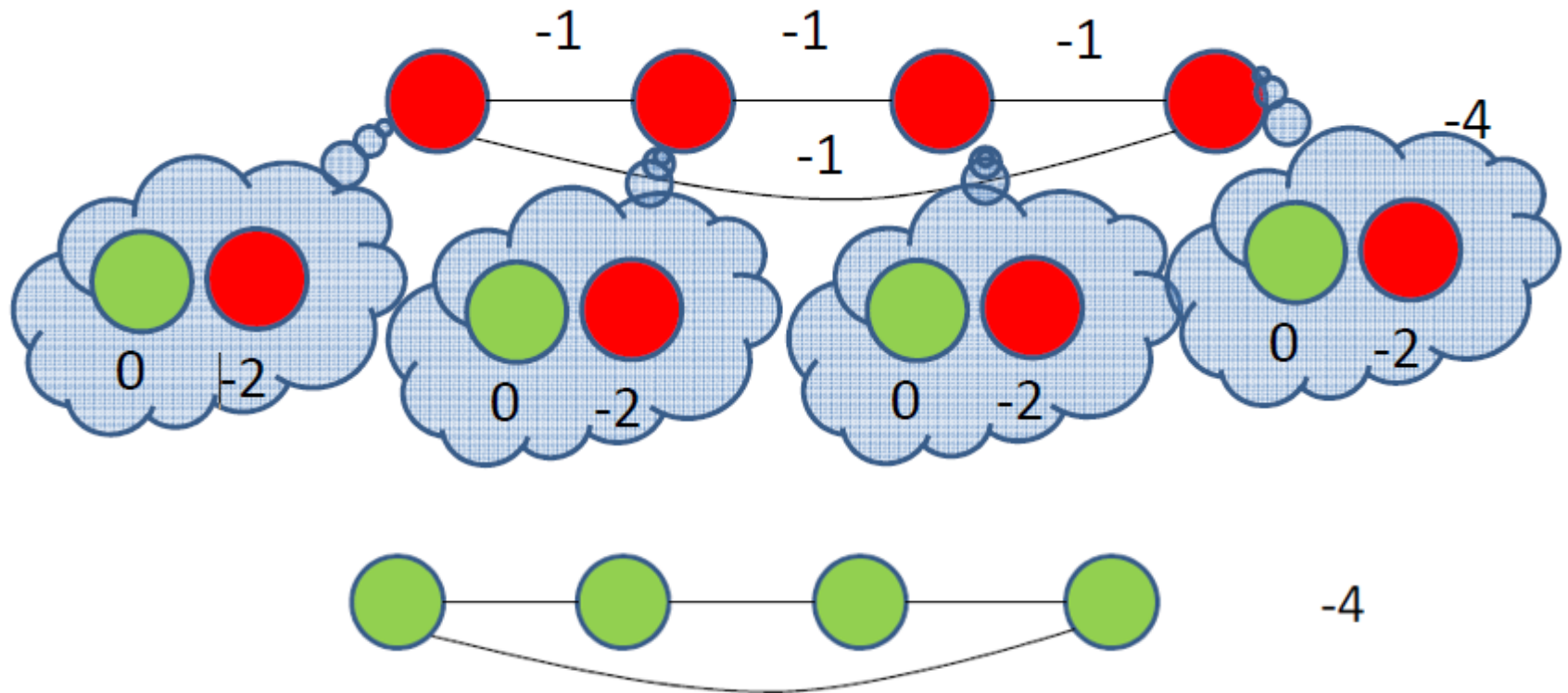
Centralized Local Greedy approaches

Start from a **random** assignment for all the variables

Do **local moves if** the new assignment **improves** the value (local gain)

Local: changing the value of a **small set** of variables (in most case just one)

The search **stops** when there is **no local move** that provides a **positive gain**, i.e., when the process reaches a local maximum.

Issues with Distributed Local Greedy Algorithms

Parallel execution: A greedy local move might be harmful/useless

➔ Need coordination

Issues with Distributed Local Greedy Alg

When operating in a decentralized context:

Problem: Out-of-date local **knowledge**

- Assumption that other agents do not change their values
- A greedy local move might be harmful/useless

Solution:

- **Stochasticity** on the decision to perform a move (**DSA**)
- **Coordination** among neighbours on who is the agent that should move (**MGM**)

Distributed Stochastic Algorithm (DSA)

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 neighbours

Each agent:

- Generates a random number and **executes** only if it is less than **activation probability**
- When executing choose a value for the variable such that the **local gain** is **maximized**
- **Communicate** and receive possible **variables change** to/from neighbours

DSA-1: Execution Example

rnd > ¼ ?　　rnd > ¼ ?　　rnd > ¼ ?　　rnd > ¼ ?

-1　　-1　　-1

-1

0　　-2

P = 1/4
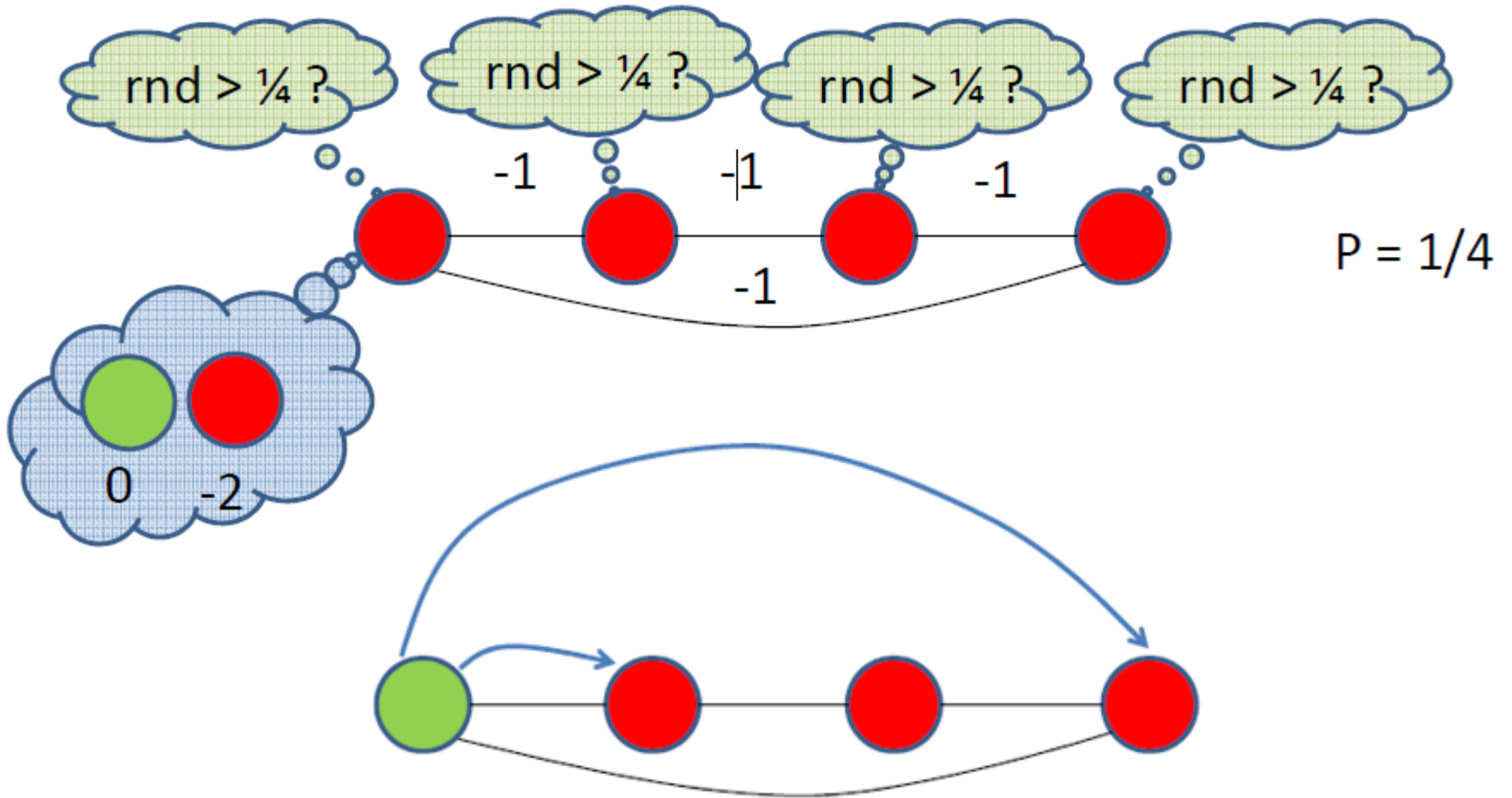
DSA-1: Discussion

Extremely **low computation/communication**

Good performance in various domains
- e.g. target tracking [Fitzpatrick Meertens 03, Zhang et al. 03],
- Shows an **anytime property** (not guaranteed)
- Benchmarking technique for coordination

Problems with the **activation probability**
- must be **tuned** [Zhang et al. 03]
- **domain-dependent**: no general rule, hard to characterise results across domains

Maximum Gain Message (MGM-1)

Coordinate among neighbours to decide which **single agent** is going to move

Initialize agents with a **random assignment** and **communicate** values to **neighbours**

Each agent:
- **Compute** and **exchange** possible **gains**
- **Agent** with **maximum** (positive) **gain executes**
- **Communicate** and **receive** possible **variables changes** to/from neighbours

MGM-1: Example
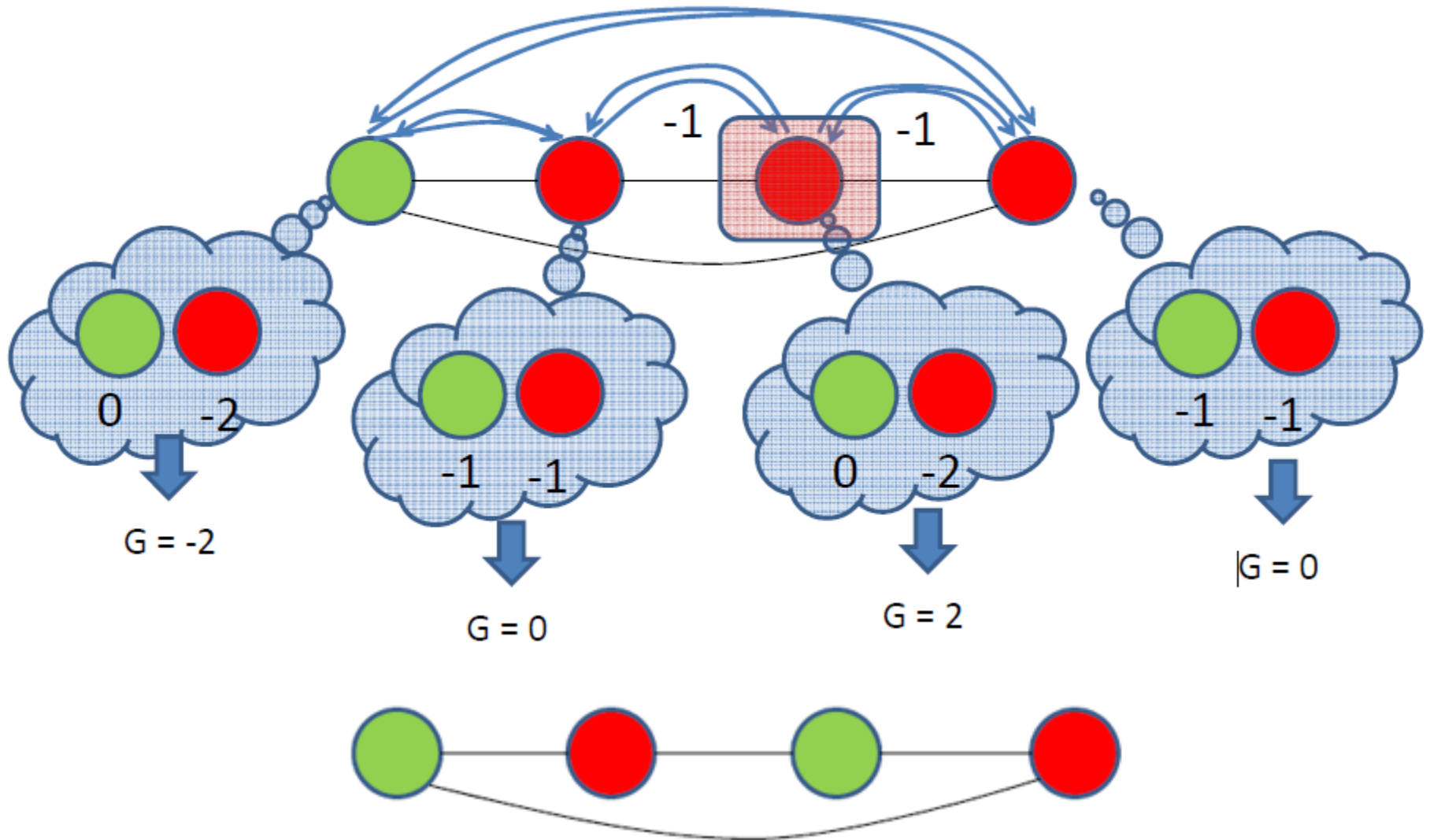
MGM-1 Discussion

More communication than DSA (but still linear)

- Empirically, similar to DSA

No threshold to set

- Does **not require** any **parameter tuning**.

Guaranteed to be monotonic (Anytime behavior)

Local Greedy Approaches

Very **little memory** and **computation**.

Anytime behaviours.

But: Could result in very bad solutions (no **guarantees**)
- local maxima arbitrarily far from optimal.

Quality Guarantees for Approximation Techniques

Key **area of research**

Address **trade-off** between **guarantees** and **computational** effort

Particularly important for:
- **dynamic** settings
- severe constrained resources (e.g. embedded devices)
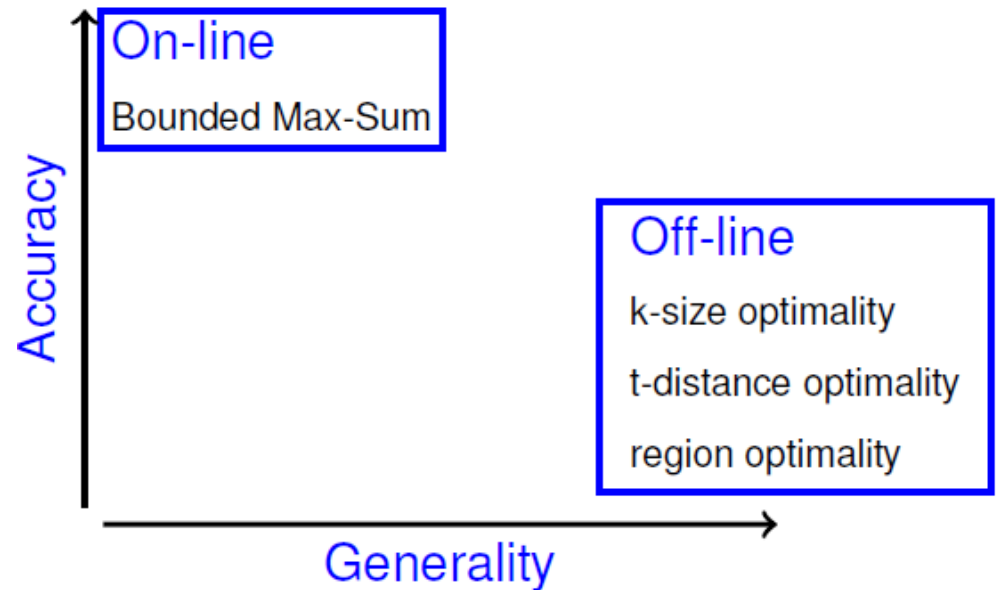- safety **critical** applications (e.g. search and rescue)

Categories of Quality Guarantees

Off-line

- Prior running the algorithm
- Not tied to specific problem instances

On-line

- After running the algorithm
- On the particular problem instance

Summary

FRODO: a FRamework for Open/Distributed Optimization

Framework for **experimental evaluation** of DCSP/DCOP algorithms

Input
- files defining optimization problems to be solved (in XCSP 2.1 format)
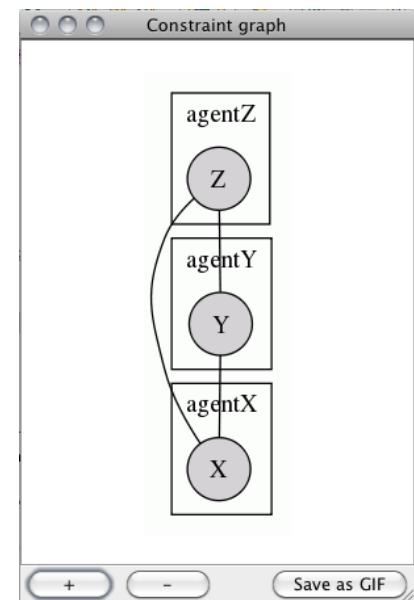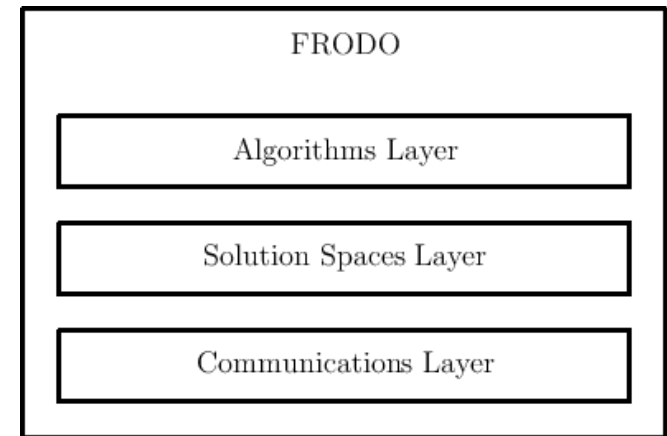- configuration files defining the algorithm to be used to solve them

Algorithms implemented
- SynchBB, MGM and MGM-2, ADOPT, DSA, DPOP, S-DPOP, MPC-Dis(W)CSP4, O-DPOP, AFB, MB-DPOP, Max-Sum, ASO-DPOP, P-DPOP, P^2-DPOP, E[DPOP], Param-DPOP, and $P^{3/2}$-DPOP

Supports various **performance metrics**
- numbers and sizes of messages sent
- Non-Concurrent Constraint Checks
- simulated time

http://frodo2.sourceforge.net/

FRODO
Algorithms Layer
Solution Spaces Layer
Communications Layer

Constraint graph
agentZ
Z
agentY
Y
agentX
X
+   −   Save as GIF

Conclusion

Distributed constraint optimization generalizes distributed constraint satisfaction by allowing real-valued constraints

Both **complete** and **approximate** algorithms exist

- complete can require exponential number of message exchanges (in the number of variables)
- approximate can return (very) suboptimal solutions

Very active areas of research with a lot of progress – new algorithms emerging frequently

Reading: [Vidal] – Chapter 2; ADOPT: asynchronous distributed constraint optimization with quality guarantees; IJCAI 2011 Optimization in Multi-Agent Systems tutorial, Part 2: 37-61min and Part 3: 0-38min