Distributed Constraint Satisfaction Problems: 2 Asynchronous Algorithms

Thomas Léauté

16.412J - Cognitive Robotics
April 7, 2004

Presentation Outline

1. Introduction to CSPs and DCSPs
2. The Asynchronous Backtracking Algorithm
3. The Asynchronous Weak-Commitment Search Algorithm
4. Conclusion and Introduction to the Task Allocation Problem

1. Introduction to CSPs and DCSPs

- Formal definition: a Constraint Satisfaction Problem (CSP) is a triple \((X, D, C)\), where
  - \(X\) is a list of variables \(x_1, x_2, \ldots, x_n\)
  - \(D\) is a list of finite, discrete value domains \(D_1, D_2, \ldots, D_n\) assigned to the variables
  - \(C\) is a set of constraints \(C_1, C_2, \ldots, C_n\) on the variables, a constraint being a predicate:

\[
C_k : D_{k_1} \times D_{k_2} \times \ldots \times D_{k_j} \rightarrow \{true, false\}
\]

- A solution to the problem is an assignment to the variables that satisfies all the constraints
1. Introduction to CSPs and DCSPs

- Many AI problems can be formulated as CSPs
- Example of a multi-agent scheduling problem*:

Agent A
- Activity A1
- Activity A2
- Activity A3
- Activity A4
- Activity A5

Agent B
- Activity B1
- Activity B2

Shared resources
- R1
- R2
- R3

1. Introduction to CSPs and DCSPs

- Centralized method: one leader agent gathers all the information from other agents and solves the problem
  - Prohibitive cost of collecting information
  - Security/Privacy reasons
  - Not computationally efficient

- Synchronous Backtracking method:

  - Sequential => not computationally efficient
2. The Asynchronous Backtracking Algorithm

- **Assumptions:**
  - Given priority order on the agents
  - An agent must be able to send messages to any other agent
  - Each agent has exactly ONE single variable

- **Key ideas:**
  - Agents work concurrently (= “asynchronously”), exchanging messages to collect required information
  - Conflict-directed search
2. Asynchronous Backtracking

The agent selects a value for its variable satisfying the constraints whose enforcement it is responsible for.

If there is at least one value satisfying the constraints, the agent picks one and changes the assignment to its variable.
2. Asynchronous Backtracking

Possible

Try to choose value

Change value

Send OK? messages

The agent communicates its new assignment to its children through “OK?” messages

Wait

2. Asynchronous Backtracking

Possible

Try to choose value

Change value

Send OK? messages

The agent then waits for answers to his OK? messages from its children (and for other OK? messages from its parents)
If the agent receives an OK? message from one of its parents, it updates its “view”, i.e. its knowledge of the values of the other variables.

The agent then checks its view, making sure that the new values do not violate any constraint it is responsible for.
2. Asynchronous Backtracking

If all the constraints it is responsible for are still satisfied, the agent keeps waiting for messages.

If the agent detects violated constraints, it tries to change the value of its variable to resolve all the violations.
If the agent finds no satisfying value for its variable, it extracts and records a list of conflicts (a conflict being a partial assignment violating at least one constraint).

If one of the conflicts is the empty set, this means any overset of {} is a conflict, so that there is no solution to the DSCP. The agent broadcasts a NO_SOLUTION message and terminates.
Otherwise, for each new conflict, the agent sends a BACKTRACK message describing the conflict to the lowest priority agent whose variable is involved in the conflict. Then it waits for this agent to send back the new assignment to its variable.
2. Asynchronous Backtracking

- Try to choose value
  - possible
    - Change value
    - Send OK? messages
  - impossible
    - Extract & record conflicts
      - yes
        - Send BACKTRACK messages
      - no
        - Send BACKTRACK messages

- Broadcast
  - NO_SOLUTION
    - yes
      - Send BACKTRACK messages
    - no
      - Send BACKTRACK messages

- OK? message
  - Wait
  - BACKTRACK message
    - match?
      - no
        - Send BACKTRACK messages
      - yes
        - Record new constraint
          - It requests a new link if necessary
            - NEW_LINK
              - yes
                - need link?
                  - yes
                    - Update view
                    - Wait
                  - no
                    - Update view
                    - Wait
              - no
                - Update view
                - Wait
          - no
            - Send BACKTRACK messages

Otherwise, it records the conflict as a new constraint it must enforce

- Update view
  - good
    - check view
    - violation!
2. Asynchronous Backtracking

When an agent receives a NEW_LINK request, it adds the sender to its children list and responds through an OK? message.

If the agent receives a NO_SOLUTION message, it terminates.
2. Asynchronous Backtracking: The Graph Coloring Example

Constraints:
\[ x_4 \neq x_3, \ x_4 \neq x_2, \ x_4 \neq x_1 \]

Constraints:
\[ x_2 \neq x_1 \]
2. Asynchronous Backtracking: The Graph Coloring Example

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DOMAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW</td>
<td>CHILDREN</td>
<td>PARENTS</td>
</tr>
<tr>
<td></td>
<td>KNOWN CONFLICTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONSTRAINTS TO ENFORCE</td>
<td></td>
</tr>
</tbody>
</table>

Each agent chooses an assignment to its variable
Each agent sends OK? messages to its children.

Agent $x_2$ and Agent $x_4$ update their view.
2. Asynchronous Backtracking: The Graph Coloring Example

Agent $x_2$ and Agent $x_4$ check their view against their constraints, and Agent $x_4$ discovers a violation.

Agent $x_4$ tries to change its assignment, which is impossible.
2. Asynchronous Backtracking: The Graph Coloring Example

Conflicts:
\{x_1=\text{B}, x_2=\text{G}, x_3=\text{B}\}

Constraints:
\(x_4 \neq x_3, x_4 \neq x_2, x_4 \neq x_1\)

View:
\(x_1=\text{B}, x_2=\text{G}, x_3=\text{B}\)

Agent \(x_4\) extracts and records the conflicts

\[
\text{Extract & record conflicts}
\]

2. Asynchronous Backtracking: The Graph Coloring Example

Conflicts:
\{x_1=\text{B}, x_2=\text{G}, x_3=\text{B}\}

Constraints:
\(x_4 \neq x_3, x_4 \neq x_2, x_4 \neq x_1\)

View:
\(x_1=\text{B}, x_2=\text{G}, x_3=\text{B}\)

\[
\text{Send BACKTRACK messages (BACKTRACK, } \{x_1=\text{B}, x_2=\text{G}, x_3=\text{B}\}\text{)}
\]

\{\} \text{ is not among the new conflicts, so}

Agent \(x_4\) sends BACKTRACK messages
Agent $x_3$ receives the message and checks the conflict against its view.

Agent $x_3$ records the conflict as a new constraint.

View: 
\{
\}

Constraints: 
\{x_1 \neq B \text{ or } x_2 \neq G \text{ or } x_3 \neq B\}
2. Asynchronous Backtracking: The Graph Coloring Example

Agent $x_3$ checks if it needs new links to enforce it

Constraints:
\{x_1 \neq B \text{ or } x_2 \neq G \text{ or } x_3 \neq B\}

Agent $x_3$ sends NEW_LINK requests
Agents $x_1$ and $x_2$ receive the NEW_LINK requests and add Agent $x_3$ to their children list.

Agents $x_1$ and $x_2$ send OK? to confirm new links.
Agent $x_3$ checks its view, and discovers that one constraint (the new one) is violated.
Agent $x_3$ tries to change its value to R

There is no more violation, so Agent $x_3$ communicates its new value to its children
Agent $x_4$ receives the OK? message and updates its view.

Agent $x_4$ checks its view against its constraints and sees the violation has not been resolved...
Agent $x_4$ tries to change its value, but this is impossible.

Agent $x_4$ extracts and records the conflicts.
{} is not among the new conflicts, so Agent $x_4$ sends BACKTRACK messages

Agent $x_3$ receives the message and checks the conflict against its view
2. Asynchronous Backtracking: The Graph Coloring Example

(BACKTRACK, 
{\{x_1=B, x_2=G, x_3=R\}})

View: 
\[x_1=B, x_2=G\]

Constraints: 
\{x_1\neq B \text{ or } x_2\neq G \text{ or } x_3\neq B\}
\{x_1\neq B \text{ or } x_2\neq G \text{ or } x_3\neq R\}

Agent x_3 records the conflict as a new constraint

Agent x_3 needs no new link to enforce it
Agent $x_3$ checks its view, and discovers that one constraint (the new one) is violated.

Agent $x_3$ tries to change its value, but no value satisfies all the constraints.
2. Asynchronous Backtracking: The Graph Coloring Example

Agent $x_3$ extracts and records new conflicts

{} is not a new conflict, so Agent $x_3$ sends BACKTRACK messages
2. Asynchronous Backtracking: The Graph Coloring Example

Agent $x_2$ receives the message and checks the conflict against its view.

Constraints:
\[ x_2 \neq x_1 \]
\[ \{x_1 \neq B \text{ or } x_2 \neq G\} \]

Record new constraint: $x_2$ records the conflict as a new constraint.
2. Asynchronous Backtracking: The Graph Coloring Example

Constraints:
\[ x_2 \neq x_1 \]
\[ \{x_1 \neq B \text{ or } x_2 \neq G\} \]

View:
\[ x_1 = B \]

Agent \( x_3 \) checks its view, and discovers that one constraint (the new one) is violated.

Agent \( x_2 \) tries to change its value to \( B \), but it would violate the first constraint.
2. Asynchronous Backtracking: The Graph Coloring Example

Constraints:
\[ x_2 \neq x_1 \]
\[ \{x_1 \neq B \text{ or } x_2 \neq G\} \]

View:
\[ x_1 = B \]

Agent \( x_2 \) tries to change its value to \( R \)

[Diagram showing the graph coloring example]

The constraint is no longer violated, so Agent \( x_2 \) chooses value \( R \) and communicates it to its children.

Send OK? messages
2. Asynchronous Backtracking: The Graph Coloring Example

Agent $x_3$ and Agent $x_4$ receive the messages and update their views.

Constraints:
- $x_4 \neq x_3$, $x_4 \neq x_2$, $x_4 \neq x_1$

View:
- $x_1 = B$, $x_2 = R$, $x_3 = R$

Agent $x_3$ and Agent $x_4$ check their view against their constraints, and no violation is discovered.

SOLVED!
Weaknesses of the Asynchronous Backtracking Algorithm

• How to better choose the assignments?
  → Use a heuristic to make better choices

• The authors prove the algorithm always reaches a stable state within a finite number of steps, BUT it still lacks a termination procedure
  → Use a “Distributed Snapshot” external procedure


Weaknesses of the Asynchronous Backtracking Algorithm (cont.)

• Need of a judicious priority ordering among the agents
  → Do it beforehand? (might be difficult + need of a centralizing agent…)
  → Dynamic priority ordering: let the agents come up with a judicious ordering themselves, as they encounter conflicts
Weaknesses of the Asynchronous Backtracking Algorithm (cont.)

• How to extract conflicts?
  → Open to all conflict extraction policies
  → There is a trade off between taking the time to extract minimal conflicts, and trying to speed up the algorithm by using the agent’s view as a super-conflict but wasting time by backtracking more often

3. The Asynchronous Weak-Commitment Search Algorithm
3. The Asynchronous Weak-Commitment Search Algorithm

- Use a local min-conflict heuristic to guide the choices of assignments
- Judiciously change the priority ordering every time the search needs to backtrack

3. The Asynchronous Weak-Commitment Search Algorithm (cont.)

- “Weak-Commitment” Search:
  - A partial assignment to the variables is constructed step by step by extending it to variables with lower priority
  - The group of agents “weakly commits” itself to the partial assignment because the partial assignment is abandoned as soon as the algorithm needs to backtrack
  - The priority ordering is then modified so that the agent which failed to find a value to its variable consistent with the constraints “promotes itself” (i.e. it changes its priority value to locally become the agent with the highest priority)
3. The Asynchronous Weak-Commitment Search Algorithm (cont.)

- **ATTENTION! Tricky point:**
  - Every time an agent discovers a known conflict in its view, it will abandon the partial solution.
  - However, if, due to message delays, the agent’s view is obsolete, it will abandon the partial assignment too early and perform an unnecessary change in its priority value.
  - To avoid reacting to such unstable situations, the agent records the conflicts it has already sent, and it will temporarily ignore a conflict if it has already sent it before.

### Flowchart

**3. Weak-Commitment Search**

- **Try to choose value**
  - **Possible**
    - **Change value**
    - **Send OK? messages**
  - **Impossible**
    - **Extract & record conflicts**
    - **Wait BACKTRACK message**
      - **Yes**
        - **Match?**
          - **Yes**
            - **Record new constraint**
            - **Terminate**
          - **No**
            - **Broadcast NO_SOLUTION and terminate**
        - **No**
          - **Send BACKTRACK messages**
    - **Send priority along with the assignment**
    - **Send the message to the parents too**
- **Update view**
  - **Wait**
  - **Good check view violation!**

3. Weak-Commitment Search

- Try to choose value
  - possible: Change value
  - impossible: Extract & record conflicts

- Send OK? messages
  - OK? message
    - Wait
      - BACKTRACK message
        - yes: record new constraint
        - no: send BACKTRACK messages

- Include the priority of the neighbors in the agent’s view

- Update view
  - Wait
    - check view
      - good
      - violation!

3. Weak-Commitment Search

- Try to choose value
  - possible: Change value
  - impossible: Extract & record conflicts

- Send OK? messages
  - OK? message
    - Wait
      - BACKTRACK message
        - yes: record new constraint
        - no: send BACKTRACK messages

- Only backtrack if the conflict has never been sent before
3. Weak-Commitment Search

Guide the choice by minimizing the number of constraint violations with lower priority agents

NOTE: agents must now be aware of ALL constraints involving their variable (OMISSION?!)
3. Weak-Commitment Search

Every time a new constraint is created, inform all involved agents through NEW_CONST messages.
3. Weak-Commitment Search

Upon reception of a NEW_CONST message, record new constraint and new neighbor (if a new link is needed)

Every time a new constraint is created, inform all involved agents through NEW_CONST messages

Wait NEW_CONST messages

check

impossible

Broadcast

and terminate

NO_SOLUTION

yes

no

Send OK? messages

Try to choose value

possible

impossible

Extract & record conflicts

Send OK? messages

Record constraint and neighbor

Send OK?

Update view

Wait

Send NEW_CONST messages

NO_SOLUTION

no

yes

Send BACKTRACK IF NEW

Change value

Promotion

OK? message

Wait

BACKTRACK message

match?

Record constraint and neighbor

Send OK?

Update view

Wait

Send NEW_CONST messages

NO_SOLUTION

Record new constraint

Terminate

good

check view

violation!
3. Weak-Commitment Search: The Graph Coloring Example

Constraints:
\[ x_1 \neq x_2, x_1 \neq x_4 \]

Constraints:
\[ x_2 \neq x_1, x_2 \neq x_4 \]

Constraints:
\[ x_4 \neq x_3, x_4 \neq x_2, x_4 \neq x_1 \]

Constraints:
\[ x_3 \neq x_4 \]
Initial priority values are all set to 0. Two agents with identical priorities are ordered with respect to their index.

Each agent chooses an assignment to its variable (at the first time step, we cannot use the heuristic because agents still have empty views).
3. Weak-Commitment Search: The Graph Coloring Example

Each agent sends OK? messages to ALL of its neighbors

All agents update their view
3. Weak-Commitment Search: The Graph Coloring Example

Constraints:

\[ x_1 \neq x_2, x_1 \neq x_3, x_1 \neq x_4 \]

View:

\[ x_1 = (B, 0), x_4 = (G, 0) \]

Constraints:

\[ x_2 \neq x_1, x_2 \neq x_4 \]

All agents check their view against the constraints they are responsible for, and Agent \( x_4 \) discovers a violation.

Agent \( x_4 \) tries to change its assignment, which is impossible.
Conflicts: \( \{x_1 = B, x_2 = G, x_3 = B\} \)

View:
\( x_1 = (B, 0), x_2 = (G, 0) \)
\( x_3 = (B, 0) \)

Agent \( x_4 \) extracts and records the conflicts

Conflicts: \( \{x_1 = B, x_2 = G, x_3 = B\} \)

(BACKTRACK, \( \{x_1 = B, x_2 = G, x_3 = B\}\))

{} is not among the new conflicts, and no new conflict has already been sent, so Agent \( x_4 \) sends BACKTRACK messages
3. Weak-Commitment Search: The Graph Coloring Example

Promotion

Agent $x_4$ promotes itself, changing its priority value from 0 to 1

Send OK? messages

Agent $x_4$ communicates its new priority value to ALL its neighbors
CONCURRENTLY, Agent $x_3$ receives the BACKTRACK message and checks the conflict against its view.

Agent $x_3$ records the conflict as a new constraint it will be responsible for.

Constraints:

- $x_1 \neq B$ or $x_2 \neq G$ or $x_3 \neq B$
- $x_3 \neq x_4$

Record new constraint
CONCURRENTLY, all agents receive the OK? messages and update their view.

(Only the priority changed, so no new violation is discovered)
Concurrently, Agent $x_3$ sends NEW_CONST messages to all agents involved in the new constraint.

Agents $x_1$ and $x_2$ receive NEW_CONST messages and record new constraint and new neighbor.

Constraints:

$\begin{align*}
&x_1 \neq x_2, \ x_1 \neq x_4 \\
&\{x_1 \neq B \text{ or } x_2 \neq G \text{ or } x_3 \neq B\}
\end{align*}$
Agents $x_1$ and $x_2$ respond to the NEW_CONST through OK? messages.

Agent $x_3$ receives messages and updates its view.

View:
$x_1=(B, 0), x_2=(G, 0), x_3=(G, 1)$
3. Weak-Commitment Search: The Graph Coloring Example

Agent $x_3$ checks its view, and discovers that one constraint it is responsible for (the new one) is violated.

Agent $x_3$ tries to change its value to R, deleting all violations of constraints it is responsible for, and minimizing the number of violations of others.
There is no more violations of constraints it is responsible for, so Agent $x_3$ communicates its new value to ALL neighbors

All agents receive the OK? messages and update their view
Agent $x_1$, $x_2$ and $x_4$ check their view against the constraints they are responsible for, and $x_2$ discovers a violation.

Try to choose value

Agents $x_2$ tries to change its value to R, deleting all violations of constraints it is responsible for, and minimizing the number of violations of others.
3. Weak-Commitment Search: The Graph Coloring Example

There is no more violations of constraints it is responsible for, so Agent $x_3$ communicates its new value to ALL neighbors.

All agents receive the OK? messages and update their view.
3. Weak-Commitment Search: The Graph Coloring Example

Agents $x_1$, $x_3$ and $x_4$ check their view against the constraints they are responsible for, and no new violation is discovered.

4. Conclusion

- Perform much better than the trivial algorithms
4. Conclusion

M. Yokoo, E. Durfee, T. Ishida and K. Kuwabara, *Distributed Constraint Satisfaction Problem: Formalization and Algorithms*, IEEE Transactions on Knowledge and Data Engineering, VOL. 10, NO. 5, Sept/Oct 1998. Fig. 7.

4. Conclusion

4. Conclusion

• Perform much better than the trivial algorithms
• Single-variable agents => Task Allocation Problem

W.-M. Shen and B. Salemi, *Distributed and Dynamic Task Reallocation in Robot Organizations*

---

References


• W.-M. Shen and B. Salemi, *Distributed and Dynamic Task Reallocation in Robot Organizations*