Special Topics in AI: Intelligent Agents and Multi-Agent Systems

Distributed Constraint Optimization (Exact approaches, DPOP)

Alessandro Farinelli

Outline

- Introduction
  - DCOP for MAS
  - how to model problems in the DCOP framework
- Solution Techniques for DCOPs
  - Exact algorithms (DCSP, DCOP)
    - DPOP
  - Heuristics and approximate algorithms (without/with quality guarantees)
    - DSA, MGM, Max-Sum; k-optimality, bounded max-sum

Working together

Coordination problem:

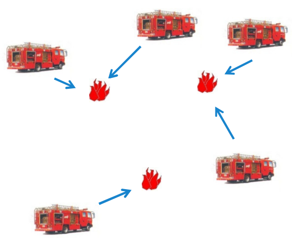
Choose agent's <u>individual actions</u> so to maximise a <u>system-wide objective</u>

Task allocation:

<u>individual actions</u>: which fire to tackle

<u>system-wide objective</u>: minimise total extinguish time

<u>solution</u>: a joint action



Decentralised Coordination

- <u>Decentralised coordination:</u> Local decision with local information
- Why Decentralised coordination ?
  - <u>In general no benefit for computation or solution quality</u>
  - Robustness
    - avoid single point of failure
  - Scalability
    - Not enough bandwidth to communicate/process all information
  - <u>Leads to problem decomposition</u>
    - Each agent cares only of local neighbours

DCOPs for Decentralized Coordination

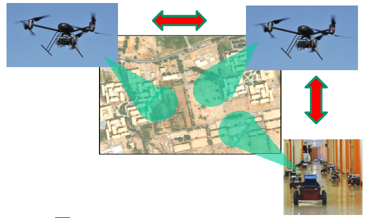
Why DCOPs for decentralized coordination ?

- Well defined problem
  - Clear mathematical formulation that captures most important aspects
  - Many solution techniques
    - Optimal: ABT, ADOPT, DPOP, …
    - Heuristics: DSA, MGM, Max-Sum, …
- Solution techniques can handle large problems
  - compared for example to sequential dec. Making (MDP, POMDP)

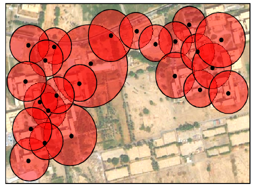
Reference Applications

Incident Management

Cooperative Exploration



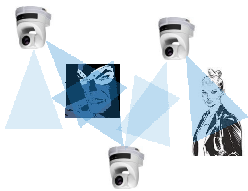
Environment monitoring



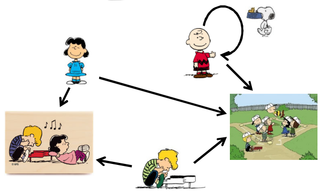
Energy management

Modeling Problems as DCOP

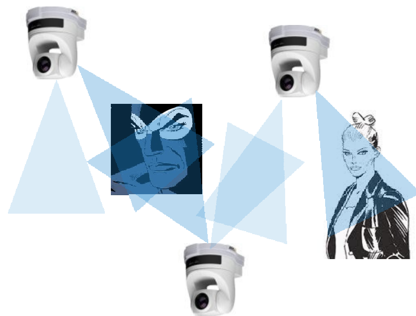
- Surveillance



- Meeting Scheduling



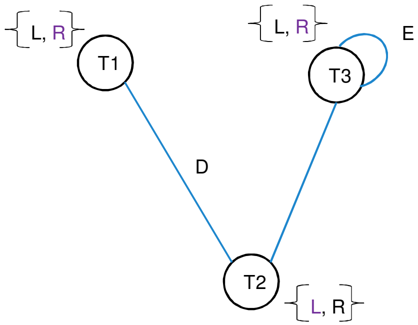
Target Tracking



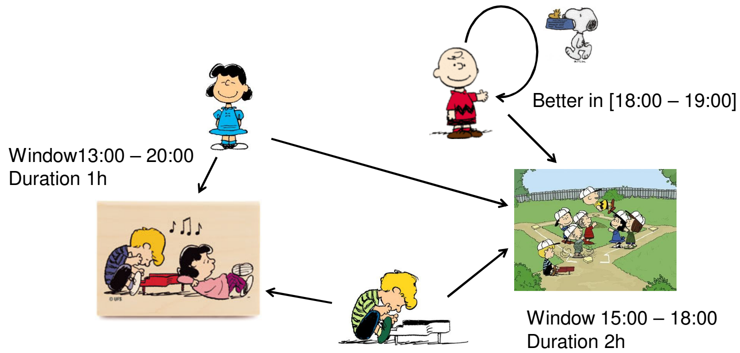
- Why decentralize
  - Robustness to failure and message loss

Target Tracking - DCOP

- Variables -> Cameras
- Domains -> Camera actions
  - look left, look right
- Constraints
  - Overlapping cameras
  - Related to targets
    - Diabolik, Eva
- Maximise sum of constraints

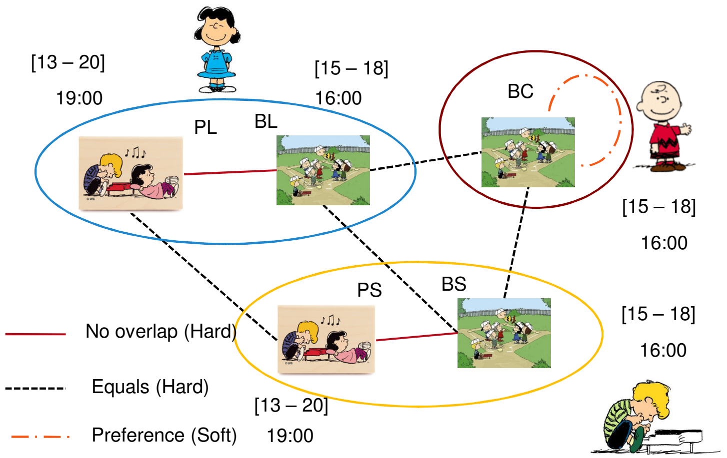


Meeting Scheduling



- Why decentralize
  - Privacy

Meeting Scheduling - DCOP



Constraint Networks

$X = \{X_1, \ldots, X_n\}$ a set of variables (e.g. meetings)

$D = \{D_1, \ldots, D_n\}$ a set of discrete variable domains (e.g. time slots)

$C = \{C_1, \ldots, C_m\}$ a set of constraints (e.g., equality, non overlap,)

$S_i \subseteq X$ Scope of constraint C_i

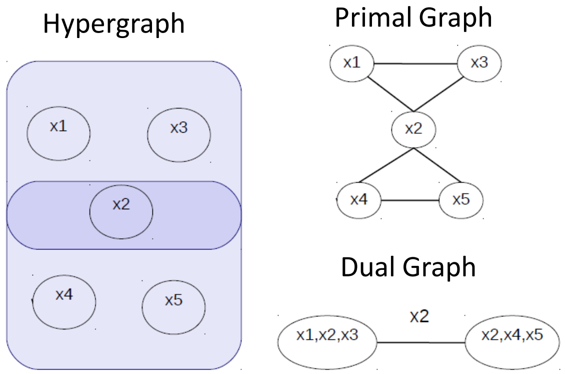
Hard constraints

R_i	x_j	x_k
	0	1
	1	0

Soft constraint

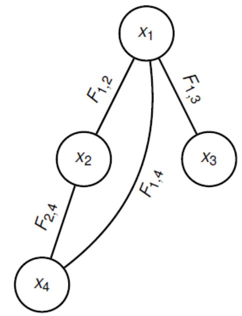
F_i	x_j	x_k
2	0	0
0	0	1
0	1	0
1	1	1

Graphical Representation



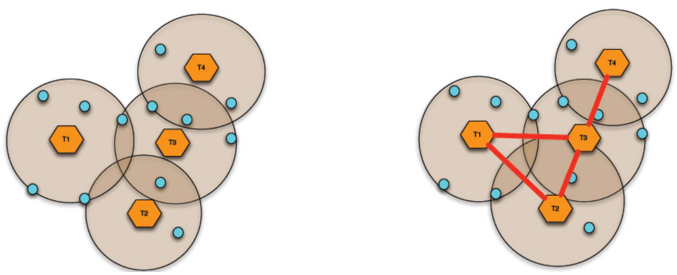
Binary constraint networks

- Each constraint is defined over two variables
- Every constraint network can be mapped to a binary constraint network but
  - Addition of variables/constraints
  - Add complexity



Constraint graph

- Link between two variables if they share at least one constraint (i.e., primal graph)
  - In general, constraint graph ≠ constraint network



Objectives for constraint networks

- Constraint Satisfaction Problem (**CSP**)
  - Objective: find an assignment for all the variables in the network that satisfies all constraints
- Constraint Optimization Problems (**COP**)
  - Objective: find an assignment for all the variables in the network that satisfies all constraints and optimizes a global objective function

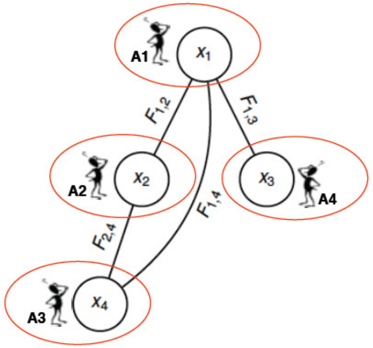
$$X^* = \arg\max_X \left(\sum_i F_i(X_i) \right)$$

Global function: an aggregation (i.e., sum) of local functions $F_i(X_i)$

Distributed Constraint Reasoning

In a decentralized context:

- Agents control Variables
- Agents communicate to solve the problem

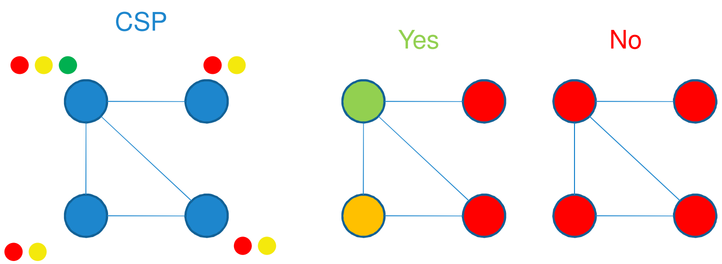


Benchmarking problems

- Motivations
  - Analysis of complexity and optimality is not enough
  - Need to empirically evaluate algorithms on the same problem
- Graph coloring
  - Simple to formalise very hard to solve
  - Well known parameters that influence complexity
    - Number of nodes, number of colors, density (number of link/number of nodes)
  - Many versions of the problem
    - CSP, MaxCSP, COP

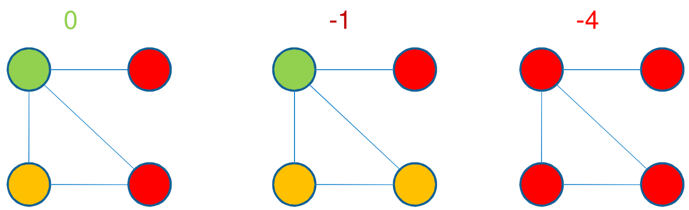
Graph Coloring

- Network of nodes
- Nodes can take on various colors
- Adjacent nodes should not have the same color
  - If it happens this is a conflict



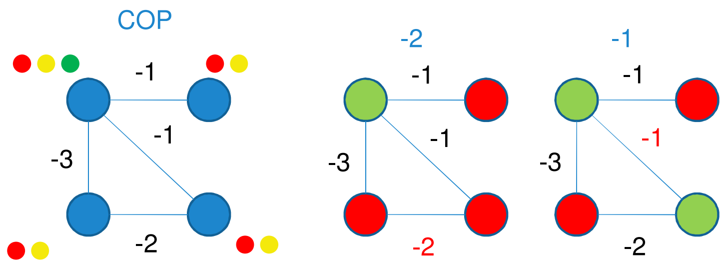
Graph Coloring - MaxCSP

- Optimization Problem
- Natural extension of CSP
- Minimise number of conflicts



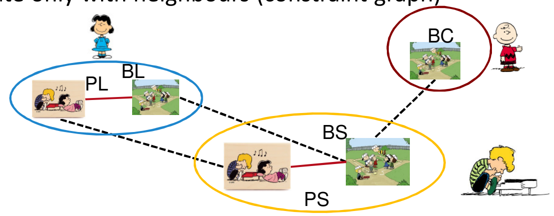
Weighted Graph Coloring - COP

- Optimization Problem
- Conflicts have a weight
- Maximise the sum of weights of violated constraints



Distributed COP

- We focus on optimization
- DCOP = Constraint Network + Agents $A = \{A_1, \ldots, A_k\}$
- Where each agent:
  - Controls a subset of the variables (typically just one)
  - Is only aware of constraints that involve the variables it controls
  - Communicate only with neighbours (constraint graph)



Performance measures

- Solution quality
  - Optimality not always achievable,
  - Optimality Guarantees
- Coordination Overdead
  - Computation: computation effort (time complexity)
  - Communication: number and **size** of messages (network load)
- Desirable properties (hard to quantify)
  - Robustness to failures, parallelism, flexibility, privacy maintenance, etc.

DCOP Solution techniques

- Exact approaches
  - Guarantee optimal solution
  - Exponential coordination overhead
  - ADOPT, DPOP, OptAPO
- Heuristics
  - Low coordination overhead
  - No guarantees on optimality
  - DSA, MGM, Max-Sum
- Approximate approaches
  - Low coordination overhead
  - Optimality guarantees
  - Bounded max-sum, k-optimality

Exact Approaches I

- ADOPT (Search based) [Modi et al 05]
  - Distributed branch and bound
  - Partial order based on a DFS search (pseudotree)
  - Asynchronous (high parallelism, flexible)
  - Number of messages exponential in number of agents
- Small messages but exponentially many

Exact Approaches II

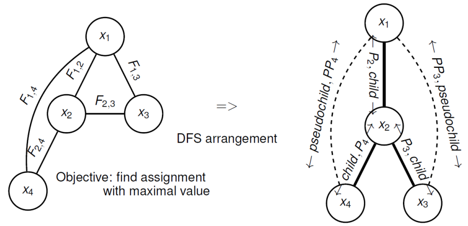
- DPOP (Inference) [Petcu and Faltings 07]
  - Distributed Bucket Elimination
  - Partial order based on a DFS search (pseudotree)
  - Linear number of messages
  - Exponential message size (in width of DFS search tree)
  - DFS-tree width tipically much less than number of agents
- Few messages but exponentially large

Dynamic Programming Optimization Protocol

1. DFS-tree building (special case of Pseudo tree)
   - Constraint graph → DFS-Tree
   - Token passing
2. Utility propagation
   - Compile information to compute optimal value
   - Util messages from leaves to root
3. Value Propagation
   - Root chooses optimal value and propagate decision
   - Value messages from root to leaves

Pseudotrees: basic concepts

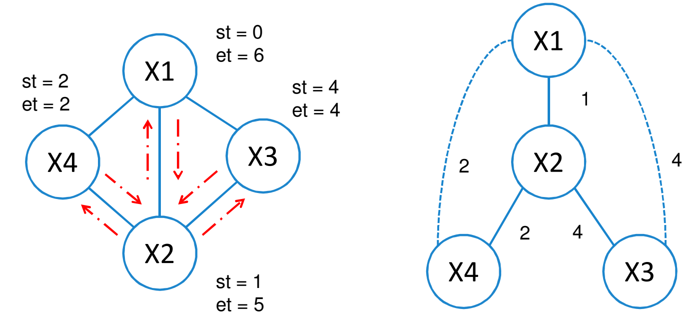
- Pseudotree arrangement of a graph G
  - A rooted tree with the same node as G
  - Adjacent nodes in G falls in the same branch of the tree
    - Nodes in different branches do not share direct coinstraints
  - A DFS visit of a graph induces a Pseudotree
    - Not every pseudotree can be obtained with a DFS visit



Building a DFS tree

- Traverse the graph using a recursive procedure
- Each time we reach Xi from Xj we mark Xi as visited and state that Xj is the father of Xi (and Xi is a children of Xj)
- When a node Xi has a visited neighbour that is not its parent we state that Xj is a pseudo-parent of Xi (and Xi is a pseudochildren of Xj)
- Can be done with a distributed procedure:
  - Each node need only to communicate with neighbours
  - Token passing to propagate information (e.g., visited nodes)

Building a DFS-tree: example



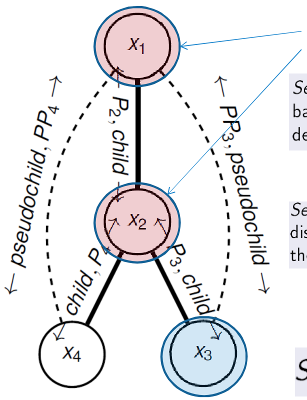
token movement

st: first time node received the token

et: last time node sent the token

time++ = each time token moves

Pseudotrees and Separator



Separator

Definition

Sep_i separator of node X_i: all ancestors (though tree and back edges) which are connected with X_i and with any descendant of X_i

Basic Property

Sep_i minimal set of ancestors that, if removed, completely disconnects the subtree rooted at node X_i from the rest of the problem

Operative Definition

$$Sep_i = \cup_{X_j \in C_i} Sep_j \cup P_i \cup PP_i \setminus X_i$$

Ci: children of node i

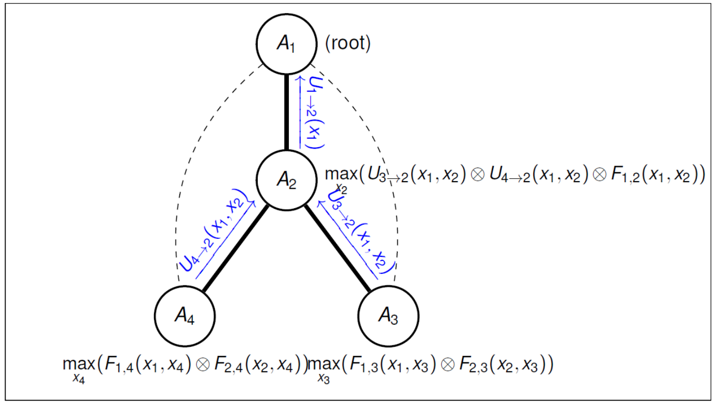
Util Propagation

Aim: build a value function so that root agent can make optimal decision.
Dynamic programming: provide only key information

Each agent computes messages for its parent based on messages received from children and relevant constraints.

Each message projects out X_i (by maximisation) and aggregates (by summation) functions received from children and all constraints with ancestors (parents and pseudoparents)

Util Propagation: messages



Message Computation

Functions → tables (variable are all discrete)
Aggregation → join operator (relational algebra)
Maximization → projection (keeping most valuable tuples)

The *Util* message $U_{i\to j}$ that agent A_i sends to its parent A_j can be computed as:

$$U_{i\to j}(Sep_i) = \max_{x_i}\left(\bigotimes_{A_k \in C_i} U_{k\to i} \otimes \bigotimes_{A_p \in P_i \cup PP_i} F_{i,p}\right)$$

The $\otimes$ operator is a join operator that sums up functions with different but overlapping scores consistently.

Join-sum operator

$F_{2,4}$	x_2	x_4
2	0	0
0	0	1
0	1	0
1	1	1

$F_{1,4}$	x_1	x_4
2	0	0
0	0	1
0	1	0
1	1	1

Add →

$F_{1,4} \otimes F_{2,4}$

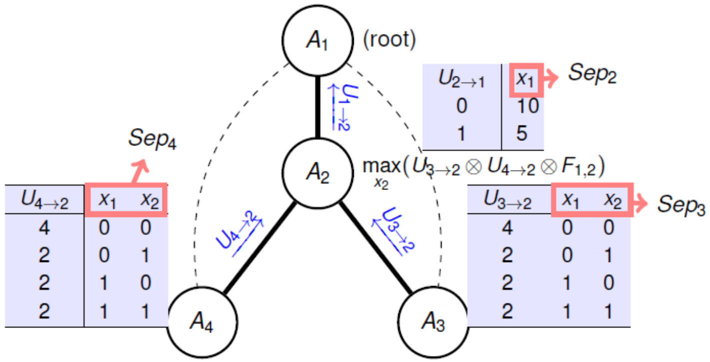
	x_1	x_2	x_4
4	0	0	0
0	0	0	1
2	0	1	0
1	0	1	1
2	1	0	0
1	1	0	1
0	1	1	0
2	1	1	1

Project out x_4 →

$\max_{\{x_4\}}(F_{1,4} \otimes F_{2,4})$

	x_1	x_2	x_4
max(4,0)	0	0	0
	0	0	1
max(2,1)	0	1	0
	0	1	1
max(2,1)	1	0	0
	1	0	1
max(0,2)	1	1	0
	1	1	1

Util Message propagation



Value Propagation

Aim: inform all agents about decision from above so that they can choose best values for their variables

Root agent A_r computes x_r^* which is the argument that maximises the sum of messages received by all children

It sends a message $V_{r \to c} = \{X_r = x_r^*\}$ containing this value to all children C_r

The generic agent A_i sends a message to each child A_j $V_{i \to j} = \{X_s = x_s^*\} \cup X_i = x_i^*$, where $X_s \in Sep_i \cap Sep_j$

Value Computation

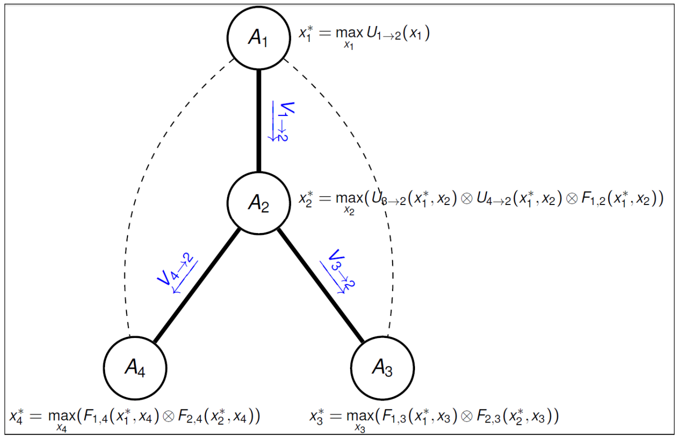
Keeping fixed the value of parent/pseudoparents, finds the value that maximizes the computed cost function in the util phase:

$$x_i^* = \arg\max_{x_i} \left(\bigotimes_{A_j \in C_i} U_{j \to i}(x_i, \mathbf{x}_s^*) \otimes \bigotimes_{A_p \in P_i \cup PP_i} F_{i,p}(x_i, x_p^*) \right)$$

where $\mathbf{x}_s^* = \cup_{A_j \in P_i \cup PP_i} \{x_j^*\}$ is the set of optimal values for A_i's parent and pseudoparents received from A_i's parent.

Can reuse stored tables for computing util messages

Value Propagation: messages



DPOP analysis

- Synchronous algorithm
- Linear number of messages but exponentially large
- Messages (and computation) is exponential in separator size
- Separator size → graph induced width with DFS ordering

Induced graph and Induced width

Given graph G = <V,E>

Width of v = number of v's ancestors

Width of a graph = maximal width of nodes

Given order o over vertices of a graph G:

G* induced graph of G given o

- Process variables from last to first
- When processing v connect all neighbours that precede v (ancestors)

Induced width of G (given o) = width of induced graph

Induced width of G = min induced width over orderings

Finding this is NP-hard

Separator size and induced width

Given DFS order o of a graph G:

Induced width of G over o equals the size of largest separator given by o

Intuition:

- Width of a node = number of induced ancestors
- connecting ancestors = propagating the children's separator in the separator computation

DFS tree and efficiency

- Depth first order is crucial for DPOP efficiency
- Finding optimal order is NP-hard
  - Optimal → minimize separator size
- Good heuristics:
  - Maximum Connected Node (MCN)
  - Maximum Cardinality Set (MCS) for DFS

DFS tree Heuristics

Maximum Connected node

- Choose node with maximum number of neighbours as root
- Select the neighbour with the highest number of neighbours
- Brake ties arbitrarily (e.g. lower/higher Id)

Maximum Cardinality Set for DFS

- Maximum cardinality does not produce a DFS in general, must be adapted to DFS
  - Choose a random node as root
  - Select the neighbour with the highest number of visited neighbours

DFS tree Pseudotrees

- DPOP would work on any pseudotree arrangement of primal graph
- But DFS induces only a specific set of orderings:
  - Not all pseudotres are DFS trees
- We might loose good orderings to keep computation local
  - Trade-off depends on applications

Summary

- DCOPs, general framework to address Multi-Agent coordination
  - Many solution techniques for (relatively) large scale systems
- Complete approaches
  - Suffers from exponential element (DCOPs are hard problems)
  - ADOPT:
    - search based, asynchronous
    - Small messages but exponentially many
  - DPOP:
    - Dynamic programming based, synchronous
    - Few message but exponentially large
    - Typically much more efficient than ADOPT

References

Constraint Network

- Constraint Processing, *R. Dechter, Morgan Kaufmann*

ADOPT

- [Modi et al., 2005] P. J. Modi, W. Shen, M. Tambe, and M.Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence Journal, (161):149-180, 2005.

DPOP

- [Petcu 2007] A. Petcu. A Class of Algorithms for Distributed Constraint Optimization. PhD. Thesis No. 3942, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), 2007. (Chapters 2, 3 and 4)