

## Distributed Constraint Programming

Branislav Božanský and Michal Pěchouček

Artificial Intelligence Center,  
Department of Computer Science,  
Faculty of Electrical Engineering,  
Czech Technical University in Prague  
*branislav.bosansky@agents.fel.cvut.cz*

December 6, 2016

Previously ... on multi-agent systems.

- 1 Distributed Constraint Satisfaction Programming

# Constraint Network

## Definition

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

- $X = x_1, \dots, x_n$  is a set of variables;
- $D = \{D_1, \dots, D_n\}$  is a set of variable domains, which enumerate all possible values of the corresponding variables; and
- $C = \{C_1, \dots, C_m\}$  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

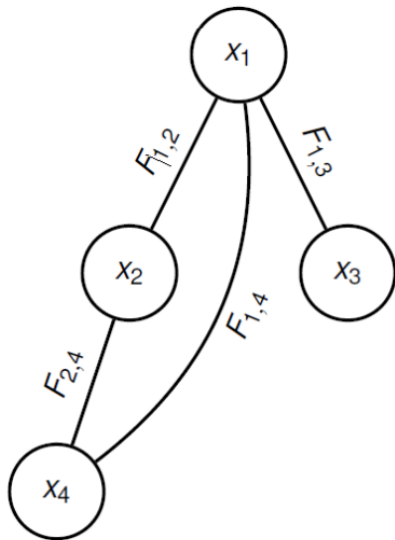
Hard constraint  $C_i^h$  is a Boolean predicate  $P_i$  that defines valid joint assignments of variables in the scope

$$P_i : D_1^i \times \dots \times D_{r_i}^i \rightarrow \{F, T\}$$

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

$$F_i : D_1^i \times \dots \times D_{r_i}^i \rightarrow \mathbb{R}$$

# Binary Constraint Networks



# Synchronous Branch-and-Bound

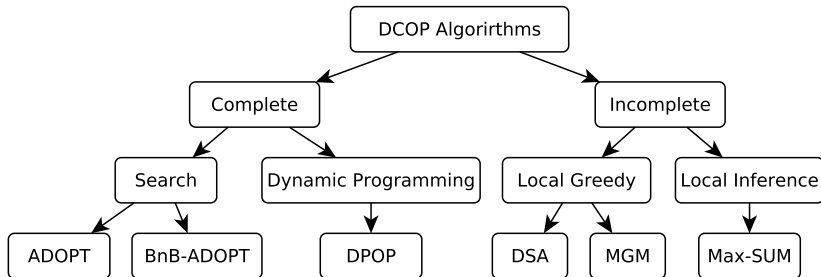
Agents agree on an variable order and repeat:

- 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  $\rightarrow$  if lower-bound (now the total cost)  $<$  upper bound, then upper bound = lower bound; remember best so far assignment
- 6 if  $k < 1$ : stop  $\rightarrow$  return best so far assignment.

# Asynchronous Backtracking Algorithm (ABT) – assumptions

- Agents communicate by sending messages
- An agent can send messages to others, iff it knows their identifiers (directed communication / no broadcasting)
- The delay transmitting a message is finite but random
- For any pair of agents, messages are delivered in the order they were sent
- Agents know the constraints in which they are involved, but not the other constraints
- Each agent owns a single variable (agents = variables)
- Constraints are binary (2 variables involved)

# Types of Asynchronous Algorithms for DCOPs





# ADOPT: Asynchronous Distributed OPTimization<sup>1</sup>

First *asynchronous 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).

Theorem (Modi et. al 2005)

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

<sup>1</sup>Modi et al. "Adopt: asynchronous distributed constraint optimization with quality guarantees" AIJ 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.

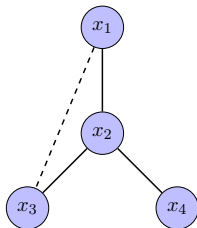
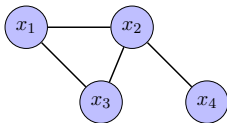
It then tells the children to look for a solution but ignore any partial solution whose cost is above the lower bound because it already knows that it can get that lower cost.

# 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
- also termed pseudochildren and pseudoparent of a 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): a child informs a parent of the best cost of its assignment; attached context to detect obsolescence;
- **threshold**(parent  $\rightarrow$  children, threshold): minimum cost of solution in child is at least threshold;
- **termination**(parent  $\rightarrow$  children): solution found, terminate

# ADOPT: Data Structures

Each agent  $x_j$  stores the following data:

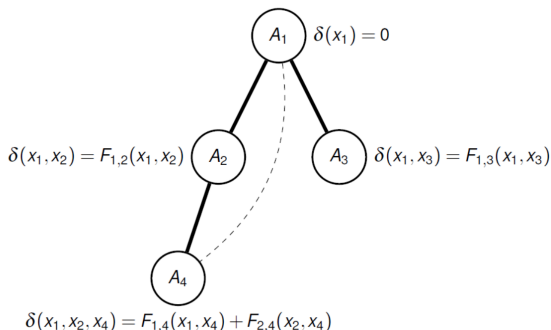
- 1 Current context** (agent view): list  $(x_i, v)$  of values  $v$  of higher-level agents  $x_i$  sharing a constraint with  $x_j$
- 2 Bounds:** for each value  $d$  and each child  $x_k$ 
  - lower bounds  $lb(d, x_k)$
  - upper bounds  $ub(d, x_k)$
  - thresholds  $t(d, x_k)$
  - contexts  $c(d, x_k)$
- 3 Threshold**

Stored contexts must be active  $\rightarrow$  left-hand side is satisfied in the current context.

If a child's  $x_k$  context becomes obsolete, it is reset, i.e.,  
 $lb(., x_k), t(., x_k) \leftarrow 0, ub(., x_k) \leftarrow \infty.$

# 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 neighbors in the DFS.



## Key Idea: Best First Search

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

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.

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

# Bound Computation

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

Similarly for upper bound:

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



# ADOPT Steps

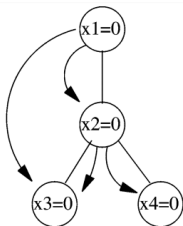
Each time an agent receives a message:

- 1 process the message
  - the message can invalidate the current context
  - may take a new value minimizing its lower bound
- 2 Sends value messages to its children and pseudochildren
- 3 Sends a cost message to its parent
- 4 Eventually sends threshold messages

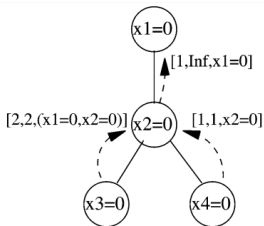
The search strategy is based on lower bounds.

- Each agent adopts the value with minimal lower bound.
- Lower/upper bounds only stored for the current context.
- Values abandoned before proven to be suboptimal.

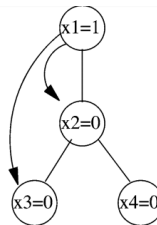
# ADOPT Example



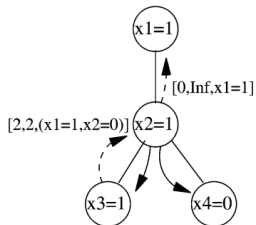
(a)



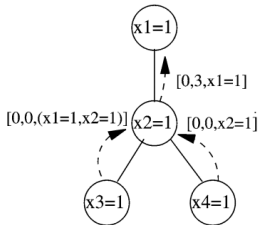
(b)



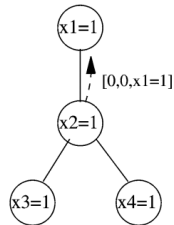
(c)



(d)



(e)



(f)

# Threshold messages

The algorithm can re-visit previously abandoned partial solutions.

However:

- Reconstructing from scratch is inefficient
- Remembering solutions is expensive (in terms of memory)

Detailed cost information lost but stored at parents node in an aggregated form.

Can be used for effective reconstruction of abandoned solutions.

# Thresholds

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  $LB(\text{current\_value}) > \text{threshold}$ , i.e., there is a strong reason to believe that current value is not the best (wait until having accumulated enough cost messages)

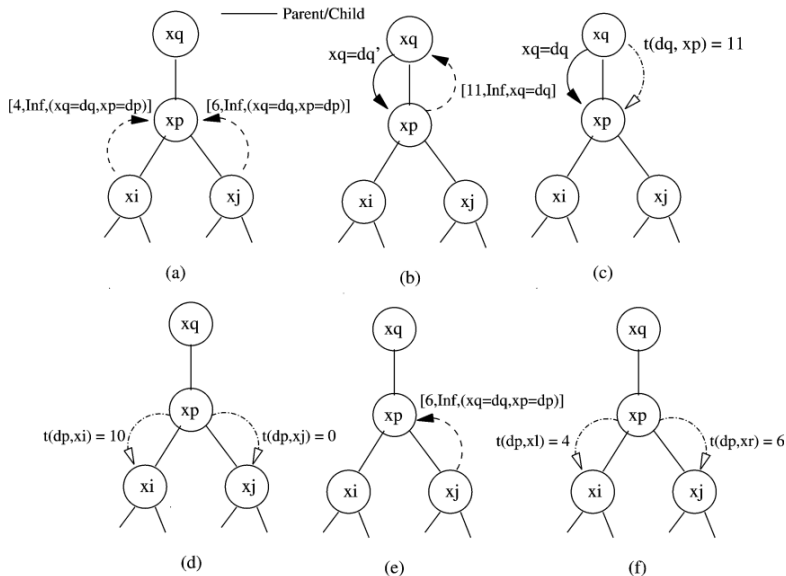
# Threshold Re-balancing

Parent distributes the accumulated bound among children and corrects subdivision as feedback is received from children.

ADOPT 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  $x_j$ .

# Backend Threshold: Example



# Approximate Algorithms

Optimality in practical applications often not achievable.

Approximate algorithms:

- sacrifice optimality in favor of computational and communication efficiency
- well-suited for large-scale distributed applications

NOTE: In the following, we assume the maximization version of DCOPs.

# Local Search Approaches

Start from a random assignment for all the variables

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

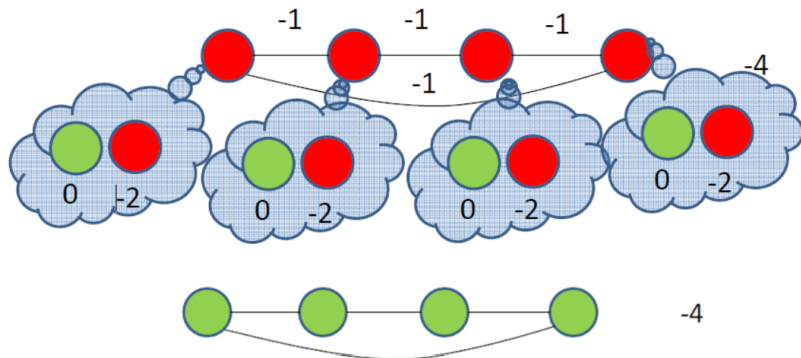
Local: changing the value of a small set of variables (in most case just one)

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



# Local Search Approaches

We need a coordination among agents:



# Local Search Approaches

- *Randomize to decide whether an agent is going to act.*
  - DSA-1 algorithm
  - Generates a random number and executes only if it is less than *an activation probability*.
  
- *Negotiate with neighbors.*
  - MGM-1 algorithm
  - Agents compute and exchange possible gains and only the with maximum (positive) gain executes the action.

# FRODO: a FFramework for Open/Distributed Optimization

Framework for experimental evaluation of DCSP/DCOP algorithms.

Input:

- files defining optimization problems to be solved
- configuration files defining the algorithm to be used to solve them

Many implemented algorithms:

- SynchBB, MGM and MGM-2, ADOPT, DSA, DPOP, SDPOP, MPC-Dis(W)CSP4, O-DPOP, AFB, MB-DPOP, Max-Sum, ASO-DPOP, P-DPOP, P<sup>2</sup>-DPOP, E[DPOP], Param-DPOP, and P <sup>$\frac{3}{2}$</sup> -DPOP

Supports various performance metrics:

- numbers and sizes of messages sent
- Non-Concurrent Constraint Checks
- simulated time

<https://sourceforge.net/projects/frodo2/>