DISTRIBUTED CONSTRAINT OPTIMIZATION

AE4M36MAS - Multiagent systems

ASSIGNMENT

n-Queens problem in a distributed way

n queens from a $n \times n$ world had
a serious dispute:

n-Queens problem in a distributed way

n queens from a $n \times n$ world had a serious dispute:

- They don't want to know of each other (i.e. no queen wants to have any other in her line of sight)

n-Queens problem in a distributed way

n queens from a $n \times n$ world had
a serious dispute:

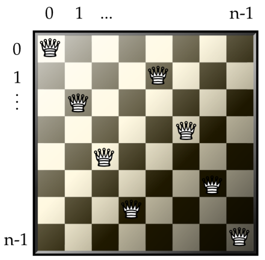
- They don't want to know of each other (i.e. no queen wants to have any other in her line of sight)
- They don't talk to each other except for few formal messages
  Ok? Nogood AddLink

n-Queens problem in a distributed way

n queens from a $n \times n$ world had a serious dispute:

- They don't want to know of each other (i.e. no queen wants to have any other in her line of sight)
- They don't talk to each other except for few formal messages Ok? Nogood AddLink

Help them to find their place in the world!



n-Queens problem in a distributed way

Every agent controls **one queen** and decides about her position within its row.

In the end, one of the following has to happen:

- One of the agents reports that no solutions exists
- Each queen reports her position in her row (i.e. a column in which it is located)

$\uparrow$ of course correctly ;-)

n-Queens problem in a distributed way

Any **asynchronous** and **distributed** solution is acceptable (e.g. ABT).

$\rightarrow$ No centralized knowledge allowed!

$\rightarrow$ No synchonization!

$\rightarrow$ No hardcoded solutions!

n-Queens problem in a distributed way

Total: **12 points**

- Solve 3×3 chessboard problem with 3 queens (3 points)
- Solve 4×4 chessboard problem with 4 queens (2 points)
- Solve 8×8 chessboard problem with 8 queens (2 points)
- Solve 12×12 chessboard problem with 12 queens (3 points)

n-Queens problem in a distributed way

Guaranteed termination detection (**1 point**)

- How to detect *quiescence* in an algorithmic way?
- You may want to get inspired by other DCSP/DCOP algorithms.

Quiescence should be discovered using **local knowledge** only.

$\rightarrow$ Sending whole solution to a single agent for verification is not an option!

n-Queens problem in a distributed way

Report (**1 point**)

- How is the n-queens problem modeled as a DCSP? (variables, domains, constraints, agents)
- How is the ABT algorithm customized for the n-queens problem?
- How do you determine priorities between agents?
- How do you detect that the search has terminated?

REVISION

Distributed CSP

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ — set of *variables* to assign
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ — set of *domains* ($x_i \in D_i$)
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ — set of *constraints*
- $\mathcal{A} = \{A_1, \ldots, A_k\}$ — set of *agents*

Distributed CSP

Agent i should come up with an assignment for his variable x_i in a **distributed** way.

Tuple $(x_1, \ldots, x_n)$ should satisfy all the constraints.

Asynchronous backtracking

Agents asynchronously decide about their variable and communicate their decisions.

Asynchronous backtracking

Agents asynchronously decide about their variable and communicate their decisions.

- **Ok?** asks lower priority subscribers whether current assignment is okay for them

Asynchronous backtracking

Agents asynchronously decide about their variable and communicate their decisions.

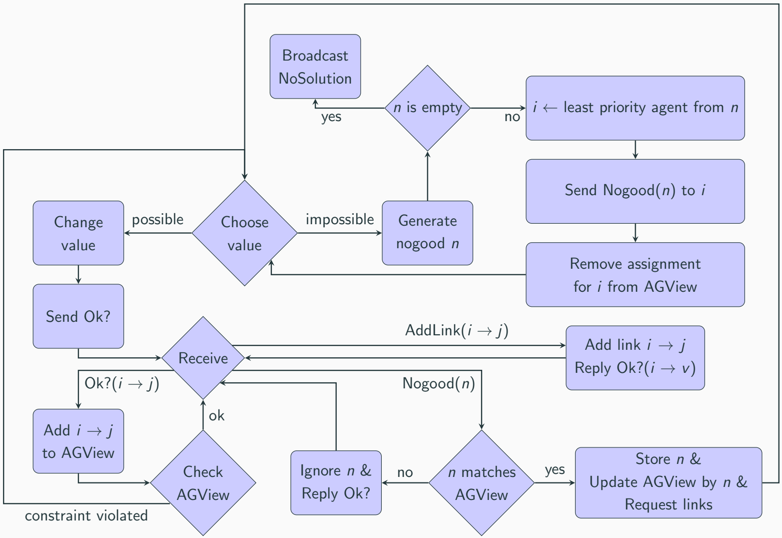
- **Ok?** asks lower priority subscribers whether current assignment is okay for them
- **Nogood** notifies one higher priority agent that he must take some action — otherwise a solution will not be found

Asynchronous backtracking

Agents asynchronously decide about their variable and communicate their decisions.

- **Ok?** asks lower priority subscribers whether current assignment is okay for them
- **Nogood** notifies one higher priority agent that he must take some action — otherwise a solution will not be found
- **AddLink** represents the subscription for a variable of a higher priority agent (when I am asked to check something I cannot check at the moment)

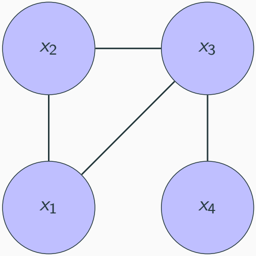
Asynchronous backtracking



DISTRIBUTED OPTIMIZATION

What we had so far?

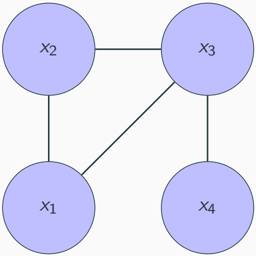
x_i	x_j	
○	○	T
○	●	F
●	○	F
●	●	T



$$C_k : D_i \times D_j \to \{\mathrm{T}, \mathrm{F}\}$$

What we have in DCOPs?

x_i	x_j	
○	○	1
○	●	2
●	○	2
●	●	0



$$C_k : D_i \times D_j \rightarrow \mathbb{N}_0$$

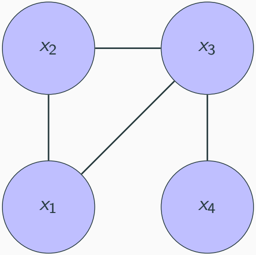
DCOPs

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ — set of *variables* to assign
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ — set of *domains* ($x_i \in D_i$)
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ — set of *constraints*
- $\mathcal{A} = \{A_1, \ldots, A_k\}$ — set of *agents*

Goal

$$\min_{\mathbf{x}} \sum_{C_i \in \mathcal{C}} C_i(\mathbf{x})$$

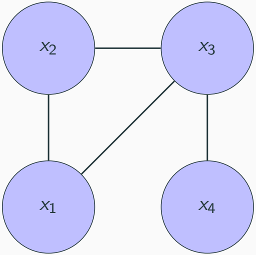
Branch & Bound



x_i	x_j	
○	○	1
○	●	2
●	○	2
●	●	0

Branch & Bound

Agent 1: $x_1 = \circ$
$LB = 0,\ UB = \infty$



x_i	x_j	
$\circ$	$\circ$	1
$\circ$	$\bullet$	2
$\bullet$	$\circ$	2
$\bullet$	$\bullet$	0

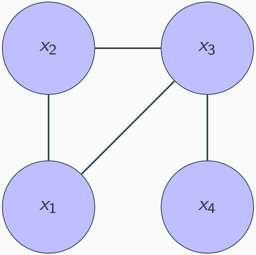
Branch & Bound

Agent 1: $x_1 = \circ$

$LB = 0, \; UB = \infty$

Agent 2: $x_2 = \circ$

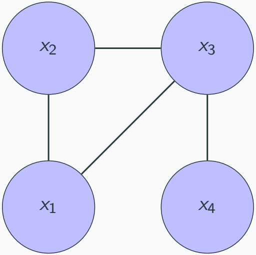
$LB = 1, \; UB = \infty$



x_i	x_j	
$\circ$	$\circ$	1
$\circ$	$\bullet$	2
$\bullet$	$\circ$	2
$\bullet$	$\bullet$	0

Branch & Bound

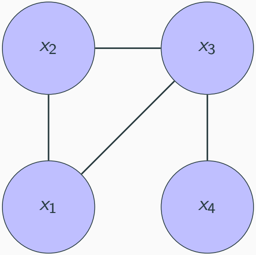
Agent 1: $x_1 = \circ$
$LB = 0,\ UB = \infty$
Agent 2: $x_2 = \circ$
$LB = 1,\ UB = \infty$
Agent 3: $x_3 = \circ$
$LB = 3,\ UB = \infty$



x_i	x_j	
$\circ$	$\circ$	1
$\circ$	$\bullet$	2
$\bullet$	$\circ$	2
$\bullet$	$\bullet$	0

Branch & Bound

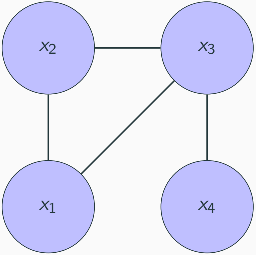
Agent 1: $x_1 = \circ$
$LB = 0,\ UB = 4$
Agent 2: $x_2 = \circ$
$LB = 1,\ UB = 4$
Agent 3: $x_3 = \circ$
$LB = 3,\ UB = 4$
Agent 4: $x_4 = \circ$
$LB = 4,\ UB = 4$



x_i	x_j	
$\circ$	$\circ$	1
$\circ$	$\bullet$	2
$\bullet$	$\circ$	2
$\bullet$	$\bullet$	0

Branch & Bound

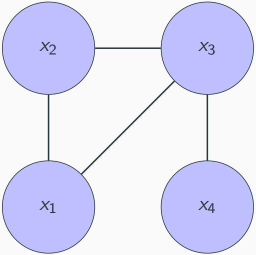
Agent 1: $x_1 = \circ$
$LB = 0,\ UB = 4$
Agent 2: $x_2 = \circ$
$LB = 1,\ UB = 4$
Agent 3: $x_3 = \circ$
$LB = 3,\ UB = 4$
Agent 4: $x_4 = \bullet$
$LB = \mathbf{5},\ UB = 4$



x_i	x_j	
○	○	1
○	●	2
●	○	2
●	●	0

Branch & Bound

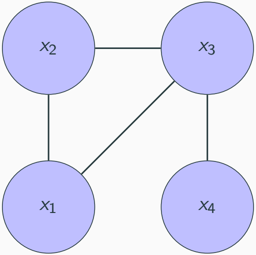
Agent 1: $x_1 = \circ$
$LB = 0,\ UB = 4$
Agent 2: $x_2 = \circ$
$LB = 1,\ UB = 4$
Agent 3: $x_3 = \bullet$
$LB = \mathbf{5},\ UB = 4$



x_i	x_j	
$\circ$	$\circ$	1
$\circ$	$\bullet$	2
$\bullet$	$\circ$	2
$\bullet$	$\bullet$	0

Branch & Bound

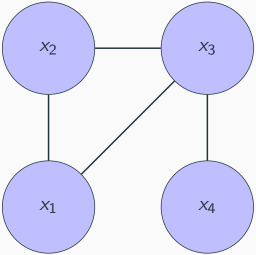
Agent 1: $x_1 = \circ$
$LB = 0,\ UB = 4$
Agent 2: $x_2 = \bullet$
$LB = 2,\ UB = 4$



x_i	x_j	
$\circ$	$\circ$	1
$\circ$	$\bullet$	2
$\bullet$	$\circ$	2
$\bullet$	$\bullet$	0

Branch & Bound

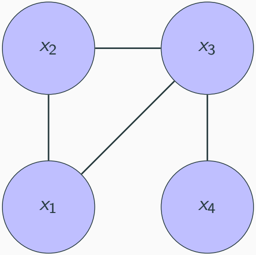
Agent 1: $x_1 = \circ$
$LB = 0,\ UB = 4$
Agent 2: $x_2 = \bullet$
$LB = 2,\ UB = 4$
Agent 3: $x_3 = \circ$
$LB = \mathbf{5},\ UB = 4$



x_i	x_j	
○	○	1
○	●	2
●	○	2
●	●	0

Branch & Bound

Agent 1: $x_1 = \circ$
$LB = 0,\ UB = 4$
Agent 2: $x_2 = \bullet$
$LB = 2,\ UB = 4$



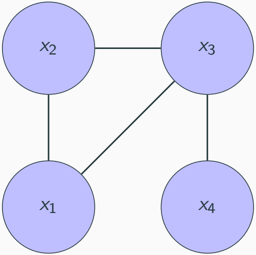
x_i	x_j	
$\circ$	$\circ$	1
$\circ$	$\bullet$	2
$\bullet$	$\circ$	2
$\bullet$	$\bullet$	0

Branch & Bound

Agent 1: $x_1 = \bullet$
$LB = 0,\ UB = 4$

... etc ...

LB=UB
$\rightarrow$ Solution found



x_i	x_j	
○	○	1
○	●	2
●	○	2
●	●	0

Branch & Bound

Why we **do not like** such an approach in MAS?

Branch & Bound

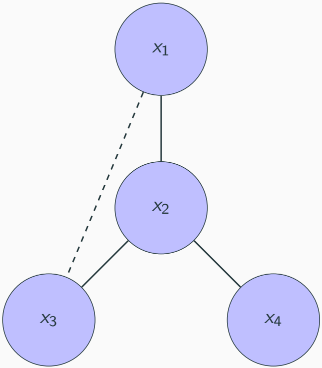
Why we **do not like** such an approach in MAS?

→ We need all agents to take decisions **simulataneously**!

Opportunistic Best-first Search

1. Introduce a hierarchy between agents

   DFS tree (back edges are dashed)



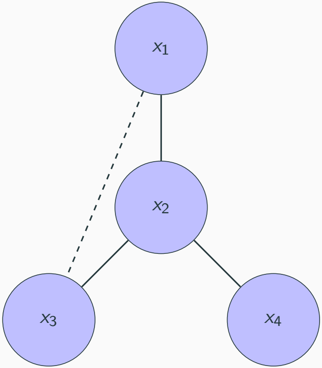
Opportunistic Best-first Search

Let $x_1 = \circ$.

Question

It's Christmas time! Assume that you can get any information about "subtrees" rooted in x_3 and x_4 at no cost.

What is the optimal assignment for x_2?



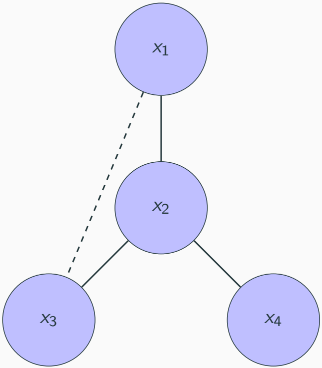
Opportunistic Best-first Search

Let $x_1 = \circ$.

Question

What is the optimal assignment for x_2?

$$\underset{v\in\{\circ,\bullet\}}{\arg\min}\Big[C(x_1=\circ,x_2=v) + OPT_{x_3}(x_1=\circ,x_2=v) + OPT_{x_4}(x_1=\circ,x_2=v)\Big]$$



Opportunistic Best-first Search

More generally:

$$\arg\min_{v \in D_i} \left[\delta_{ctx}(v) + \sum_{c \in child(i)} OPT_c\left(ctx \cup \{x_i = v\}\right) \right]$$

where

ctx	current context (assignment for i's ancestors) (~agent view)
$\delta_{ctx}(v)$	penalty for constraints involving x_i and some ancestor of i when $x_i = v$
$OPT_c(ctx)$	optimal solution of the subtree rooted in c in the given context

Opportunistic Best-first Search

There is a problem — we do not know $OPT_c(ctx)$

(otherwise we wouldn't be here right now ;-))

Inspire yourself in Branch & Bound algorithm!

Opportunistic Best-first Search

There is a problem — we do not know $OPT_c(ctx)$

(otherwise we wouldn't be here right now ;-))

Inspire yourself in Branch & Bound algorithm!

$\rightarrow$ Keep bounds on solutions of subtrees
(given my assignment)

Opportunistic Best-first Search

There is a problem — we do not know $OPT_c(ctx)$

(otherwise we wouldn't be here right now ;-))

Inspire yourself in Branch & Bound algorithm!

$\rightarrow$ Keep bounds on solutions of subtrees
(given my assignment)

Solution: Take the opportunity and pick the value that may lead to the best solution! (i.e. the one with minimal lower bound)

$$LB(v) = \delta_{ctx}(v) + \sum_{c \in child(i)} lb_c(v)$$

What we need to store?

For every my assignment:

What we need to store?

For every my assignment:
For every child of mine:

○	○	●	●
x_3	x_4	x_3	x_4

What we need to store?

For every my assignment:

For every child of mine:

Store bounds:

	$\circ$	$\circ$	$\bullet$	$\bullet$
	x_3	x_4	x_3	x_4
$lb_c(v)$	0	0	0	0
$ub_c(v)$	∞	∞	∞	∞

What we need to store?

For every my assignment:

For every child of mine:

Store bounds:

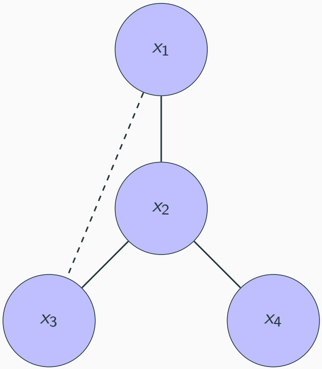
Context:

	$\circ$	$\circ$	$\bullet$	$\bullet$
	x_3	x_4	x_3	x_4
$lb_c(v)$	0	0	0	0
$ub_c(v)$	∞	∞	∞	∞
	$x_1 = \circ$	$x_1 = \circ$	$x_1 = \circ$	$x_1 = \circ$

Challenge

It's pre-2005 era. A complete asynchronous distributed algorithm for solving DCOPs is non-existent...

It's your turn to make ADOPT work!



ADOPT messages

ADOPT messages

- **value?**
  Agent notifies ancestors that he changed his value
  (only those interested!)

ADOPT messages

- **value?**
  Agent notifies ancestors that he changed his value
  (only those interested!)
- **cost!**
  Agent notifies his parent about bounds on the solution of his subtree

ADOPT messages

- **value?**
  Agent notifies ancestors that he changed his value
  (only those interested!)
- **cost!**
  Agent notifies his parent about bounds on the solution of his subtree
  Include context! Otherwise the whole system goes out of sync

ADOPT messages

- **value?**
  Agent notifies ancestors that he changed his value
  (only those interested!)
- **cost!**
  Agent notifies his parent about bounds on the solution of his subtree
  Include context! Otherwise the whole system goes out of sync
- **solution!**
  Broadcasted by root agent in the DFS tree when detecting LB=UB.

ADOPT messages

- **value?**
  Agent notifies ancestors that he changed his value
  (only those interested!)
- **cost!**
  Agent notifies his parent about bounds on the solution of his subtree
  Include context! Otherwise the whole system goes out of sync
- **solution!**
  Broadcasted by root agent in the DFS tree when detecting LB=UB.
- **threshold!** (optional)
  Sent to children not to make them swap their value too often.

ADOPT properties

Optimal and **asynchronous** algorithm for solving DCOPs.

Question: What is the key difference in the way ADOPT backtracks? (compared to ABT / synchronous BnB)

ADOPT properties

Optimal and **asynchronous** algorithm for solving DCOPs.

Question: What is the key difference in the way ADOPT backtracks? (compared to ABT / synchronous BnB)

- ABT backtrack when it has no other option
  (i.e. inconsistency has been proven)

ADOPT properties

Optimal and **asynchronous** algorithm for solving DCOPs.

Question: What is the key difference in the way ADOPT backtracks? (compared to ABT / synchronous BnB)

- ABT backtrack when it has no other option
  (i.e. inconsistency has been proven)
- BnB backtracks when suboptimality is detected
  (i.e. once LB $\geq$ UB)

ADOPT properties

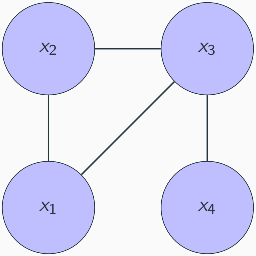
Optimal and **asynchronous** algorithm for solving DCOPs.

Question: What is the key difference in the way ADOPT backtracks? (compared to ABT / synchronous BnB)

- ABT backtrack when it has no other option
  (i.e. inconsistency has been proven)
- BnB backtracks when suboptimality is detected
  (i.e. once LB $\geq$ UB)
- ADOPT keeps informing parent about solution bounds
  (backtrack may happen due to the **opportunity** to change)

Example

x_i	x_j	
○	○	1
○	●	2
●	○	2
●	●	0



Approximate algorithms

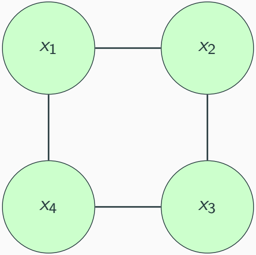
When we need solution **fast** and with **little effort**.

$\rightarrow$ Optimality guarantees are sacrificed

$\rightarrow$ Much better scalability

Approximate algorithms

At least some **coordination is needed**.

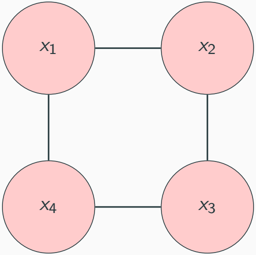


Graph coloring — each agent can decide to be either green or red.

Question: What is the best choice for each of the agents?

Approximate algorithms

At least some **coordination is needed**.



Graph coloring — each agent can decide to be either green or red.

Approximate algorithms

Recall of mining in Jason. How to solve this issue?

Approximate algorithms

Recall of mining in Jason. How to solve this issue?

- *Randomize to decide whether an agent is going to act.*
  $\rightarrow$ DSA-1 algorithm

Approximate algorithms

Recall of mining in Jason. How to solve this issue?

- *Randomize to decide whether an agent is going to act.*
  $\rightarrow$ DSA-1 algorithm
- *Negotiate with neighbors.*
  $\rightarrow$ MGM-1 algorithm

DSA-1 — Distributed stochastic algorithm

Toss a coin to decide whether:

- I will do the greedy step
- I will wait for others to do something

Keep exchanging individual assignments.