Multi-Agent Systems

Albert-Ludwigs-Universität Freiburg

Bernhard Nebel, Felix Lindner, and Thorsten Engesser

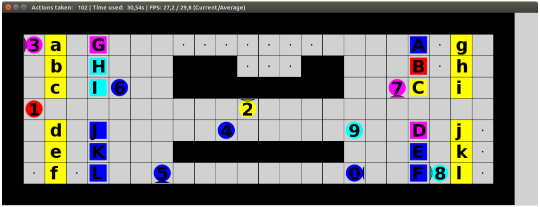
Summer Term 2017

Course outline

1. Introduction
2. Agent-Based Simulation
3. Agent Architectures
4. Beliefs, Desires, Intentions
5. Norms and Duties
6. Communication and Argumentation
7. Coordination and Decision Making
    - Distributed Constraint Satisfaction
    - Auctions and Markets
    - Cooperative Game Theory

Motivation

- Agents' abilities and/or preferences differ. How can they reach agreements?



- Argumentation Frameworks approach
  - Centralized approach: Agents exchange their arguments and then compute solution.
- Distributed Constraint Satisfaction approach
  - De-centralized: Agents hold private constraints and exchange partial solutions.

Constraint Satisfaction: Intro

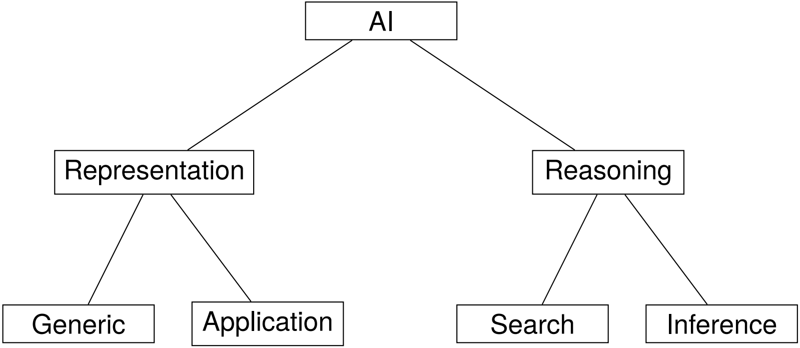
UNI FREIBURG

CSP (Freuder & Mackworth, 2006)

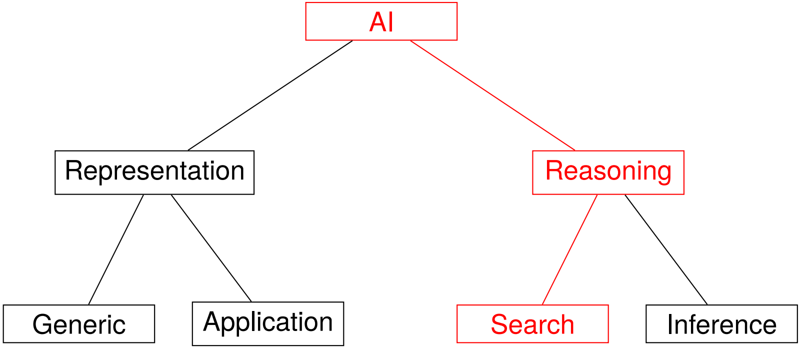
"Constraint satisfaction involves finding a value for each one of a set of problem variables where constraints specify that some subsets of values cannot be used together." ([1, p. 11])

- Examples:
  - Pick appetizer, main dish, wine, dessert such that everything fits together.
  - Place furniture in a room such that doors, windows, light switches etc. are not blocked.
  - ...

AI Research on Constraint Satisfaction



AI Research on Constraint Satisfaction



Constraint Satisfaction Problem

CSP

A CSP is a triple $\mathcal{P} = (X, D, C)$:

- $X = (x_1, \ldots, x_n)$: finite list of variables
- $D = (D_1, \ldots, D_n)$: finite domains
- $C = (C_1, \ldots, C_k)$: finite list of constraint predicates

- Variable x_i can take values from D_i
- Constraint predicate $C(x_i, \ldots, x_l)$ is defined on $D_i \times \ldots \times D_l$

- Unary constraints: $C(Wine) \leftrightarrow Wine \neq riesling$
- Binary constraints: $C(WineAppetizer, WineMainDish) \leftrightarrow WineAppetizer \neq WineMainDish$
- k-ary: $C(Alice, Bob, John) \leftrightarrow Alice \wedge Bob \rightarrow John$

CSP: Graph coloring

Problem statement

Given a graph $G = (V, E)$ and a set of colors N. Find a coloring $f : V \rightarrow N$ that assigns to each $v_i \in V$ a color different from those of its neighbors.

CSP formulation

Represent graph coloring as CSP $\mathcal{P} = (X, D, C)$:

- Each variable $x_i \in X$ represents the color of node $v_i \in V$
- Each $x_i \in X$ can get a value from its domain $D_i = N$
- For all $(x_i, x_j) \in E$ add a constraint $c(x_i, x_j) \leftrightarrow x_i \neq x_j$.

CSP: Graph coloring

Problem statement

Given a graph $G = (V, E)$ and a set of colors N. Find a coloring $f : V \rightarrow N$ that assigns to each $v_i \in V$ a color different from those of its neighbors.

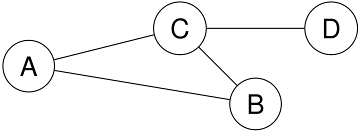
CSP formulation

Represent graph coloring as CSP $\mathcal{P} = (X, D, C)$:

- Each variable $x_i \in X$ represents the color of node $v_i \in V$
- Each $x_i \in X$ can get a value from its domain $D_i = N$
- For all $(x_i, x_j) \in E$ add a constraint $c(x_i, x_j) \leftrightarrow x_i \neq x_j$.

Graph coloring: Encoding

Colors: 1, 2, 3



CSP Encoding

Represention of this instance as a CSP $\mathcal{P} = (X, D, C)$:

- $X = (x_A, x_B, x_C, x_D)$
- $D = (\{1,2,3\}, \{1,2,3\}, \{1,2,3\}, \{1,2,3\})$
- $C(x_A, x_B) \leftrightarrow x_A \neq x_B$, $C(x_A, x_C) \leftrightarrow x_A \neq x_C$, $C(x_B, x_C) \leftrightarrow x_B \neq x_C$, $C(x_C, x_D) \leftrightarrow x_C \neq x_D$

Solution of a CSP

UNI FREIBURG

Definition

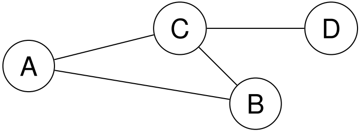
A solution of a CSP $\mathcal{P} = (X, D, C)$ is an assignment $a : X \to \bigcup_{i:x_i \in X} D_i$ such that:

- $a(x_i) \in D_i$ for each $x_i \in X$
- Every constraint $C(x_i, \ldots, x_m) \in C$ is evaluated true under $\{x_i \to a(x_i), \ldots, x_m \to a(x_m)\}$.

- $\mathcal{P}$ is satisfiable iff $\mathcal{P}$ has a solution.

Graph coloring: Solution

Colors: 1, 2, 3



Solutions

$a(x_A) = 1, a(x_B) = 2, a(x_C) = 3, a(x_D) = 1$
$a(x_A) = 1, a(x_B) = 2, a(x_C) = 3, a(x_D) = 2$
$a(x_A) = 2, a(x_B) = 1, a(x_C) = 3, a(x_D) = 1$
$\ldots$

- Here: 81 assignments, 12 solutions. Can we do better than listing all assignments?

CSP: NP-completeness (Sketch)

- Deciding if a graph (V, E) can be colored with k colors is known to be NP-complete.
- CSP is NP-complete:
  - NP-hardness: Polynomial-time reduction on slide 7.
  - Given an assignment, determining whether the assignment is a solution can be done in polynomial time (just check that all the $|E| \in O(|V|^2)$ constraints). □

- Remark: Graph coloring can also be solved by asking if there is a stable labelling for a corresponding argumentation framework (also a NP-complete problem). $\Rightarrow$CSPs can be solved using argumentation and vice-versa.

Using search

- In case of n variables with domains of size d there are $O(d^n)$ assignments.
- We can use all sorts of search algorithms to intelligently explore the space of assignments and to eventually find a solution.
- We will use backtracking search and employ two concepts:
  - Partial solution
  - Nogood

Partial solution of a CSP

Definition

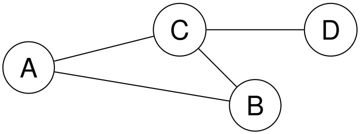
Given a CSP $\mathcal{P} = (X, D, C)$.

- An instantiation of a subset $X' \subseteq X$ is an assignment $a : X' \to \bigcup_{i: x_i \in X'} D_i$.
- An instantiation a of X' is a partial solution if a satisfies all constraints in C defined over some subset of X'. Then a is locally consistent.

- Hence, a solution is a locally consistent instantiation of all $x \in X$.

Graph coloring: Partial Solution

Colors: 1, 2, 3



Locally consistent partial solutions

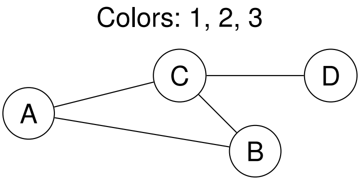
$a(x_A) = \perp, a(x_B) = \perp, a(x_C) = \perp, a(x_D) = \perp$
$a(x_A) = 1, a(x_B) = \perp, a(x_C) = \perp, a(x_D) = \perp$
$a(x_A) = 1, a(x_B) = 2, a(x_C) = \perp, a(x_D) = \perp$
$a(x_A) = 1, a(x_B) = 2, a(x_C) = 3, a(x_D) = \perp$
$a(x_A) = 1, a(x_B) = 2, a(x_C) = 3, a(x_D) = 1$

Nogoods

Definition

Given a CSP $\mathcal{P} = (X, D, C)$. An instantiation a' of $X' \subseteq X$ is a nogood of $\mathcal{P}$ iff a' cannot be extended to a full solution of $\mathcal{P}$.

Graph coloring: Nogood



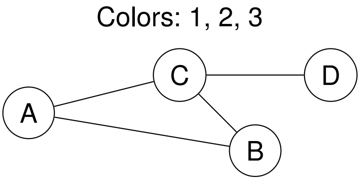
Nogood

$a(x_A) = 1, a(x_B) = 1, a(x_C) = \perp, a(x_D) = \perp$

Backtracking Algorithm

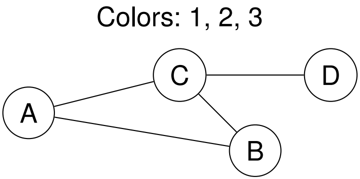
```
function BT(𝒫, part_sol)
    if isSolution(part_sol) then
        return part_sol
    end if
    if isNoGood(part_sol, 𝒫) then
        return false
    end if
    select some x_j so far undefined in part_sol
    for possible values d ∈ D_j for x_j do
        par_sol ← BT(𝒫, par_sol[x_j|d])
        if par_sol ≠ False then
            return par_sol
        end if
    end for
    return False
end function
```

Graph coloring: Backtracking



$BT(\mathcal{P}, \{\})$

Graph coloring: Backtracking

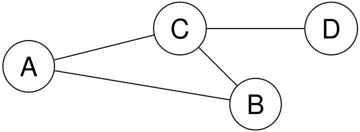


$BT(\mathcal{P}, \{\})$

$BT(\mathcal{P}, \{x_A \rightarrow 1\})$

Graph coloring: Backtracking

Colors: 1, 2, 3



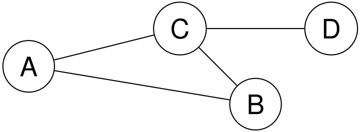
$BT(\mathcal{P}, \{\})$

$BT(\mathcal{P}, \{x_A \rightarrow 1\})$

$BT(\mathcal{P}, \{x_A \rightarrow 1, x_B \rightarrow 1\})$

Graph coloring: Backtracking

Colors: 1, 2, 3

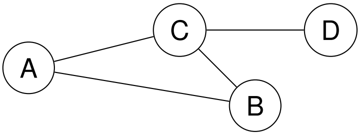


$BT(\mathcal{P}, \{\})$

$BT(\mathcal{P}, \{x_A \to 1\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 1\})$ $\quad$ $BT(\mathcal{P}, \{x_A \to 1, x_B \to 2\})$

Colors: 1, 2, 3



$BT(\mathcal{P}, \{\})$

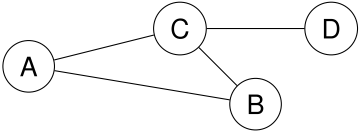
$BT(\mathcal{P}, \{x_A \to 1\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 1\})$ $BT(\mathcal{P}, \{x_A \to 1, x_B \to 2\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 1\})$

Graph coloring: Backtracking

Colors: 1, 2, 3



$BT(\mathcal{P}, \{\})$

$BT(\mathcal{P}, \{x_A \to 1\})$

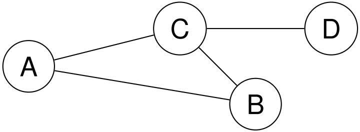
$BT(\mathcal{P}, \{x_A \to 1, x_B \to 1\})$ $BT(\mathcal{P}, \{x_A \to 1, x_B \to 2\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 1\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 2\})$

Graph coloring: Backtracking

Colors: 1, 2, 3



$BT(\mathcal{P}, \{\})$

$BT(\mathcal{P}, \{x_A \to 1\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 1\})$ $BT(\mathcal{P}, \{x_A \to 1, x_B \to 2\})$

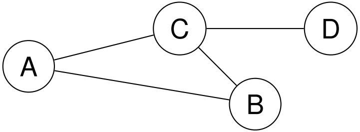
$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 1\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 2\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 3\})$

Graph coloring: Backtracking

Colors: 1, 2, 3



$BT(\mathcal{P}, \{\})$

$BT(\mathcal{P}, \{x_A \to 1\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 1\})$ $BT(\mathcal{P}, \{x_A \to 1, x_B \to 2\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 1\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 2\})$

$BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 3\})$

$\boxed{BT(\mathcal{P}, \{x_A \to 1, x_B \to 2, x_C \to 3, x_D \to 1\})}$

An MAS Example

- Nodes A, B, C, and D represent families living in a neighborhood. An edge between two nodes models that the represented families are direct neighbors. Each family wants to buy a new car, but they don't want their respective neighbors to own the same car as they do.
- Centralized solution: A, B, C, D meet, make their constraints public and find a solution together.
- Decentralized solution: A, B, C, D do not meet. Instead, they just buy cars. If someone dislikes one other's choice (s)he will either buy another one or tell the neighbor to do so (without telling why).

Distributed Constraint Satisfaction (DisCSP): Motivation

- Centralized agent decision making encoded as CSP:
  - Each variable stands for the action of an agent. Constraints between variables model the interrelations between the agents' actions. A CSP solver solves the CSP and communicates the result to each of the agents.
- This, however, presupposes a central component that knows about all the variables and constraints. So what?
  - In some applications, gathering all information to one component is undesirable or impossible, e.g., for security/privacy reasons, because of too high communication costs, because of the need to convert internal knowledge into an exchangeable format.
- ⇒Distributed Constraint Satisfaction (DisCSP)

Distributed Constraint Satisfaction Problem

CSP

A DistCSP is a tuple $\mathcal{P} = (A, X, D, C)$:

- $A = (ag_1, \ldots, ag_n)$: finite list of agents
- $X = (x_1, \ldots, x_n)$: finite list of variables
- $D = (D_1, \ldots, D_n)$: finite list of domains
- $C = (C_1, \ldots, C_k)$: finite list of constraint predicates

- Variable x_i can take values from D_i
- Constraint predicate $C(x_i, \ldots, x_l)$ is defined on $D_i \times \ldots \times D_l$
- Variable x_i belongs (only) to agent ag_i
- Agent ag_i knows all constraints on x_i

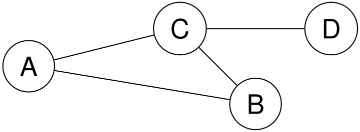
DisCSP: Solution

Definition

- An assignment a is a solution to a distributed CSP (DisCSP) instance if and only if:
  - Every variable x_i has some assigned value $d \in D_i$, and
  - For all agents ag_i: Every constraint predicate that is known by ag_i evaluates to **true** under the assignment $a(x_i) = d$

Example as a DisCSP

Colors: 1, 2, 3



Encoding

- $\mathcal{A} = (A, B, C, D)$, $X = (x_A, x_B, x_C, x_D)$, $D_A = \{1, 2, 3\}$, $D_B = \{1\}$, $D_C = \{2, 3\}$, $D_D = \{3\}$
- Constraints
  - $A : x_A \neq x_B, x_A \neq x_C$
  - $B : x_B \neq x_A, x_B \neq x_C$
  - $C : x_C \neq x_A, x_C \neq x_B, x_C \neq x_D$
  - $D : x_D \neq x_C$

Synchronous Backtracking

- Modification of the backtracking algorithm
  1. Agents agree on an instantiation order for their variables (x_1 goes first, then goes x_2 etc.)
  2. Each agent receiving a partial solution instantiates its variable based on the constraints it knows about
  3. If the agent finds such a value it will append it to the partial solution and pass it on to the next agent
  4. Otherwise, it sends a backtracking message to the previous agent

Synchronous Backtracking: Example Trace

1. A, B, C, and D agree on acting in this order
2. A sets x_A to 1 and sends $\{x_A \to 1\}$ to B
3. B sends *backtrack!* to A
4. A sets x_A to 2 and sends $\{x_A \to 2\}$ to B
5. B sets x_B to 1 and sends $\{x_A \to 2, x_B \to 1\}$ to C
6. C sets c_C to 3 and sends $\{x_A \to 2, x_B \to 1, x_C \to 3\}$ to D
7. D sends *backtrack!* to C
8. C sends *backtrack!* to B
9. B sends *backtrack!* to A
10. A sets x_A to 3 and sends $\{x_A \to 3\}$ to B
11. B sets x_B to 1 and sends $\{x_A \to 3, x_B \to 1\}$ to C
12. C sets x_C to 2 and sends $\{x_A \to 3, x_B \to 1, x_C \to 2\}$ to D
13. D sets x_D to 3.

Synchronous Backtracking: Pro/Con

- Pro: No need to share private constraints and domains with some centralized decision maker
- Con: Determining instantiation order requires communication costs
- Con: Agents act sequentially instead of taking advantage of parallelism, i.e., at any given time, only one agent is receiving a partial solution and acts on it

Asynchronous Backtracking

- Each agent maintains three properties:
  - *current_value*: value of its owned variable (subject to revision)
  - *agent_view*: what the agent knows so far about the values of other agents
  - *constraint_list*: ist of private constraints and received nogoods

- Each agent *i* can send messages of two kinds:
  - *(ok?, $x_j \rightarrow d$)*
  - *(nogood!, i, {$x_j \rightarrow d_j, x_k \rightarrow d_k, \ldots$})*

Asynchronous Backtracking: Assumption

- Assumption: For each contraint, there is one evaluating agent and one value sending agent. Hence, the graph is directed!
  - In some applications this may be naturally so (e.g., only one of the agents actually cares about the constraint)
  - In other applications, two agents involved in a constraint have to decide who will be the sender/evaluator.

Asynchronous Backtracking

if received (ok?, (x_j, d_j)) **then**
    add (x_j, d_j) to *agent_view*
    CHECKAGENTVIEW( )
end if

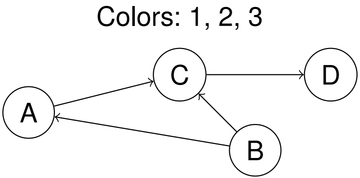
function CHECKAGENTVIEW
    if *agent_view* and *current_value* are not consistent **then**
        if no value in D_i is consistent with *agent_view* **then**
            BACKTRACK( )
        else
            select $d \in D_i$ s.th. *agent_view* and d consistent
            $current\_value \leftarrow d$
            send (ok?, (x_i, d)) to outgoing links
        end if
    end if
end function

Asynchronous Backtracking (cont.)

function BACKTRACK
  if $\emptyset$ is a nogood **then**
    broadcast that there is no solution and terminate
  end if
  generate a nogood V (inconsistent subset of *agent_view*)
  select $(x_j, d_j) \in V$ s.th. x_j has lowest priority in V
  send (nogood!, x_i, V) to x_j; remove (x_j, d_j) from *agent_view*
end function

if received (nogood!, x_j, {*nogood*})) **then**
  add *nogood* to *constraint_list*
  if *nogood* contains agent x_k that is not yet a neighbor **then**
    add x_k as neighbor and ask x_k to add x_i as neighbor
  end if
  CHECKAGENTVIEW( )
end if

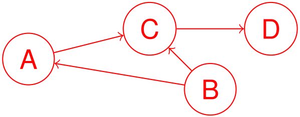
Asynchronous Backtracking: Example



- The graph is now directed (source: sender agent, sink: evaluator agent). All other things the same as before.

Example Trace

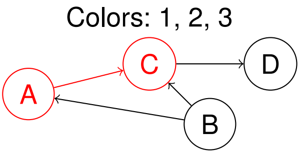
Colors: 1, 2, 3



1 Each agent initializes its private variable and sends ok?-messages down the links

Agent	Current Value	Agent View	Constraint List
A	1	$\{x_B \to 1\}$	$x_A \neq x_B$
B	1	$\emptyset$	$\emptyset$
C	2	$\{x_A \to 1, x_B \to 1\}$	$x_C \neq x_A, x_C \neq x_B$
D	3	$\{x_C \to 2\}$	$x_D \neq x_C$

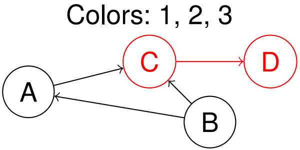
Example Trace



2 Agent A changes its value to 2 and sends ok? to C

Agent	Current Value	Agent View	Constraint List
A	2	$\{x_B \to 1\}$	$x_A \neq x_B$
B	1	$\emptyset$	$\emptyset$
C	2	$\{x_A \to 2, x_B \to 1\}$	$x_C \neq x_A, x_C \neq x_B$
D	3	$\{x_C \to 2\}$	$x_D \neq x_C$

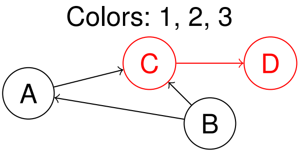
Example Trace



3 Agent C changes its value to 3 and sends ok? to D

Agent	Current Value	Agent View	Constraint List
A	2	$\{x_B \rightarrow 1\}$	$x_A \neq x_B$
B	1	$\emptyset$	$\emptyset$
C	3	$\{x_A \rightarrow 2, x_B \rightarrow 1\}$	$x_C \neq x_A, x_C \neq x_B$
D	3	$\{x_C \rightarrow 3\}$	$x_D \neq x_C$

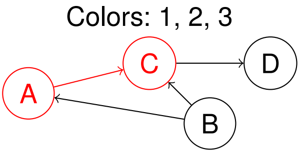
Example Trace



4 Agent D sends (nogood!, D, $\{x_c \to 3\}$) to C

Agent	Current Value	Agent View	Constraint List
A	2	$\{x_B \to 1\}$	$x_A \neq x_B$
B	1	$\emptyset$	$\emptyset$
C	3	$\{x_A \to 2, x_B \to 1\}$	$x_C \neq x_A, x_C \neq x_B, x_C \neq 3$
D	3	$\emptyset$	$x_D \neq x_C$

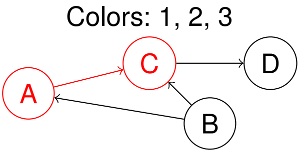
Example Trace



5 Agent C sends (nogood!, C, $\{x_A \to 2\}$) to A

Agent	Current Value	Agent View	Constraint List
A	2	$\{x_B \to 1\}$	$x_A \neq x_B, x_A \neq 2$
B	1	$\emptyset$	$\emptyset$
C	3	$\{x_B \to 1\}$	$x_C \neq x_A, x_C \neq x_B, x_C \neq 3$
D	3	$\emptyset$	$x_D \neq x_C$

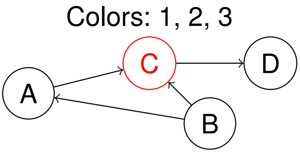
Example Trace



6 Agent A sets value to 3 and sends ok? to C

Agent	Current Value	Agent View	Constraint List
A	3	$\{x_B \rightarrow 1\}$	$x_A \neq x_B, x_A \neq 2$
B	1	$\emptyset$	$\emptyset$
C	3	$\{x_A \rightarrow 3, x_B \rightarrow 1\}$	$x_C \neq x_A, x_C \neq x_B, x_C \neq 3$
D	3	$\emptyset$	$x_D \neq x_C$

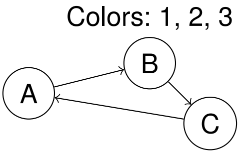
Example Trace



7 Agent C sets value to 2 and sends ok? to D

Agent	Current Value	Agent View	Constraint List
A	3	$\{x_B \to 1\}$	$x_A \neq x_B, x_A \neq 2$
B	1	$\emptyset$	$\emptyset$
C	2	$\{x_A \to 3, x_B \to 1\}$	$x_C \neq x_A, x_C \neq x_B, x_C \neq 3$
D	3	$\{x_C \to 2\}$	$x_D \neq x_C$

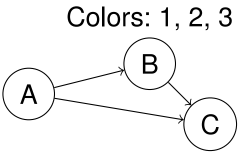
Loops



1. A, B, and C set their variables to 1 and send ok?
2. A, B, and C set their variables to 2 and send ok?
3. A, B, and C set their variables to 1 and send ok?
4. ...

Avoiding Loops

- Postulate an order over the agents (e.g., IDs). Based on that order, e.g., a link always goes from a higher-order to a lower-order agent.



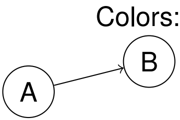
1. A, B, and C set their variables to 1, A and B send ok?
2. B and C set their variables to 2, B sends ok?
3. C sets its variable to 3

The empty Nogood

Theorem (see [2])

The CSP is unsatisfiable iff the empty Nogood is generated.

- Example of an empty nogood:



1. A and B set their variables to 1, A sends ok?
2. B sends (nogood!, $x_A \to 1$)
3. A generates a nogood, and as A's agent view is empty, the generated nogood is empty as well.

Summary and Outlook

- This time
  - Constraint Satisfaction Problem & Backtracking algorithm
  - Distributed Constraint Satisfaction Problem & Synchronous and Asynchronous Backtracking
- Next time
  - Agents express preferences by numbers $\Rightarrow$Optimal assigments using Auctions & Markets

Course outline

1. Introduction
2. Agent-Based Simulation
3. Agent Architectures
4. Beliefs, Desires, Intentions
5. Norms and Duties
6. Communication and Argumentation
7. Coordination and Decision Making
   - Distributed Constraint Satisfaction
   - Auctions and Markets
   - Cooperative Game Theory

Literature I

- E. C. Freuder, A. K. Mackworth, Constraint satisfaction: An emerging paradigm, In F. Rossi, P. van Beek, T. Walsh (Eds.) Handbook of Constraint Programming, Elsevier, 2006.
- M. Yokoo, T. Ishida, E. H. Durfee, K. Kuwabara, Distributed constraint satisfaction for formalizing distributed problem solving, In 12th IEEE International Conference on Distributed Computing Systems '92, pp. 614–621, 1992.
- M. Yokoo, K. Hirayama, Algorithms for distributed constraint satisfaction: A review, Autonomous Agents and Multi-Agent Systems, Vol. 3, No. 2, pp. 198–212, 2000.