Chapter 12: Distributed Constraint Handling and Optimization

A. Farinelli[1] M. Vinyals[1] A. Rogers[2] N.R. Jennings[2]

[1]Computer Science Department
University of Verona, Italy

[2]Agents, Interaction and Complexity Group
School of Electronics and Computer Science
University of Southampton,UK

Multi-Agent Systems. MIT Press, 2012 (2nd edition), edited by Gerhard Weiss. http://www.the-mas-book.info

Outline

Outline

Constraints

- Pervade our everyday lives
- Are usually perceived as elements that limit solutions to the problems we face

Constraints

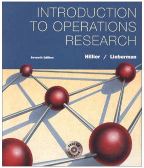
From a computational point of view, they:

- Reduce the space of possible solutions
- Encode knowledge about the problem at hand
- Are key components for efficiently solving hard problems

Constraint Processing

Many different disciplines deal with hard computational problems that can be made tractable by carefully considering the constraints that define the structure of the problem.

Planning
Scheduling



Operational
Research

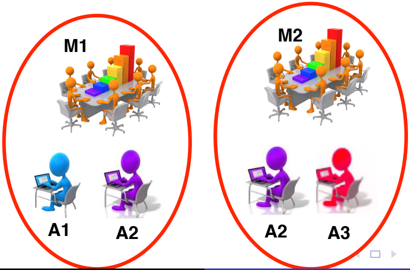
Automated Reasoning
Decision Theory

Computer
Vision

Constraint Processing in Multi-Agent Systems

Focus on how constraint processing can be used to address optimization problems in Multi-Agent Systems (MAS) where:

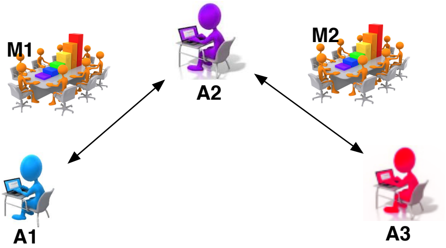
A set of agents must come to some agreement, typically via some form of negotiation, about which action each agent should take in order to jointly obtain the best solution for the whole system.



Distributed Constraint Optimization Problems (DCOPs)

We will consider Distributed Constraint Optimization Problems (DCOP) where:

Each agent negotiates locally with just a subset of other agents (usually called neighbors) that are those that can directly influence his/her behavior.



Distributed Constraint Optimization Problems (DCOPs)

After reading this chapter, you will understand:

- The mathematical formulation of a DCOP
- The main exact solution techniques for DCOPs
  - Key differences, benefits and limitations
- The main approximate solution techniques for DCOPs
  - Key differences, benefits and limitations
- The quality guarantees these approach provide:
  - Types of quality guarantees
  - Frameworks and techniques

Outline

Constraint Networks

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

- $X = \{x_1, \ldots, x_n\}$ is a set of discrete variables;
- $D = \{D_1, \ldots, D_n\}$ is a set of variable domains, which enumerate all possible values of the corresponding variables; and
- $C = \{C_1, \ldots, C_m\}$ 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
  - $r = |S_i|$ is the arity of the constraint
  - Two types: hard or soft

Hard constraints

- A hard constraint C_i^h is a relation R_i that enumerates all the valid joint assignments of all variables in the scope of the constraint.

$$R_i \subseteq D_{i_1} \times \ldots \times D_{i_r}$$

R_i	x_j	x_k
	0	1
	1	0

Soft constraints

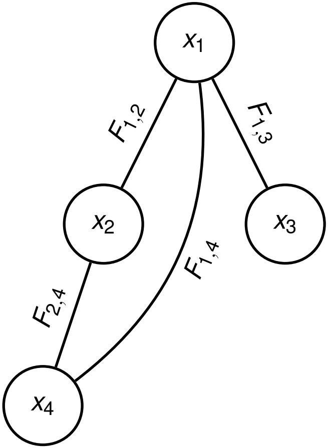
- A soft constraint C_i^s is a function F_i that maps every possible joint assignment of all variables in the scope to a real value.

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

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

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
- They can be represented by a constraint graph



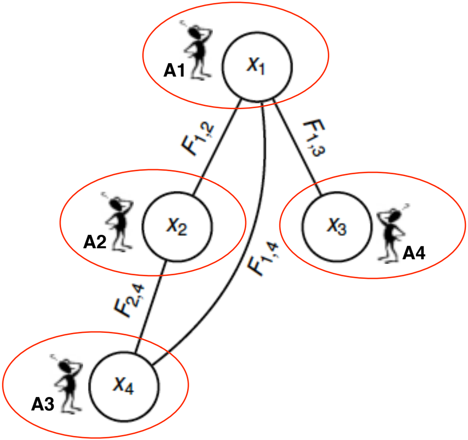
Different objectives, different problems

- **Constraint Satisfaction Problem (CSP)**
  - Objective: find an assignment for all the variables in the network that satisfies all constraints.
- **Constraint Optimization Problem (COP)**
  - Objective: find an assignment for all the variables in the network that satisfies all constraints and optimizes a global function.
  - Global function = aggregation (typically sum) of local functions. $F(x) = \sum_i F_i(x_i)$

Distributed Constraint Reasoning

When operating in a decentralized context:

- a set of agents control variables
- agents interact to find a solution to the constraint network



Distributed Constraint Reasoning

Two types of decentralized problems:

- **distributed CSP (DCSP)**
- **distributed COP (DCOP)**

Here, we focus on DCOPs.

Distributed Constraint Optimization Problem (DCOP)

A DCOP consists of a constraint network $\mathcal{N} = \langle X, D, C \rangle$ 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

Distributed Constraint Optimization Problem (DCOP)

- Agents are assumed to be fully cooperative
  - Goal: find the assignment that optimizes the global function, not their local local utilities.
- Solving a COP is NP-Hard and DCOP is as hard as COP.

Motivation

Why distribute?

- Privacy
- Robustness
- Scalability

Outline

Real World Applications

Many standard benchmark problems in computer science can be modeled using the DCOP framework:

- graph coloring

As can many real world applications:

- human-agent organizations (e.g. meeting scheduling)
- sensor networks and robotics (e.g. target tracking)

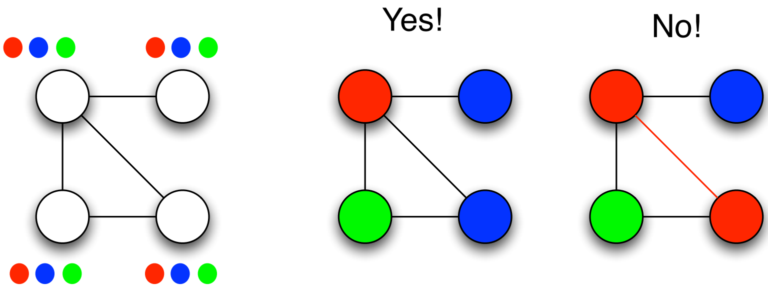
Outline

Graph coloring

- Popular benchmark
- Simple formulation
- Complexity controlled with few parameters:
  - Number of available colors
  - Number of nodes
  - Density ($\#nodes/\#constraints$)
- Many versions of the problem:
  - CSP, MaxCSP, COP

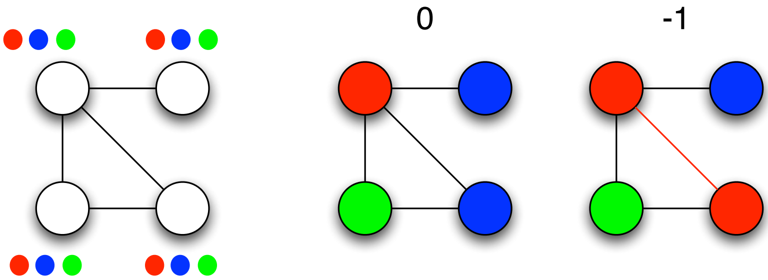
Graph coloring - CSP

- Nodes can take k colors
- Any two adjacent nodes should have different colors
    - If it happens this is a conflict



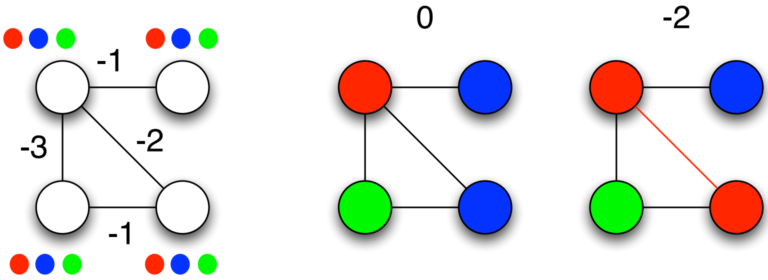
Graph coloring - Max-CSP

- Minimize the number of conflicts



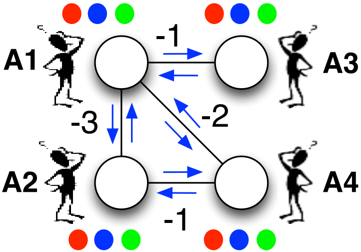
Graph coloring - COP

- Different weights to violated constraints
- Preferences for different colors



Graph coloring - DCOP

- Each node:
  - controlled by one agent
- Each agent:
  - Preferences for different colors
  - Communicates with its direct neighbours in the graph



- A1 and A2 exchange preferences and conflicts
- A3 and A4 do not communicate

Outline

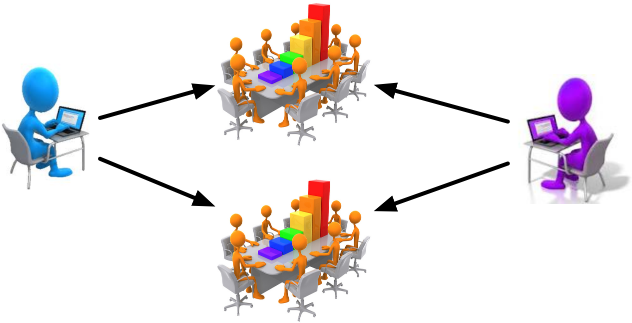
Meeting Scheduling

Motivation:

- Privacy
- Robustness
- Scalability

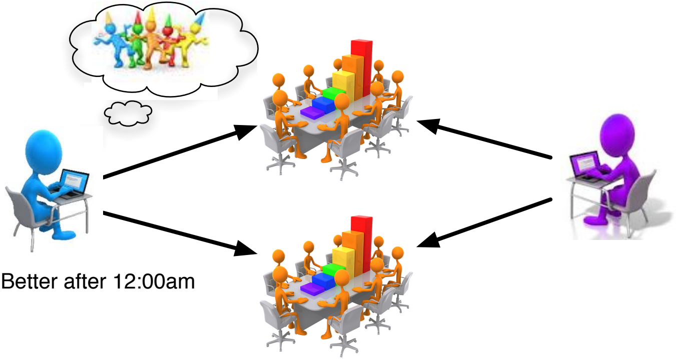
Meeting Scheduling

In large organizations many people, possibly working in different departments, are involved in a number of work meetings.



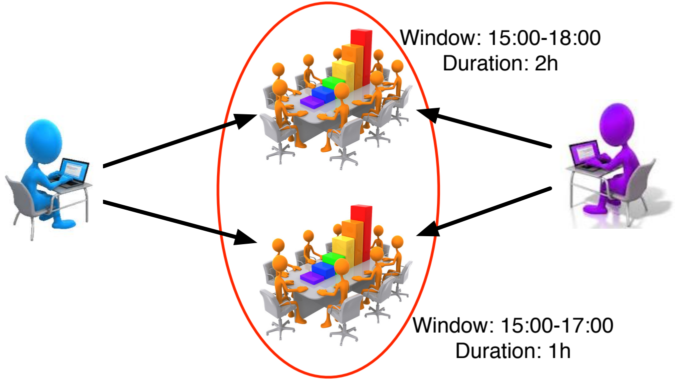
Meeting Scheduling

People might have various private preferences on meeting start times



Meeting Scheduling

Two meetings that share a participant cannot overlap



DCOP formalization for the meeting scheduling problem

- A set of agents representing participants
- A set of variables representing meeting starting times according to a participant.
- Hard Constraints:
  - Starting meeting times across different agents are equal
  - Meetings for the same agent are non-overlapping.
- Soft Constraints:
  - Represent agent preferences on meeting starting times.

Objective: find a valid schedule for the meeting while maximizing the sum of individuals' preferences.

Outline

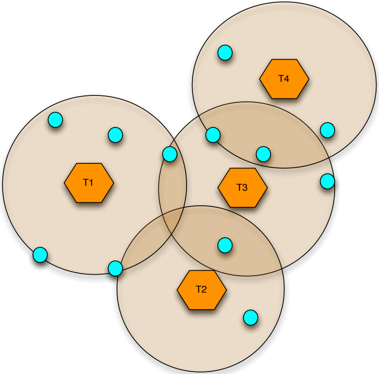
Target Tracking

Motivation:

- Privacy
- Robustness
- Scalability

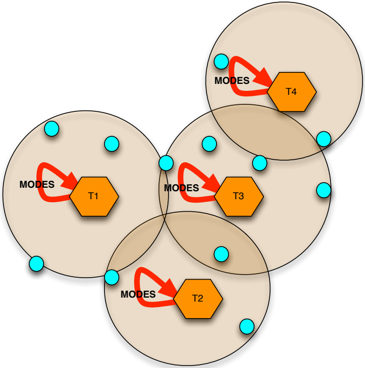
Target Tracking

A set of sensors tracking a set of targets in order to provide an accurate estimate of their positions.



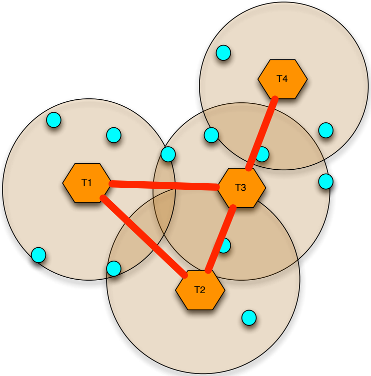
Target Tracking

Sensors can have different sensing modalities that impact on the accuracy of the estimation of the targets' positions.



Target Tracking

Collaboration among sensors is crucial to improve system performance



DCOP formalization for the target tracking problem

- Agents represent sensors
- Variables encode the different sensing modalities of each sensor
- Constraints
  - relate to a specific target
  - represent how sensor modalities impacts on the tracking performance
- Objective:
  - Maximize coverage of the environment
  - Provide accurate estimations of potentially dangerous targets

Outline

Complete Algorithms

- 👍 Always find an optimal solution
- 👎 Exhibit an exponentially increasing coordination overhead
- 👎 Very limited scalability on general problems.

Complete Algorithms

- **Completely decentralised**
  - Search-based.
    - Synchronous: SyncBB, AND/OR search
    - Asynchronous: ADOPT, NCBB and AFB.
  - Dynamic programming.
- **Partially decentralised**
  - OptAPO

Next, we focus on completely decentralised algorithms

Decentralised Complete Algorithms

Search-based

- Uses distributed search
- Exchange individual values
- Small messages but
  ... exponentially many

Representative: ADOPT [Modi et al., 2005]

Dynamic programming

- Uses distributed inference
- Exchange constraints
- Few messages but
  ... exponentially large

Representative: DPOP [Petcu and Faltings, 2005]

Outline

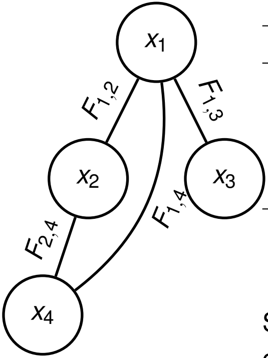
ADOPT

ADOPT (Asynchronous Distributed OPTimization) [Modi et al., 2005]:

- Distributed backtrack search using a best-first strategy
- Best value based on local information:
  - Lower/upper bound estimates of each possible value of its variable
  - 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)

ADOPT by example

4 variables (4 agents): x_1, x_2, x_3, x_4 with $D = \{0, 1\}$



4 identical cost functions

$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

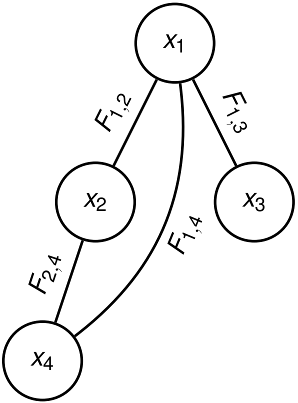
Goal: find a variable assignment with *minimal* cost

Solution: $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ and $x_4 = 1$ giving total cost 1.

DFS arrangement

- Before executing ADOPT, agents must be arranged in a depth first search (DFS) tree.
- DFS trees have been frequently used in optimization because they have two interesting properties:
  - Agents in different branches of the tree do not share any constraints;
  - Every constraint network admits a DFS tree.

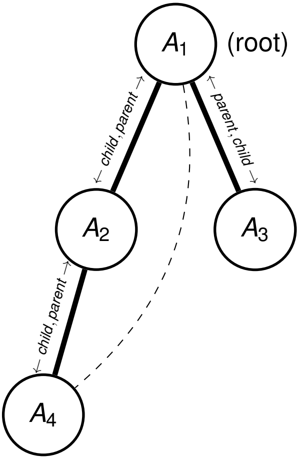
ADOPT by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

$\rightarrow$

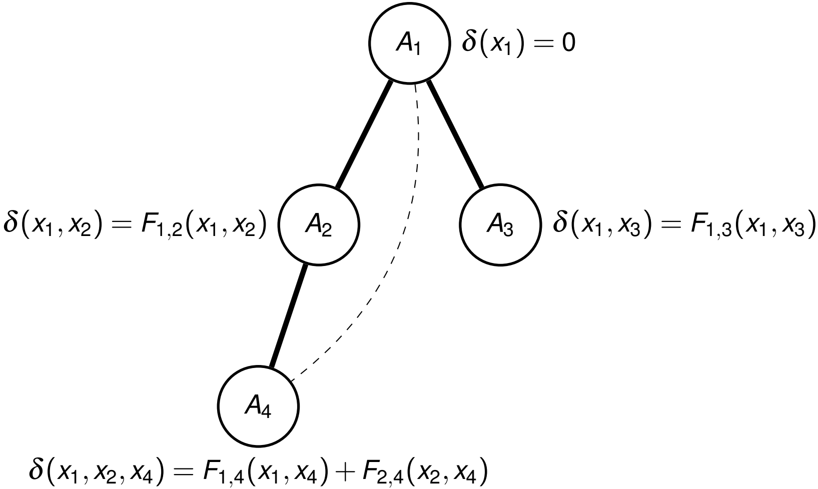
DFS arrangement



Cost functions

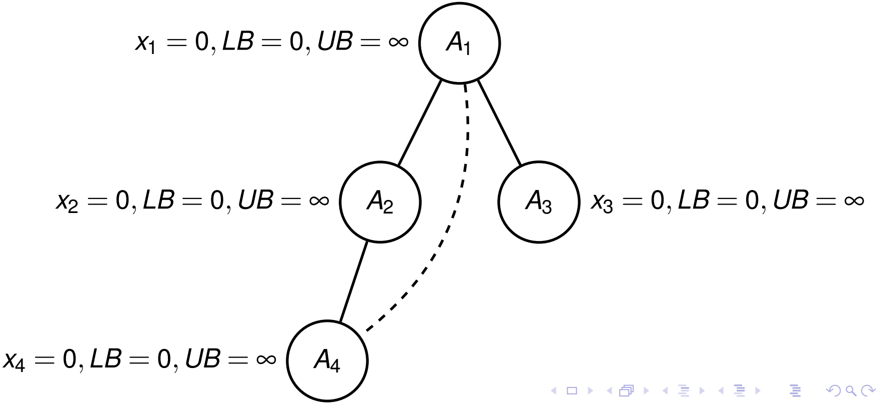
The local cost function for an agent A_i ($\delta(x_i)$) is the sum of the values of constraints involving only higher neighbors in the DFS.

ADOPT by example



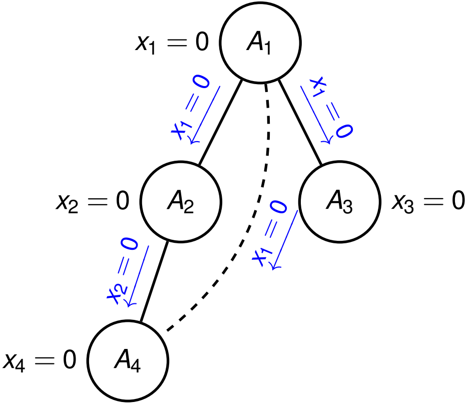
Initialization

Each agent initially chooses a random value for their variables and initialize the lower and upper bounds to zero and infinity respectively.



ADOPT by example

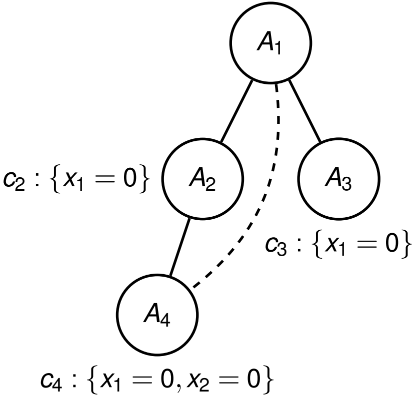
Value messages are sent by an agent to all its neighbors that are lower in the DFS tree



A_1 sends three value message to A_2, A_3 and A_4 informing them that its current value is 0.

ADOPT by example

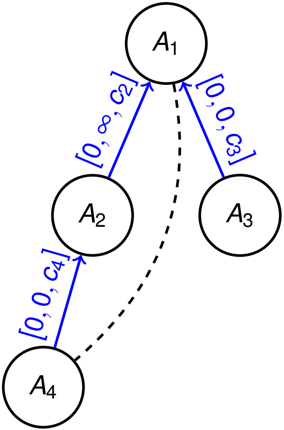
Current Context: a partial variable assignment maintained by each agent that records the assignment of all higher neighbours in the DFS.



- Updated by each VALUE message
- If current context is not compatible with some child context, the latter is re-initialized (also the child bound)

ADOPT by example

Each agent A_i sends a *cost message* to its parent A_p



Each cost message reports:

- The minimum lower bound (LB)
- The maximum upper bound (UB)
- The context (c_i)

$$[LB, UP, c_i]$$

Lower bound computation

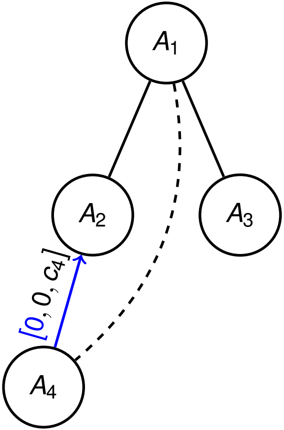
Each agent evaluates for each possible value of its variable:

- its local cost function with respect to the current context
- adding all the compatible lower bound messages received from children.

Analogous computation for upper bounds

ADOPT by example

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

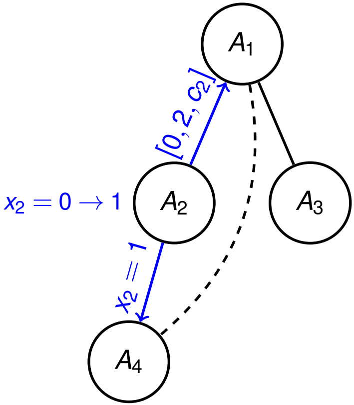


- Recall that A_4 local cost function is: $\delta(x_1, x_2, x_4) = F_{1,4}(x_1, x_4) + F_{2,4}(x_2, x_4)$
- Restricted to the current context $c_4 = \{(x_1 = 0, x_2 = 0)\}$: $\lambda(0, 0, x_4) = F_{1,4}(0, x_4) + F_{2,4}(0, x_4)$.
- For $x_4 = 0$: $\lambda(0, 0, 0) = F_{1,4}(0, 0) + F_{2,4}(0, 0) =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

ADOPT by example

Each agent asynchronously chooses the value of its variable that minimizes its lower bound.



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

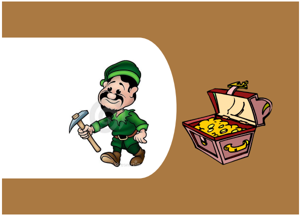
A_2 changes its value to $x_2 = 1$ with $LB = 0$.

Backtrack thresholds

The search strategy is based on lower bounds

Problem

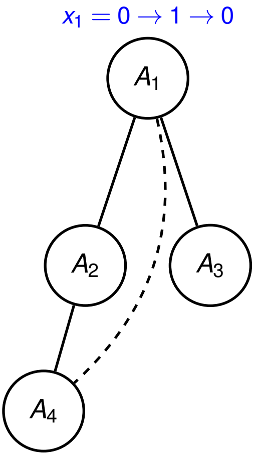
- Values abandoned before proven to be suboptimal
- Lower/upper bounds only stored for the current context



Solution

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

ADOPT by example



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

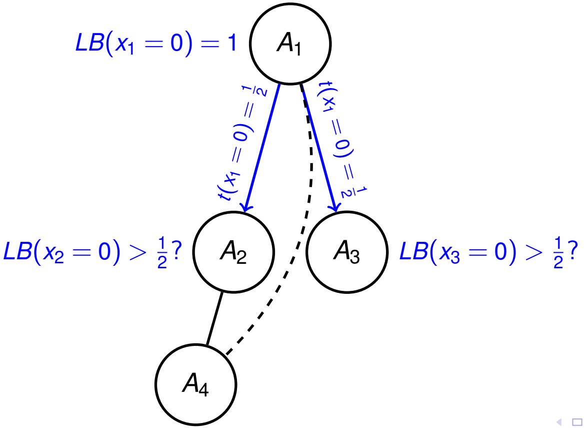
- Reconstructing from scratch is inefficient
- Remembering solutions is expensive

Backtrack thresholds

Solution: Backtrack thresholds

- Lower bound previously determined by children
- Polynomial space
- Control backtracking to efficiently search
- Key point: do not change value until LB(currentvalue)> threshold

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.

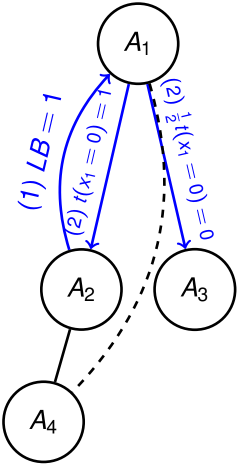


Rebalance incorrect threshold

How to correctly subdivide threshold among children?

- Parent distributes the accumulated bound among children
  - Arbitrarily/Using some heuristics
- Correct subdivision as feedback is received from children
  - $LB < t(CONTEXT)$
  - $t(CONTEXT) = \sum_{C_i} t(CONTEXT) + \delta$

Backtrack Threshold Computation



- When A_1 receives a new lower bound from A_2 rebalances thresholds
- A_1 resends threshold messages to A_2 and A_3

ADOPT extensions

- BnB-ADOPT [Yeoh et al., 2008] reduces computation time by using depth-first search with branch and bound strategy
- [Ali et al., 2005] suggest the use of preprocessing techniques for guiding ADOPT search and show that this can result in a consistent increase in performance.

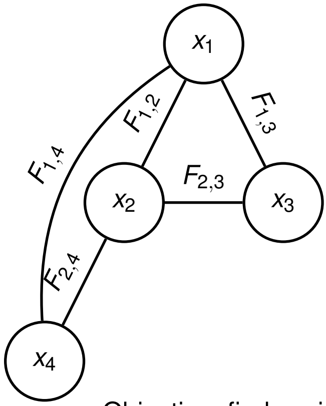
Outline

DPOP

DPOP (Dynamic Programming Optimization Protocol) [Petcu and Faltings, 2005]:

- Based on the dynamic programming paradigm.
- Special case of Bucket Tree Elimination Algorithm (BTE) [Dechter, 2003].

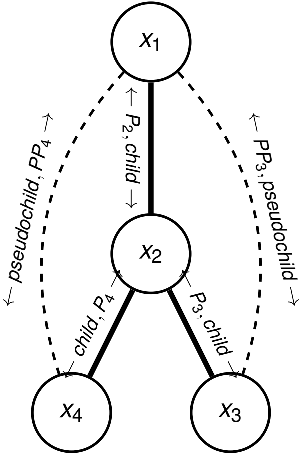
DPOP by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

=>

DFS arrangement



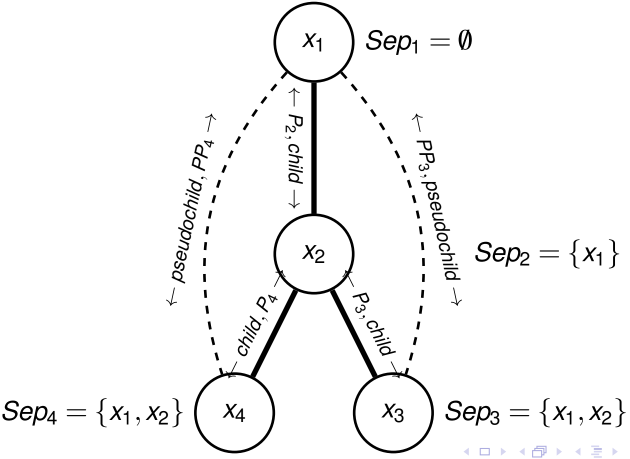
Objective: find assignment with maximal value

DPOP phases

Given a DFS tree structure, DPOP runs in two phases:

- *Util* propagation: agents exchange *util messages* up the tree.
  - Aim: aggregate all info so that root agent can choose optimal value
- *Value* propagation: agents exchange value messages down the tree.
  - Aim: propagate info so that all agents can make their choice given choices of ancestors

Sep_i: set of agents preceding A_i in the pseudo-tree order that are connected with A_i or with a descendant of A_i.



Util message

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

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

Size exponential in Sep_i

All incoming messages from children

Shared constraints with parents/pseudoparents

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

Join 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

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

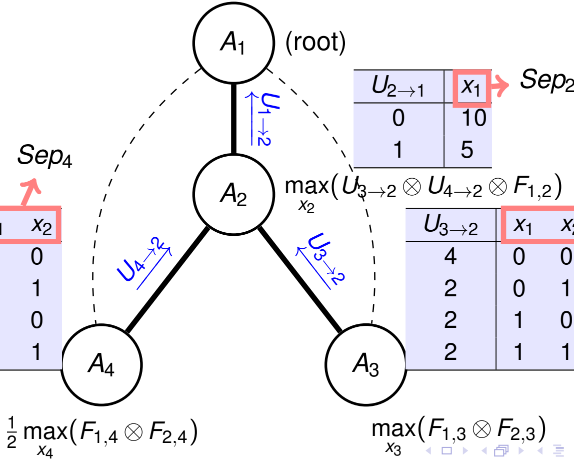
	x_1	x_2	x_4
4	0	0	0
0	0	0	1
2	0	1	0
1	0	1	1
2	1	0	0
2	1	0	1
0	1	1	0
2	1	1	1

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(2,2)	1	0	0
	1	0	1
max(0,2)	1	1	0
	1	1	1

Complexity exponential to the largest Sep_i.
Largest Sep_i = induced width of the DFS tree ordering used.



$U_{2\to 1}$	x_1
0	10
1	5

$U_{4\to 2}$	x_1	x_2
4	0	0
2	0	1
2	1	0
2	1	1

$U_{3\to 2}$	x_1	x_2
4	0	0
2	0	1
2	1	0
2	1	1

Value message

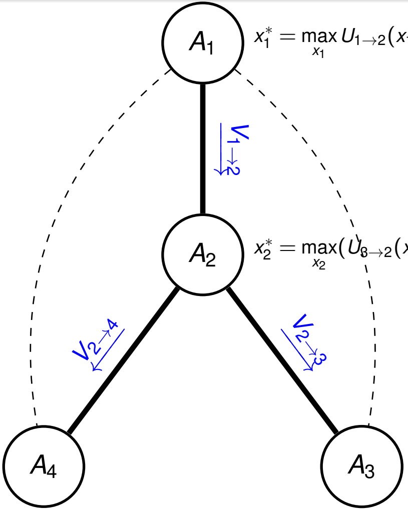
Keeping fixed the value of parent/pseudoparents, finds the value that maximizes the computed cost function in the util phase:

$$x_i^* = arg\max_{x_i} \left(\sum_{A_j \in C_i} U_{j \to i}(x_i, x_p^*) + \sum_{A_j \in P_i \cup PP_i} F_{i,j}(x_i, x_j^*) \right)$$

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

Propagates this value through children down the tree:

$$V_{i \to j} = \{x_i = x_i^*\} \cup \bigcup_{x_s \in Sep_i \cap Sep_j} \{x_s = x_s^*\}$$



$$x_1^* = \max_{x_1} U_{1\to 2}(x_1)$$

$$x_2^* = \max_{x_2}(U_{3\to 2}(x_1^*, x_2) \otimes U_{4\to 2}(x_1^*, x_2) \otimes F_{1,2}(x_1^*, x_2))$$

$$x_4^* = \max_{x_4}(F_{1,4}(x_1^*, x_4) \otimes F_{2,4}(x_2^*, x_4))$$

$$x_3^* = \max_{x_3}(F_{1,3}(x_1^*, x_3) \otimes F_{2,3}(x_2^*, x_3))$$

DPOP extensions

- MB-DPOP [Petcu and Faltings, 2007] trades-off message size against the number of messages.
- A-DPOP trades-off message size against solution quality [Petcu and Faltings, 2005(2)].

Outline

Why Approximate Algorithms

"Very often optimality in practical applications is not achievable"

Approximate algorithms

- Sacrify optimality in favor of computational and communication efficiency
- Well-suited for large scale distributed applications:
  - sensor networks
  - mobile robots

Outline

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.

Distributed Local Greedy approaches

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)

Activation probability *to mitigate issues with parallel executions*

[S. Fitzpatrick and L. Meetrens, 2003]

- Initialize agents with a random assignment and communicate values to neighbors
- Each agent:
  - Generates a random number and executes only if it is less than the 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 neighbors

DSA-1: discussion

- Extremely low computation/communication
- Shows an anytime property (not guaranteed)
- Activation probability:
  - Must be tuned
  - Domain dependent (no general rule)

Maximum Gain Message (MGM-1)

***Coordination** among neighbours to decide which single agent can perform the move.*

[R. T. Maheswaran et al., 2004]

- Initialize agents with a random assignment and communicate values to neighbors
- Each agent:
  - Compute and exchange possible gains
  - Agent with maximum (positive) gain executes
  - Communicate and receive possible variables changes to/from neighbors

MGM-1: discussion

- = More communication than DSA but still linear
- = Empirically similar to DSA
- 👍 Guaranteed to be anytime
- 👍 Does not require any parameter tuning.

Decentralised greedy approaches

- 👍 Very little memory and computation
- 👍 Anytime behaviours
- 👎 Could result in very bad solutions
  - local maxima arbitrarily far from optimal

Outline

1. Introduction
2. Distributed Constraint Reasoning
3. Applications and Exemplar Problems
4. Complete algorithms for DCOPs
5. Approximated Algorithms for DCOPs
   - Local greedy methods: DSA-1, MGM-1 (Heuristic)
   - GDL-based approaches: Max-Sum (Heuristic)
   - Quality guarantees: k-optimality, region optimality, bounded Max-Sum

GDL Based Approximate Algorithms (GDL)

Generalized Distributive Law (GDL)

- Unifying framework for inference in Graphical Models
- Builds on basic mathematical properties of semi-rings
- Widely used in information theory, statistical physics, graphical models

	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	$(\wedge,1)$	$(\vee,0)$.	

GDL Based Approximate Algorithms (GDL)

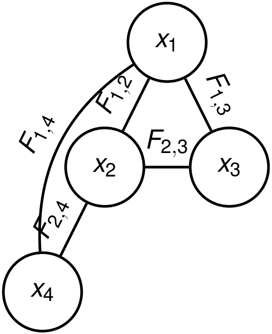
Max-Sum [A. Farinelli et al., 2008]

- DCOP-Settings: maximize the social welfare
- GDL approximate iterative message passing algorithm

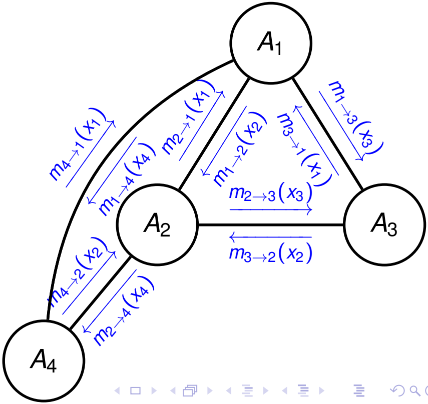
	K	"$(+,0)$"	"$(\cdot,1)$"	short name
1.	A	$(+,0)$	$(\cdot,1)$	
2.	$A[x]$	$(+,0)$	$(\cdot,1)$	
3.	$A[x,y,\ldots]$	$(+,0)$	$(\cdot,1)$	
4.	$[0,\infty)$	$(+,0)$	$(\cdot,1)$	sum-product
5.	$(0,\infty]$	$(\min,\infty)$	$(\cdot,1)$	min-product
6.	$[0,\infty)$	$(\max,0)$	$(\cdot,1)$	max-product
7.	$(-\infty,\infty]$	$(\min,\infty)$	$(+,0)$	min-sum
8.	$[-\infty,\infty)$	$(\max,-\infty)$	$(+,0)$	max-sum
9.	$\{0,1\}$	(OR, 0)	(AND, 1)	Boolean
10.	2^S	$(\cup,\emptyset)$	$(\cap,S)$	
11.	Λ	$(\vee,0)$	$(\wedge,1)$	
12.	Λ	$(\wedge,1)$	$(\vee,0)$.	

The Max-Sum algorithm

Agents iteratively exchange messages to build a local function that depends only on the variables they control



=>



Max-Sum messages

At each execution step, each agent A_i sends to each of its neighbors A_j the message:

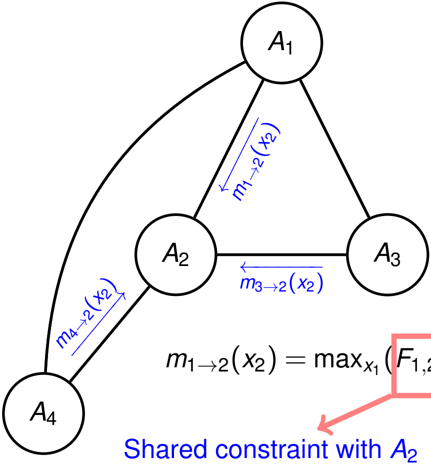
$$m_{i\to j}(x_j) = \alpha_{ij} + \max_{x_i}\left(\boxed{F_{ij}(x_i, x_j)} + \boxed{\sum_{k\in N(i)\setminus j} m_{k\to i}(x_i)}\right)$$

Shared constraint with A_j All incoming messages except from A_j

where:

- α_{ij} is a normalization constant added to all components of the message so that $\sum_{x_j} m_{i\to j}(x_j) = 0$
- $N(i)$ is the set of indices for variables that are connected to x_i

Max-Sum by example



$$m_{1\to 2}(x_2) = \max_{x_1}\left(\boxed{F_{1,2}(x_1, x_2)} + \boxed{m_{3\to 1}(x_1) + m_{4\to 1}(x_1)}\right)$$

Shared constraint with A_2 All incoming messages except from A_2

Max-Sum assignments

At each iteration, each agent A_i:

- computes its local function as:

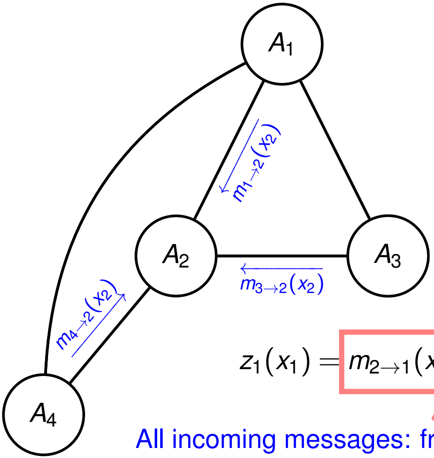
$$z_i(x_i) = \boxed{\sum_{k \in N(i)} m_{k \to i}(x_i)}$$

All incoming messages

- sets its assignment as the value that maximizes its local function:

$$\tilde{x}_i = arg \max_{x_i} z_i(x_i)$$

Max-Sum by example

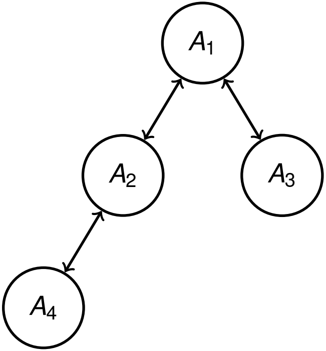


$$z_1(x_1) = \boxed{m_{2\to 1}(x_1) + m_{3\to 1}(x_1) + m_{4\to 1}(x_1)}$$

All incoming messages: from A_4, A_3 and A_1

Max-Sum on acyclic graphs

- Optimal on acyclic graphs
  - Different branches are independent
  - z functions provide correct estimations of agents contributions to the global problem
- Convergence guaranteed in a polynomial number of cycles



Max-Sum on cyclic graphs

On cyclic graphs, limited theoretical results:

- Lack of convergence guarantees
- When converges, it does converge to a neighborhood maximum
- Neighborhood maximum: guaranteed to be greater than all other maxima within a particular region of the search space

Outline

Quality guarantees

So far, algorithms presented (DSA-1, MGM-1, Max-Sum) do not provide any guarantee on the quality of their solutions

- Quality highly dependent on many factors which cannot always be properly assessed before deploying the system.
- Particularly adverse behaviour on specific pathological instances.

Challenge:

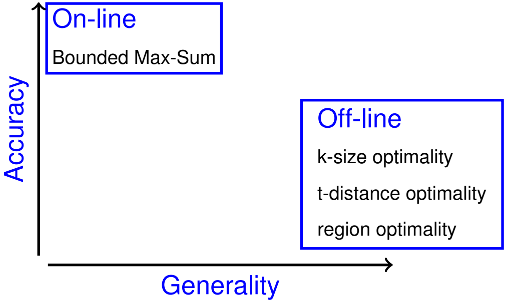
- Quality assessment on approximate algorithms

Quality guarantees for approx. 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)

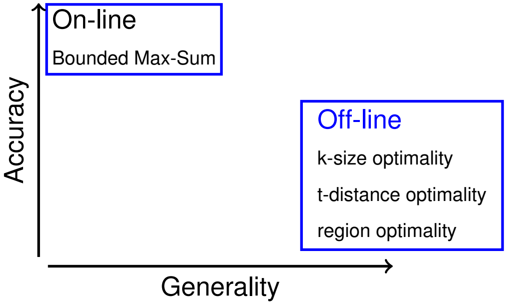
Quality guarantees categories

- Off-line
  - Prior running the algorithm
  - Not tied to specific problem instances
- On-line
  - After running the algorithm
  - On the particular problem instance



Quality guarantees categories

- Off-line
  - Prior running the algorithm
  - Not tied to specific problem instances
- On-line
  - After running the algorithm
  - On the particular problem instance

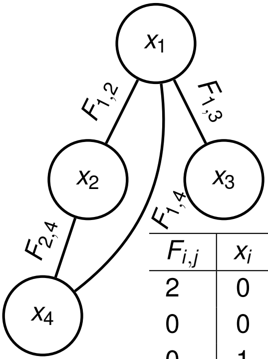


Enable trade-offs at design time

k-size optimality framework

- Gives a bound on the solution quality of any k-optimal solution [J.P.Pearce and M.Tambe, 2007]
- The k-optimal solution is a local maximum in a region characterized by size
- Its value cannot be improved by changing the assignment of k or fewer agents

k-optimality by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

Goal: *maximize*

$$\hat{\mathbf{x}} = \{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1\}$$

with value

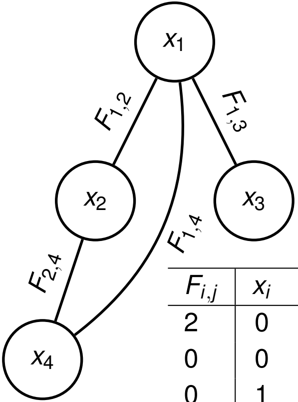
$$\begin{aligned} F(\hat{\mathbf{x}}) &= F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} \\ &= 1 + 1 + 1 + 1 = 4 \end{aligned}$$

Optimal? No

$$\mathbf{x}^* = \{x_1 = x_2 = x_3 = x_4 = 0\}$$

with value $F(\mathbf{x}^*) = 8$.

k-optimality by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

Goal: *maximize*.

$$\hat{\mathbf{x}} = \{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1\}$$

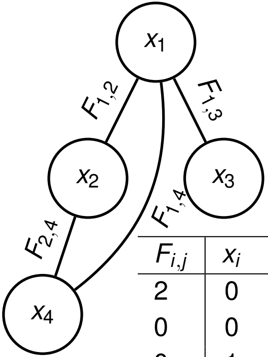
with value

$$\begin{aligned} F(\hat{\mathbf{x}}) &= F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} \\ &= 1 + 1 + 1 + 1 = 4 \end{aligned}$$

2-size-Optimal? Yes

If only two agents can change their variables' values there is no solution that obtains higher value.

k-optimality by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

Goal: *maximize*.

$$\hat{\mathbf{x}} = \{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1\}$$

$$\begin{aligned} F(\hat{\mathbf{x}}) &= F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} \\ &= 1 + 1 + 1 + 1 = 4 \end{aligned}$$

3-size-optimal? No, a better solution if three agents change their values:
$\hat{\mathbf{x}}' = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0\}$

$$\begin{aligned} F(\hat{\mathbf{x}}) &= F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} \\ &= 2 + 0 + 2 + 2 = 6 \geq 4 \end{aligned}$$

K-optimality guarantees

For any DCOP with non-negative values [Pearce and M.Tambe, 2007]

Number of agents

Number of constraints

$$F(\boxed{\hat{\mathbf{x}}}) \geq \underbrace{\boxed{\frac{\binom{\boxed{n}-\boxed{m}}{k-m}}{\binom{n}{k}-\binom{n-m}{k}}}}_{\alpha} F(\mathbf{x}^*)$$

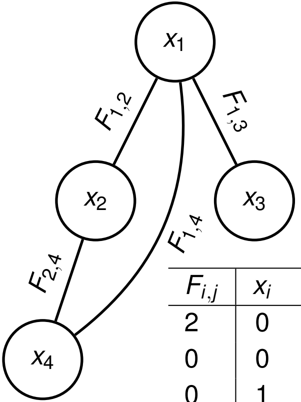
k-optimal solution

For binary constraints ($m=2$):

$$F(\hat{\mathbf{x}}) \geq \frac{k-1}{2\boxed{n}-k-1} F(\mathbf{x}^*)$$

Number of agents

k-optimality by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

2-size optimal solution:

$$\hat{\mathbf{x}} = \{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1\}$$

For $k = 2$, $n = 4$

$$F(\hat{\mathbf{x}}) = 4 \geq \frac{2-1}{2 \cdot 4 - 2 - 1} = \frac{1}{5} F(\mathbf{x}^*)$$

Goal: *maximize*.

k-optimality guarantees

Apply to:

- any constraint graph with *n* agents
- independently of
  - graph structure
  - reward structure

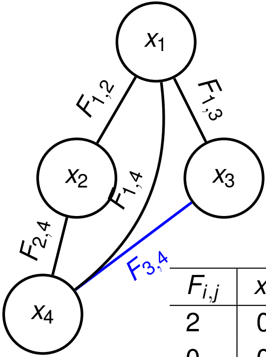
Very strong and general result

Depend on:

- arity of constraints
- value of k
- number of agents

Very low guarantees on large-scale systems

k-optimality by example



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

After adding a constraint between x_3 and x_4

The value of any 2-size optimal is still guaranteed to be greater than $\frac{1}{5}$ of the value of the optimal.

Goal: *maximize*.

k-optimality guarantees

Apply to:

- any constraint graph with *n* agents
- independently of
  - graph structure
  - reward structure

Very strong and general result

Depend on:

- arity of constraints
- value of k
- number of agents

Very low guarantees on large-scale systems

k-optimality algorithms

k-optimality guarantees are independent of the algorithm employed to find k-optimal solutions

How do agents search for a k-size optimal solution?

- A group of k agents coordinate their choice to find a solution optimal for the group.
- Hill climbing algorithms (e.g. DSA-1, MGM-1) are able to find a 1-size optimal solution but no guarantee for $k \leq 1$.

k-optimality algorithms

Need algorithms for computing k-optimal solutions:

- $k = 2$ variants of MGM and DSA [R. T. Maheswaran et al., 2004]
- DALO finds k-size optimal solutions for arbitrary k [C. Kiekintveld et al., 2010]

The higher k the more complex the computation (exponential)

Region optimality: Arbitrary region criteria

- Size is only one possible criteria to define optimality of a solution.
- Other work explored other criteria:
  - t-distance: based on the distance between nodes in the graph [C. Kiekintveld et al., 2010].
  - size-bounded-distance: based on the distance between nodes in the graph but bounded on their size [M. Vinyals et al., 2011].

The region optimality framework allows guarantees for region optimal defined with any criteria [M. Vinyals et al., 2011].

Max-Sum and region optimality

- Upon convergence Max-Sum is optimal on SLT regions [Y. Weiss and W. T. Freeman, 2001]
- Single Loops and Trees (SLT): all groups of agents whose vertex induced subgraph contains at most one cycle.
- Region optimality defines bounds for Max-Sum assignments [M. Vinyals et al., 2010].

Any Max-Sum solution on convergence is 3-size optimal

k-optimality guarantees

Apply to:

- any constraint graph with *n* agents
- independently of
  - graph structure
  - reward structure

Very strong and general result

Depend on:

- arity of constraints
- value of k
- number of agents

Very low guarantees *on* *large-scale* *systems*

Solution: exploit a priori knowledge of the problem

Exploiting a priori knowledge on graph structure

Exploit a priori knowledge of the graph structure

k-size optimality guarantees:

- valid for any constraint network.
- result of a worst case analysis on a complete graph.

Exploiting a priori knowledge on graph structure

E.g., for a ring topology, where each agent has only two constraints:

$$F(\hat{\mathbf{x}}) \geq \frac{k-1}{k+1} F(\mathbf{x}^*)$$

Apply to:

- any ring topology graph
- independently of
  - ~~graph structure~~
  - reward structure

Less strong and general result

Depend on:

- arity of constraints
- value of k
- ~~number of agents~~

High guarantees on large-scale systems

Exploiting a priori knowledge on reward structure

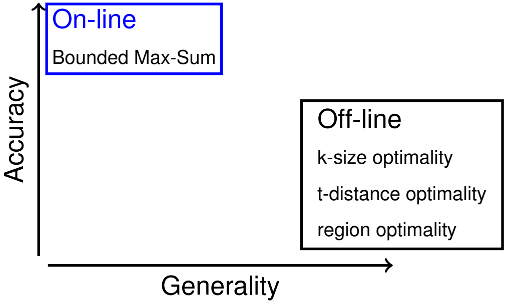
Exploit a priori knowledge on *reward structure*

- Guarantees can be improved by knowing the ratio between the minimum to the maximum reward [E. Bowring et al., 2008].

Quality guarantees categories

"The more the knowledge about a problem, the tighter the 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



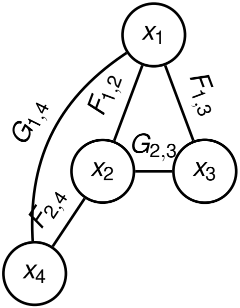
On-line guarantees are usually much tighter than off-line ones

Bounded Max-Sum (BMS)

Bounded Max-Sum (BMS) [A. Rogers et al., 2011]

- remove cycles in the original constraint network by simply ignoring dependencies among agents.

Bounded Max-Sum (BMS)

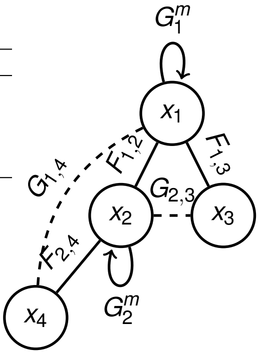


$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

$G_{i,j}$	x_i	x_j
4	0	0
3	0	1
1	1	0
2	1	1

$\Rightarrow$

(1) Maximum Weight Spanning Tree



$\Downarrow$

(2) Run Max-Sum

$\Leftarrow$

$F(\mathbf{x}^*) \leq \rho F(\tilde{\mathbf{x}})$ (3) Compute Bound $\tilde{\mathbf{x}} = (x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0)$

Computing edge weights

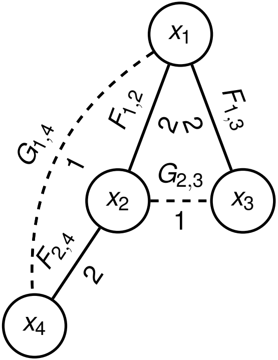
Edge weight: maximum possible impact of removing a constraint:

$$w_{ij} = min\{w'_{ij}, w''_{ij}\}$$

$$w'_{14} = \max_{x_4}[\max_{x_1} G_{14} - \min_{x_1} G_{14}] = 3$$

$$w''_{14} = \max_{x_1}[\max_{x_4} G_{14} - \min_{x_4} G_{14}] = 1$$

$$w_{14} = min(3, 1) = 1$$



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

$G_{i,j}$	x_i	x_j
4	0	0
3	0	1
1	1	0
2	1	1

Computing the bound

After running max-sum, the bound is computed as:

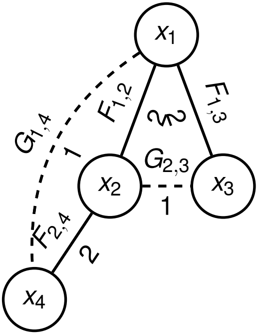
$$\rho = \frac{\boxed{F^m(\tilde{\mathbf{x}})} + W}{\boxed{F(\tilde{\mathbf{x}})}}$$

tree-structured constraint network

original constraint network

where:

- W is the sum of the weights of the removed constraints.
- $\tilde{\mathbf{x}}$ is the BMS assignment over the tree-structured constraint network



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

$G_{i,j}$	x_i	x_j
4	0	0
3	0	1
1	1	0
2	1	1

$$W = w_{14} + w_{23} = 2$$

Outline

- Constraint processing
  - exploit problem structure to solve hard problems efficiently
- DCOP framework
  - applies constraint processing to solve decision making problems in Multi-Agent Systems
  - increasingly being applied within real world problems.

References I

- [Modi et al., 2005] P. J. Modi, W. Shen, M. Tambe, and M.Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence Jour- nal, (161):149-180, 2005.
- [Yeoh et al., 2008] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. In Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems, pages 591Ð598, 2008.
- [Ali et al., 2005] S. M. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pages 1041Ð1048, 2005.
- [Petcu and Faltings, 2005] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint opti- mization. In Proceedings of the Nineteenth International Joint Conference on Arti- ficial Intelligence, pages 266-271, 2005.
- [Dechter, 2003] R. Dechter. Constraint Processing. Morgan Kaufmann, 2003.

References II

- [Petcu and Faltings, 2005(2)] A. Petcu and B. Faltings. A-DPOP: Approximations in distributed optimization. In Principles and Practice of Constraint Programming, pages 802-806, 2005.
- [Petcu and Faltings, 2007] A. Petcu and B. Faltings. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In Proceedings of the Twentieth International Joint Confer- ence on Artificial Intelligence, pages 1452-1457, 2007.
- [S. Fitzpatrick and L. Meetrens, 2003] S. Fitzpatrick and L. Meetrens. Distributed Sensor Networks: A multiagent perspective, chapter Distributed coordination through anarchic optimization, pages 257- 293. Kluwer Academic, 2003.
- [R. T. Maheswaran et al., 2004] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical game-based approach. In Proceedings of the Seventeenth International Conference on Parallel and Distributed Computing Systems, pages 432-439, 2004.

References III

- [A. Farinelli et al., 2008] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems, pages 639-646, 2008.
- [J.P.Pearce and M.Tambe, 2007] J.P.Pearce and M.Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pages 1446-1451, 2007.
- [C. Kiekintveld et al., 2010] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, pages 133-140, 2010.
- [M. Vinyals et al., 2011] M. Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, and M. Bowring. Quality guarantees for region optimal DCOP algorithms. In Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems, pages 133-140, 2011.

References IV

- [Y. Weiss and W. T. Freeman, 2001] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. IEEE Transactions on Information Theory, 47(2):723-735, 2001.
- [M. Vinyals et al., 2010] M. Vinyals, J. Cerquides, A. Farinelli, and J. A. Rodriguez-Aguilar. Worst-case bounds on the quality of max-product fixed-points. In Neural Information Process- ing Systems, pages 2325-2333, 2010.
- [E. Bowring et al., 2008] E.Bowring, J.Pearce, C.Portway, M.Jain and M.Tambe. On k-optimal distributed constraint optimization algorithms: New bounds and algorithms. In Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent systems, pages 607-614, 2008.
- [A. Rogers et al., 2011] A. Rogers, A. Farinelli, R. Stranders, and N. R Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. Artificial Intelligence Journal, 175(2):730-759, 2011.