Distributed Constraint Processing
An Introduction

Gauthier Picard

MINES Saint-Étienne
LaHC UMR CNRS 5516
gauthier.picard@emse.fr

— Some contents taken from OPTMAS 2011 and OPTMAS-DCR 2014 Tutorials—
Contents

Introduction
  Constraint Satisfaction Problems
  Multi-Agent Approaches to DisCSP

ABT and Extensions
  Asynchronous Algorithms for DisCSP
  ABT
  AWCS

Distributed Local Search
  Classical Centralised LS Algorithms
  Distributed Breakout Algorithm (DBA)
  Environment, Reactive rules and Agents (ERA)

Synthesis
  Panorama
  Using Distributed Problem Solving
Introduction

Motivations

- Multi-agent systems are a way to model decentralised problem solving (privacy, distribution)
- Agents, having personal goals and constraints, negotiate as to reach a global equilibrium

⇒ distributed problem solving using agents

Approaches

- Classical CSP solver extensions
- Classical local search solver extensions
Cooperative Decentralized Decision Making

- **Decentralised Decision Making**
  - Agents have to coordinate to perform best actions

- **Cooperative settings**
  - Agents form a team → best actions for the team

- **Why DDM in cooperative settings is important**
  - Surveillance (target tracking, coverage)
  - Robotics (cooperative exploration)
  - Autonomous cars (cooperative traffic management)
  - Scheduling (meeting scheduling)
  - Rescue Operation (task assignment)
Distributed Constraint Optimisation Problems (DCOPs) for DDM

Why DCOPs for Cooperative DDM?

- Well defined problem
  - Clear formulation that captures most important aspects
  - Many solution techniques
    - Optimal: ABT, ADOPT, DPOP, ...
    - Approximate: DSA, MGM, Max-Sum, ...

- Solution techniques can handle large problems
  - compared for example to sequential decision making (MDP, POMDP)
Modeling Problems as DCOP

- Target Tracking
- Meeting Scheduling

Why decentralize
  - Robustness to failure and message loss
  - Privacy
Target Tracking as a DCOP

- Variables → Cameras
- Domains → Camera actions
  - look left, look right
- Constraints
  - Overlapping cameras
  - Related to targets
    - Diabolik, Eva
- Maximise sum of constraints
Meeting Scheduling as a DCOP

Window 13:00 – 20:00
Duration 1h

Window 15:00 – 18:00
Duration 2h

Better after 18:00
Meeting Scheduling as a DCOP

[13 – 20]
19:00
[15 – 18]
16:00

No overlap (Hard)
Equals (Hard)
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

CSP
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

CSP

Yes

No
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
Definition (CSP)
A CSP is a triplet $\langle X, D, C \rangle$ such as:
- $X = \{x_1, \ldots, x_n\}$ is the set of variables to instantiate.
- $D = \{D_1, \ldots, D_m\}$ is the set of domains. Each variable $x_i$ is related to a domain of value.
- $C = \{c_1, \ldots, c_k\}$ is the set of constraints, which are relations between some variables from $X$ that constrain the values the variables can be simultaneously instantiated to.

Definition (Solution to a CSP)
A solution to a CSP is a complete assignment of values from $D$ to variables from $X$ such that every constraint in $C$ is satisfied.
Issues in CSP

Classical CSPs

- Constraint satisfaction is NP-complete in general
- Constraints are generally expressed as binary constraints
- The topology of a constraint-based problem can be represented by a constraint network, in which vertexes represent variables and edges represent binary constraints between variables

Extensions

- Distribution: variables, constraints
  - ex.: constraint $c_i$ belongs to stakeholder $j$, $\phi(c_i) = j$ (or $\text{belongs}(c_i, j)$)
- Dynamics: adding removing variables and/or constraints at runtime
Multi-Agent Approaches to CSP

- **Complete and asynchronous solvers** for combinatorial problems, within the DisCSP framework, such as Asynchronous Backtracking (ABT) or Asynchronous Weak-Commitment Search (AWCS)
- **Distributed local search** methods, such as Distributed Breakout Algorithm (DBA) or Environment, Reactive rules and Agents (ERA) approach
Asynchronous Algorithms for DisCSP

Idea

- Inspired by classical centralised algorithms to solve CSP
- Each agent is responsible for assigning one (or several) variables
- Agents propose values to some other agents (depending on the organisation i.e. constraint network)

Main algorithm: Asynchronous backtracking (ABT) [Yokoo, 2001]

- Agents will perform a distributed version of the backtracking procedure
- ABT is complete
- Extensions exist to handle dynamics

Definition (DisCSP or DCSP)

A DisCSP (or DCSP) is a 5-uplet \( \langle A, X, D, C, \phi \rangle \) where \( \langle X, D, C \rangle \) is a CSP, \( A \) is a set of agents and \( \phi : X \mapsto A \) is a function assigning variables from \( X \) to agents from \( A \).
Centralised Backtracking

\[
i \leftarrow 0
D'_i \leftarrow D_i
\]

\[\text{while } 0 \leq i < n \text{ do}\]
\[
x_i \leftarrow \text{null}
ok? \leftarrow \text{false}
\]
\[\text{while not } ok? \text{ and } D'_i \text{ not empty do}\]
\[
a \leftarrow \text{a value from } D'_i
\]
\[
\text{remove } a \text{ from } D'_i
\]
\[
\text{if } a \text{ is in conflict with } \{x_0, \ldots, x_{i-1}\} \text{ then}\]
\[
x_i \leftarrow a
\]
\[
ok? \leftarrow \text{true}
\]
\[\text{end}\]
\[\text{end}\]
\[
\text{if } x_i \text{ is null then backtrack}\]
\[
i \leftarrow i - 1
\]
\[
\text{else}\]
\[
i \leftarrow i + 1
\]
\[
D'_i \leftarrow D_i
\]
\[\text{end}\]
\[\text{end}\]
Asynchronous Backtracking (ABT) [Yokoo, 2001]

- First complete asynchronous algorithm for DisCSP solving
- Asynchronous:
  - All agents active, take a value and inform
  - No agent has to wait for other agents
- Total order among agents: to avoid cycles
  - $i < j < k$ means that: $i$ more priority than $j$, $j$ more priority than $k$
- Constraints are directed, following total order
- ABT plays in asynchronous distributed context the same role as backtracking in centralized
ABT: Directed Constraints

- Directed: from higher to lower priority agents
- Higher priority agent ($j$) informs the lower one ($k$) of its assignment
- Lower priority agent ($k$) evaluates the constraint with its own assignment
  - If permitted, no action
  - else it looks for a value consistent with $j$
    - If it exists, $k$ takes that value
    - else, the agent view of $k$ is a nogood, backtrack
ABT: Directed Constraints

- Directed: from higher to lower priority agents
- Higher priority agent \((j)\) informs the lower one \((k)\) of its assignment
- Lower priority agent \((k)\) evaluates the constraint with its own assignment
  - If permitted, no action
  - else it looks for a value consistent with \(j\)
    - If it exists, \(k\) takes that value
    - else, the agent view of \(k\) is a nogood, backtrack

generates nogoods: eliminate values of \(k\)
Definition (Nogood)
Conjunction of (variable, value) pairs of higher priority agents, that removes a value of the current one

Example

- $x \neq y$, $d_x = d_y = \{a, b\}$, $x$ higher than $y$
- When $[x \leftarrow a]$ arrives to $y$, this agent generates the nogood $[x = a \Rightarrow y \neq a]$ that removes value $a$ of $d_y$
- If $x$ changes value, when $[x \leftarrow b]$ arrives to $y$, the nogood $[x = a \Rightarrow y \neq a]$ is eliminated, value $a$ is again available and a new nogood removing $b$ is generated
ABT: Nogood Resolution

- When all values of variable $y$ are removed, the conjunction of the left-hand sides of its nogoods is also a nogood
- **Resolution**: the process of generating the new nogood

**Example**

- $x \neq y, z \neq y, d_x = d_y = d_z = \{a, b\}, x, z$ higher than $y$
  
  $x = a \Rightarrow y \neq a$  
  
  $z = b \Rightarrow y \neq b$

  
  $x = a \land z = b$ is a nogood

  
  $x = a \Rightarrow z \neq b$ (assuming $x$ higher than $z$)
How ABT works

- **ABT agents**: asynchronous action, spontaneous assignment
- **Assignment**: \( j \) takes value \( a \), \( j \) informs lower priority agents
- **Backtrack**: \( k \) has no consistent values with high priority agents, \( k \) resolves nogoods and sends a backtrack message
- **New links**: \( j \) receives a nogood mentioning \( i \), unconnected with \( j \); \( j \) asks \( i \) to set up a link
- **Stop**: “no solution” detected by an agent, stop
- **Solution**: when agents are silent for a while (quiescence), every constraint is satisfied \( \rightarrow \) solution; detected by specialized algorithms
ABT: Messages

Ok

In ABT, a message from node $i$ to node $k$ with value $a$ means:
- $i$ informs $k$ that it takes value $a$.
- All $k$ values are forbidden.
- $k$ requests $j$ to backtrack.
- $k$ forgets $j$ value.
- $k$ takes some value.
- $j$ may detect obsolescence.

Addl from $j$ to $i$:
- Set a link from $i$ to $j$, to know $i$ value.

Stop:
- There is no solution.
**ABT: Messages**

- \textit{Ok?}(i \rightarrow k, a): 
  - \(i\) informs \(k\) that it takes value \(a\)

\[
\text{Ngd}\left(k \rightarrow j, i = a \Rightarrow j \neq b\right)
\]

\(\Rightarrow\) all \(k\) values are forbidden

\(\Rightarrow\) \(k\) requests \(j\) to backtrack

\(\Rightarrow\) \(k\) forgets \(j\) value

\(\Rightarrow\) \(k\) takes some value

\(\Rightarrow\) \(j\) may detect obsolescence

- \textit{Addl}\((j \rightarrow i)\)
  - set a link from \(i\) to \(j\), to know \(i\) value

- **Stop:**
  - there is no solution
ABT: Messages

- \textit{Ok}(i \rightarrow k, a):  
  - \(i\) informs \(k\) that it takes value \(a\)

- \textit{Ngd}(k \rightarrow j, i = a \Rightarrow j \neq b)  
  - all \(k\) values are forbidden
  - \(k\) requests \(j\) to backtrack
  - \(k\) forgets \(j\) value
  - \(k\) takes some value
  - \(j\) may detect obsolescence
**ABT: Messages**

- **Ok?**(\(i \rightarrow k, a\)):
  - \(i\) informs \(k\) that it takes value \(a\)

- **Ngd**(\(k \rightarrow j, i = a \Rightarrow j \neq b\))
  - all \(k\) values are forbidden
  - \(k\) requests \(j\) to backtrack
  - \(k\) forgets \(j\) value
  - \(k\) takes some value
  - \(j\) may detect obsolescence

- **Addl**(\(j \rightarrow i\)):
  - set a link from \(i\) to \(j\), to know \(i\) value
**ABT: Messages**

- **Ok?(i → k, a):**
  - i informs k that it takes value a

- **Ngd(k → j, i = a ⇒ j ≠ b)**
  - all k values are forbidden
  - k requests j to backtrack
  - k forgets j value
  - k takes some value
  - j may detect obsolescence

- **Addl(j → i):**
  - set a link from i to j, to know i value

- **Stop:**
  - there is no solution
ABT Procedures

when received (ok?, (x, d)) do — (i)
    revise agent_view;
    check_agent_view;
end do;

when received (nogood, x, nogood) do — (ii)
    record nogood as a new constraint;
    when nogood contains an agent x that is not its neighbor
        do request x to add x as a neighbor,
        and add x to its neighbors; end do;
    old_value ← current_value; check_agent_view;
    when old_value = current_value do
        send (ok?, (x, current_value)) to x; end do; end do;

procedure check_agent_view
    when agent_view and current_value are not consistent do
        if no value in D, is consistent with agent_view then backtrack;
        else select d ∈ D, where agent_view and d are consistent;
            current_value ← d;
            send (ok?, (x, d)) to neighbors; end if; end do;

procedure backtrack
    generate a nogood V — (iii)
    when V is an empty nogood do
        broadcast to other agents that there is no solution,
        terminate this algorithm; end do;
    select (x, d) where x has the lowest priority in a nogood;
    send (nogood, x, V) to x;
    remove (x, d) from agent_view;
    check_agent_view;

Algorithm 2: ABT Procedures
ABT: Correctness and Completeness

**Correctness**
- silent network $\iff$ all constraints are satisfied

**Completeness**
- ABT performs an exhaustive traversal of the search space
- Parts not searched: those eliminated by nogoods
- Nogoods are legal: logical consequences of constraints
- Therefore, either there is no solution $\Rightarrow$ ABT generates the empty nogood, or it finds a solution if exists
ABT: Remarks

- Fixed ordered organisation
  - Agents only communicate with agents with lower priority for `ok`
  - Agents only communicate with the agent with direct higher priority for `nogood`
- No termination procedure is given (but it is easily implemented using Dijkstra’s tokens)
- Really distributable
- What if $x_0$ disappears?...

Extensions and Filiation

- **Changing ordering** in every conflict with AWCS [YOKOO, 2001]
- Satisfaction $\rightarrow$ **Optimisation** with ADOPT (Asynchronous B&B) [MODI et al., 2005] or APO [MAILLER and LESSER, 2006]
- **Adding new agents** at runtime in DynAPO [MAILLER, 2005]
Asynchronous Weak-Commitment Search (AWCS) [YOKOO, 2001]

procedure check_agent_view
  when agent_view and current_value are not consistent do
    if no value in \( D_i \) is consistent with agent_view then backtrack;
    else select \( d \in D_i \) where agent_view and \( d \) are consistent
      and \( d \) minimizes the number of constraint violations
        with lower priority agents; — (i)
        current_value \( \leftarrow d; \)
        send (ok?, \((x_i, d, current\_priority)\)) to neighbors;
  end if; end do;

procedure backtrack
  generate a nogood \( V \);
  when \( V \) is an empty nogood do
    broadcast to other agents that there is no solution,
    terminate this algorithm; end do;
  when \( V \) is a new nogood do — (ii)
    send \( V \) to the agents in the nogood;
    current_priority \( \leftarrow 1 + p_{\text{max}}, \)
      where \( p_{\text{max}} \) is the maximal priority value of neighbors;
    select \( d \in D_i \) where agent_view and \( d \) are consistent,
      and \( d \) minimizes the number of constraint violations
        with lower priority agents;
    current_value \( \leftarrow d; \)
    send (ok?, \((x_i, d, current\_priority)\)) to neighbors; end do;

Algorithm 3: AWCS Procedures

\[\begin{align*}
\text{Agent} & \quad \text{Value} \\
\text{x1} & \quad (0) \\
\text{x2} & \quad (0) \\
\text{x3} & \quad (0) \\
\text{x4} & \quad (0)
\end{align*}\]
Distributed Local Search Approaches

Local Search (LS)

- LS algorithms explore the search space from state to state
- Always tend to improve the current state of the system
- Can naturally handle dynamics (adding constraints, changing values)
- Time efficient
- Not complete and require some subtle parameter tuning

choose an initial assignment $s(0)$

while $s(t)$ not terminal do

    select an acceptable move $m(t)$ to another assignment

    apply move $m(t)$ to reach $s(t + 1)$

    $t := t + 1$

end

Algorithm 4: A generic centralised local search algorithm
Classical Centralised LS Algorithms

Common points

- Initial point (ex: randomly chosen)
- Termination criterion (ex: limit time, $\delta$ improvement)
- Acceptable move (ex: $+\epsilon$)

Famous LS Methods

- **Tabu search** [Glover and Laguna, 1997]
- **Simulated annealing** [Kirkpatrick et al., 1983]
- **Iterative Breakout method** [Morris, 1993]
Distributed Breakout Algorithm (DBA)

wait_ok? mode — (i)
when received (ok?, $x_i$, $d_i$) do
  add ($x_i$, $d_i$) to agent_view;
  when received ok? messages from all neighbors do
    send_improve;
    goto wait_improve mode; end do;
  goto wait_ok mode; end do;

procedure send_improve
  current_eval ← evaluation value of current_value;
  my_improve ← possible maximal improvement;
  new_value ← the value which gives the maximal improvement;
  send (improve, $x_i$, my_improve, current_eval) to neighbors;

wait_improve? mode — (ii)
when received (improve, $x_i$, improve, eval) do
  record this message;
  when received improve? messages from all neighbors do
    send_ok; clear agent_view;
    goto wait_ok mode; end do;
  goto wait_improve mode; end do;

procedure send_ok
  when its improvement is largest among neighbors do
    current_value ← new_value; end do;
  when it is in a quasi-local-minimum do
    increase the weights of constraint violations; end do;
  send (ok?, $x_i$, current_value) to neighbors;

Algorithm 5: DBA Message Handler
Distributed Breakout Algorithm (DBA) (cont.)

Principles of DBA [Yokoo, 2001]

- **Distribution difficulties:**
  1. if two neighbouring agents concurrently change their value, the system may oscillate  
  2. detecting the fact that the whole system is trapped in local minimum requires the agents to globally exchange data

- **DBA answers:**
  1. for a given neighbourhood, only the agent that can maximally improve the evaluation value is given the right to change its value  
  2. agents only detects quasi-local-minimum, which is a weaker local-minimum that can be detected only by local interactions
Remarks

- Distributed version of the iterative breakout algorithm
- Two-mode behaviour alternating between exchange of potential improvement and exchange of assignments
- There is no order over the agents society → neighbourhoods
- The system halts if a solution is found or if the weight of constraints have reached a *predefined upper bound* → the only difficult parameter to set
- DBA is not complete
- DBA is able to detect the termination or a global solution only by reasoning on local data.
Components

- A discrete grid environment, that is used as a communication medium.
- Agents that evolve in some regions of the grid (their domain)
  - Agents move synchronously
  - Agents cannot move in the domain of other agents, but can mark it with the number of potential conflicts
  - These marks represent therefore the number of violated constraints if an agent chooses the marked cell.
- Rules (moves) that agents follow to reach an equilibrium
  - 3 possible actions
    - least-move: the next cell is the one with minimum cost
    - better-move: the next cell is randomly chosen and if it has less conflicts than the actual one the agent moves else the agent rests
    - random-move: the next cell is randomly chosen
  - A decision consists in a random Monte-Carlo choice of the action to perform

\[
 t \leftarrow 0 \\
\text{initialise the grid to 0 violation in each cell; } \text{foreach agent i do} \\
\text{randomly move to a cell of row } i \\
\text{end} \\
\text{while } t < t_{\text{max}} \text{ and no solution do} \\
\text{foreach agent i do} \\
\text{select a move behaviour} \\
\text{compute new position} \\
\text{decrease markers in all cells with past violations} \\
\text{end}
\]
Remarks

- The environment is the communication medium
  - ✔ There is no asynchronous mechanisms and message handling
  - ✗ Synchronisation point: high synchronous solving process with no benefit from distribution, in case of high connected constraint networks

- ERA quickly finds assignments close to the solution → repairing issues

- ✗ Redundant usage of random choices: non-guided method, close to random walk, and non complete

- ✗ Termination: ERA requires a time limit ($t_{max}$) (problem-dependant)
### Panorama

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Type</th>
<th>Memory</th>
<th>Messages</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABT</td>
<td>CSP</td>
<td>Exponential</td>
<td>–</td>
<td>Complete, Static ordering</td>
</tr>
<tr>
<td>AWCS</td>
<td>CSP</td>
<td>Exponential</td>
<td>–</td>
<td>Complete (only with exponential space), Reordering, fast</td>
</tr>
<tr>
<td>DBA</td>
<td>Max-CSP</td>
<td>Linear</td>
<td>Bounded</td>
<td>Incomplete, Fast</td>
</tr>
<tr>
<td>ERA</td>
<td>Max-CSP</td>
<td>Polynomial</td>
<td>n/a</td>
<td>Incomplete, randomness</td>
</tr>
</tbody>
</table>

**Table:** DCSP and DCOP algorithms
Using Distributed Problem Solving

Problem and Environment Characteristics

- Geographic distribution
  - ex: agents are physically distributed, and solving the whole problem is not possible in a centralised manner

- Constraint network topology
  - ex: bounded vertex degrees or large constraint graph diameter

- Knowledge encapsulation
  - ex: privacy preserving, limited knowledge

- Dynamics
  - ex: rather than solving the whole problem again, only repair sub-problems

Some Applications

- Frequency assignment
- Scheduling
- Resource allocation, Manufacturing control
- Supply chain
- ...

Gauthier Picard
Distributed Constraint Processing
References


