Multi-Agent Constraint Problem Solving

Gauthier Picard

ENS Mines Saint-Etienne

gauthier.picard@emse.fr
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
  \[ \Rightarrow \text{distributed problem solving using agents} \]

Approaches

- Classical CSP solver extensions
- Classical local search solver extensions
- Other points of view:
  - Population-based algorithms
  - Cooperation-based solving
Definition (CSP)

A CSP is a triplet \( \langle X, D, C \rangle \) such as:

- \( X = \{x_1, \ldots, x_n\} \) is the set of \emph{variables} to instantiate.
- \( D = \{D_1, \ldots, D_m\} \) is the set of \emph{domains}. Each variable \( x_i \) is related to a domain of value.
- \( C = \{c_1, \ldots, c_k\} \) is the set of \emph{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
Contents

1 Introduction

2 Constraint Satisfaction Problems

3 Multi-Agent Approaches to DisCSP
   - ABT and Extensions
   - Distributed Local Search Approaches

4 Multi-Agent Approaches to DCOP

5 Population-based approaches

6 Cooperation for Problem Solving

7 Synthesis
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
- **Population-based methods**, such as Particle Swarm Optimisation (PSO) or Ant Colony Optimisation (ACO)
- **Self-organising methods** inspired by the previous ones
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) [Yok01]
- 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 \( \phi : X \leftrightarrow A \) is a function assigning variables from \( X \) to agents from \( A \).
Centralised Backtracking

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

Algorithm 1: A classical centralised backtracking search method
Asynchronous Backtracking (ABT)

when received \((\text{ok?}, (x_j, d_j))\) do — (i)
   revise \(\text{agent\_view}\);
   check_agent_view;
end do;

when received \((\text{nogood}, x_j, \text{nogood})\) do — (ii)
   record \(\text{nogood}\) as a new constraint;
   when \(\text{nogood}\) contains an agent \(x_k\) that is not its neighbor
      do request \(x_k\) to add \(x_j\) as a neighbor,
      and add \(x_k\) to its neighbors; end do;
   \(\text{old\_value} \leftarrow \text{current\_value}\); check_agent_view;
when \(\text{old\_value} = \text{current\_value}\) do
   send \((\text{ok?}, (x_j, \text{current\_value}))\) to \(x_j\); end do; end do;

procedure check_agent_view
   when \(\text{agent\_view}\) and \(\text{current\_value}\) are not consistent do
      if no value in \(D_j\) is consistent with \(\text{agent\_view}\) then backtrack;
      else select \(d \in D_j\) where \(\text{agent\_view}\) and \(d\) are consistent;
         \(\text{current\_value} \leftarrow d\);
         send \((\text{ok?}, (x_j, d))\) to neighbors; end if; end do;

procedure backtrack — (iii)
   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;
   select \((x_j, d_j)\) where \(x_j\) has the lowest priority in a nogood;
   send \((\text{nogood}, x_j, V)\) to \(x_j\);
   remove \((x_j, d_j)\) from \(\text{agent\_view}\);
   check_agent_view;

Algorithm 2: ABT Procedures
Asynchronous Backtracking (ABT) (cont.)

**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 [Yok01]
- Satisfaction $\rightarrow$ **Optimisation** with ADOPT (Asynchronous B&B) [MSTY05] or APO [ML06]
- **Adding new agents** at runtime in DynAPO [Mai05]
Asynchronous Weak-Commitment Search (AWCS) [Yok01]

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 \((\text{ok?}, (x_i, d, current\_priority))\) 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 \((\text{ok?}, (x_i, d, current\_priority))\) to neighbors; end do;

Algorithm 3: AWCS Procedures
**Introduction**

CSP  DisCSP  DCOP  Population  Cooperation  Synthesis

ABT et al.  Distributed LS

# 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 $s(0)$

```plaintext
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

<table>
<thead>
<tr>
<th>Common points</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Initial point (ex: randomly chosen)</td>
</tr>
<tr>
<td>- Termination criterion (ex: limit time, $\delta$ improvement)</td>
</tr>
<tr>
<td>- Acceptable move (ex: $+\epsilon$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Famous LS Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Tabu search [GL97]</td>
</tr>
<tr>
<td>- Simulated annealing [KGV83]</td>
</tr>
<tr>
<td>- Iterative Breakout method [Mor93]</td>
</tr>
</tbody>
</table>
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_j, 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
Distribution difficulties:

(i) if two neighbouring agents concurrently change their value, the system may oscillate
(ii) detecting the fact that the whole system is trapped in local minimum requires the agents to globally exchange data

DBA answers:

(i) for a given neighbourhood, only the agent that can maximally improve the evaluation value is given the right to change its value
(ii) 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 evolves 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 represents therefore the number of violated constraints if an agent chooses the marked cell
- **Rules** (*moves*) that agent 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
Environment, Reactive rules and Agents (ERA) [LJT02] (cont.)

Algorithm 6: ERA Outline

\[
t \leftarrow 0
\]
initialise the grid to 0 violation in each cell; \textbf{foreach agent} \textit{i} \textbf{do}
\hspace{1cm} \begin{align*}
    & \text{randomly move to a cell of row } i \\
\end{align*}
\textbf{end}

\textbf{while} \; t < t_{\text{max}} \; \text{and no solution} \; \textbf{do}
\hspace{1cm} \begin{align*}
    & \text{foreach agent} \; i \; \textbf{do} \\
    & \hspace{1cm} \begin{align*}
        & \text{select a move behaviour} \\
        & \text{compute new position} \\
        & \text{decrease markers in all cells with past violations} \\
        & \text{increase markers in all cells with new violations} \\
    \end{align*} \\
\end{align*}
\hspace{1cm} t \leftarrow t + 1
\textbf{end}

\textbf{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 \( \rightarrow \) repairing issues
- ✗ Redundant usage of random choices: non-guided method, close to random walk, and non complete
- ✗ Termination: ERA requires a time limit \( t_{\text{max}} \) (problem-dependant)
Contents

1 Introduction

2 Constraint Satisfaction Problems

3 Multi-Agent Approaches to DisCSP

4 Multi-Agent Approaches to DCOP
   - DCOP Framework
   - Asynchronous Distributed Optimisation (ADOPT)

5 Population-based approaches

6 Cooperation for Problem Solving

7 Synthesis
Motivations

- In dynamic and complex environments not all constraints can be satisfied completely
- Satisfaction → **Optimisation** (combinatorial)
  - ex: minimizing the number of unchecked constraints, minimizing the sum of the costs of violated constraints, etc.

Definition (DCOP)

A **DCOP** is a DCSP \( \langle A, X, D, C, \phi \rangle \) with
- a cost function \( f_{ij} : D_i \times D_j \mapsto \mathbb{N} \cup \infty \) for each pair \( x_i, x_j \)
- an objective function \( F : D \mapsto \mathbb{N} \cup \infty \) evaluating an assignment \( \mathcal{A} \) with \( f_{ij}(d_i, d_j) \) for each pair \( x_i, x_j \)
Objective Function

\[ F(\mathcal{A}) = \sum_{x_i, x_j \in X} f_{ij}(d_i, d_j) \text{ where } x_i \leftarrow d_i \text{ and } x_i \leftarrow d_i \text{ in } \mathcal{A} \]

In figure (a):

- \[ F(\{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0)\}) = 4 \]
- \[ F(\{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1)\}) = 0 \]

and

\[ \mathcal{A}^* = \{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1)\} \]
Asynchronous Distributed Optimisation (ADOPT) [MSTY05]

**Algorithm 7: ADOPT Procedures**

**Initialize:**
\[ Currentvw \leftarrow \{\}; d_i \leftarrow \text{null}; \]
\[ \forall d \in D_i : c(d) \leftarrow 0 \]
\[ \text{hill\_climb;} \]

**when received** (VALUE, \((x_i, d_i)\))
\[ \text{add } (x_i, d_i) \text{ to } Currentvw; \]
\[ \# \text{ context change} \]
\[ \text{if } Currentvw \text{ changed then} \]
\[ \forall d \in D_i : c(d) \leftarrow 0 \]
\[ \text{end if; hill\_climb; } \]

**when received** (VIEW, \(vw, cost\))
\[ d \leftarrow \text{value of } x_i \text{ in } vw \]
\[ \text{if } vw \text{ is compatible with } \]
\[ \forall d \in D_i : \]
\[ c(d) \leftarrow \max(c(d), cost); \]
\[ \text{if } c(d) \text{ changed then} \]
\[ \text{hill\_climb; } \]
\[ \text{end if; } \]

**procedure hill\_climb**
\[ \forall d \in D_i : \]
\[ \# e(d) \text{ is } x_i \text{'s estimate of cost if it chooses } d \]
\[ e(d) \leftarrow \delta(x_i, Currentvw \cup \{(x_i, d)\}) + c(d); \]
\[ \text{choose } d \text{ that minimizes } e(d); \]
\[ \text{prefer current value } d_i \text{ for tie; } \]
\[ d_i \leftarrow d; \]
\[ \text{SEND (VALUE, } (x_i, d_i)\text{) to all linked descendents; } \]
\[ \text{SEND (VIEW, } Currentvw, e(d_i)\text{) to } \]
\[ \text{parent; } \]

\[ \text{Fig. 2. Procedure for asynchronous search (Simple-Adopt) } \]

---

**Introduction CSP DisCSP DCOP Population Cooperation Synthesis Framework ADOPT**

Asynchronous Distributed Optimisation (ADOPT) [MSTY05]
Remarks

- Directly inspired by ABT and branch-and-bound (B&B)
- ADOPT utilises local views and local evaluation $\delta$ to calculate bounds
- ADOPT uses a total order
Contents

1 Introduction

2 Constraint Satisfaction Problems

3 Multi-Agent Approaches to DisCSP

4 Multi-Agent Approaches to DCOP

5 Population-based approaches

6 Cooperation for Problem Solving

7 Synthesis
Population-based approaches

Principles

- A population is a set of individuals (agents)
- Each agent is able to find a solution to the problem
  - an agent knows the whole set of variables that define the problem (contrary to aforementioned approaches)
- Agents coordinate to find a solution

⇒ How to coordinate several concurrent searches to efficiently find a good solution?

Examples

- Evolutionary algorithms, genetic algorithms (GA) [Hol93]
- Particle Swarm Optimisation (PSO)[KE95]
- Ant Colony Optimisation (ACO) [DS04]
We will not see these approaches in this course...

Look at [Hol93, KE95, DS04] for more information!
1 Introduction

2 Constraint Satisfaction Problems

3 Multi-Agent Approaches to DisCSP

4 Multi-Agent Approaches to DCOP

5 Population-based approaches

6 Cooperation for Problem Solving

7 Synthesis
Cooperation: the Engine of Distributed Problem Solving

- The **nogoods** (conflictual configurations) and potential solutions communicated by agents to their neighbourhood in ABT or AWCS help agents to cooperatively solve a DisCSP.

- The **min-conflict heuristic** used in AWCS or ERA is a means to represent the fact that agents cooperatively act by minimising the negative impact of their actions.

- The **pheromone** deposited by ants in ACO gives relevant information about the region of the search space and modifies later the behaviour of the other ants.

- Particles in PSO are influenced by the velocity and position of the local and global bests: cooperative information exchange allowing efficient exploration phase.

- The **fitness function** of GA determines at a time the better individuals which will share their genes with other members of the population to produce new relevant offspring.
Towards a Generic Cooperation-based Method (AMAS) [PG06]

- Cooperation can be viewed as a generic concept manipulated by problem solvers
- It transcends to all the aforementioned methods
- Taking inspiration from biological and socio-economic notions of cooperation
- An agent is unable to find alone the global solution and consequently it has to interact locally with its neighbours in order to find its current actions able to reach its individual goals and help its neighbours

⇒ Cooperation-based algorithms
Cooperation in Problem Solving

- Agents cooperate to find a global solution
  - ex: Agents assign values to variables as to find a global assignment without knowing the global state of the system
- Agents that have difficulties (critical level $\kappa$) to find a local solution are top-priority
  - ex: an agent that do not find a “good” value for a long time can choose its next value before the other agents in its neighbourhood ($\nu$)
- Agents acts cooperatively
  - ex: minimizing conflict, minimizing negative impact, etc.

```plaintext
foreach agent $i$ do
  set an initial assignment to $x_i$
  $x_i.\kappa \leftarrow 0$
end

while termination conditions not met do
  concurrently
  order the possible solutions according to their $\kappa$ value
  $x_{worst} \leftarrow \arg\max \{x_j.\kappa \mid x_j \in \nu(x_i)\}$
  assign cooperative value to $x_i$ such as $x_{worst}.\kappa$ decreases
  compute $x_i.\kappa$
  send $x_i.\kappa$ to neighbours in $\nu(x_i)$
end
```

**Algorithm 8**: An algorithm outline based on critical level ($\kappa$) in AMAS
Example of Cooperative Solving

Cooperative Solving of $n$-queens and $n^2/2$-knights problems [PG06]
1 Introduction
2 Constraint Satisfaction Problems
3 Multi-Agent Approaches to DisCSP
4 Multi-Agent Approaches to DCOP
5 Population-based approaches
6 Cooperation for Problem Solving
7 Synthesis
**Synthesis**

**Dimensions**

*Problem distribution:* the manner and the degree of distribution of the problem among agents (ex: one variable per agent)

*Decision decentralisation:* decentralisation means that no agent has the power and the capability to decide for the others, or to solve the whole part of the problem, at a given time

*Bounded rationality and local actions:* an agent cannot know the values of other agents, and their actions are limited to their own limited neighbourhood

*Robustness to dynamics:* the impact of environmental disturbances occurring at run-time is minimised by the peer-to-peer propagation

*Non determinism:* agents behave or are initially set non deterministically

*Global state is unknown:* micro-level entities are not conscious of the global state of the system, which is however evaluable at macro-level
**Table:** Analysis for all the overviewed methods

<table>
<thead>
<tr>
<th></th>
<th>ABT et al.</th>
<th>DBA</th>
<th>ERA</th>
<th>ACO</th>
<th>PSO</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Distribution</strong></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>Decentralisation</strong></td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Locality</strong></td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Dynamics</strong></td>
<td>limited</td>
<td>limited</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>Non determinism</strong></td>
<td>execution</td>
<td>execution</td>
<td>behaviours</td>
<td>behaviours</td>
<td>behaviours</td>
<td>mechanisms</td>
</tr>
<tr>
<td><strong>Global state</strong></td>
<td>known</td>
<td>unknown</td>
<td>unknown</td>
<td>known</td>
<td>known</td>
<td>known</td>
</tr>
</tbody>
</table>
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
- ...

Introduction  CSP  DisCSP  DCOP  Population  Cooperation  Synthesis
References


References (cont.)


