Distributed Constraints

José M Vidal

Department of Computer Science and Engineering, University of South Carolina

January 15, 2010

Abstract

Outline
Graph Coloring
Graph Coloring
Constraint Satisfaction Problem (CSP)

Given variables $x_1, x_2, \ldots x_n$ with domains $D_1, D_2, \ldots D_n$ and a set of boolean constraints $P$ of the form $p_k(x_{k1}, x_{k2}, \ldots, x_{kj}) \rightarrow \{0, 1\}$, find assignments for all the variables such that no constraints are violated.
Depth First Search for the CSP

DEPTH-FIRST-SEARCH-CSP\((i, g)\)

1. if \(i > n\) then return \(g\)
2. for \(v \in D_i\)
3. do if setting \(x_i \leftarrow v\) does not violate any constraint in \(P\) given \(g\)
4. then \(g' \leftarrow\) DEPTH-FIRST-SEARCH-CSP\((i + 1, g + \{x_i \leftarrow v\})\)
5. if \(g' \neq \emptyset\) then return \(g'\)
6. return \(\emptyset\)
Definition (Distributed Constraint Satisfaction Problem (DCSP))

Give each agent one of the variables in a CSP. Agents are responsible for finding a value for their variable and can find out the values of their neighbors’ via communication.
Outline
REVISE($x_i$, $x_j$)

1. $old\text{-}domain \leftarrow D_i$
2. for $v_i \in D_i$
3. do if there is no $v_j \in D_j$ consistent with $v_i$
4. then $D_i \leftarrow D_i - v_i$
5. if $old\text{-}domain \neq D_i$
6. then $\forall k \in \{\text{neighbors of } i\} \ \text{HANDLE-NEW-DOMAIN}(i, D_i)$
Distributed Constraints
Constraint Satisfaction
Filtering Algorithm
Distributed Constraints
Constraint Satisfaction
Filtering Algorithm

\[ \text{REVISE}(x_1, x_3) \]
Distributed Constraints
Constraint Satisfaction
Filtering Algorithm

\[ \text{REVISE}(x_1, x_3) \]
Distributed Constraints
Constraint Satisfaction
Filtering Algorithm

\textbf{Revise}(x_1, x_3)

Communicate

\textbf{Diagram:}

- Variables: x_1, x_2, x_3
- Constraints represented by x's in the grid
- Communication between variables

\textbf{Graph:}

- Nodes: x_1, x_2, x_3
- Edges connecting x_1 to x_2 and x_3

\textbf{Explanation:}

The diagram illustrates a constraint satisfaction problem where variables x_1, x_2, and x_3 are connected, and constraints are represented by x's in the grid. The process of \textbf{Revise} is shown as a step in the filtering algorithm, updating the constraints and communicating changes among the variables.
Distributed Constraints

Constraint Satisfaction

Filtering Algorithm
Filtering fails to detect no-solution.
Filtering fails to detect no-solution.
Outline
Definition (k-consistency)

Given any instantiation of any \( k - 1 \) variables that satisfy all constraints it is possible to find an instantiation of any \( k^{th} \) variable such that all \( k \) variable values satisfy all constraints.
Definition (Strongly $k$-consistent)

A problem is strongly $k$-consistent if it is $j$-consistent for all $j \leq k$. 
Definition (Hyper-Resolution Rule)

\[ A_1 \lor A_2 \lor \cdots \lor A_m \]
\[ \neg (A_1 \land A_{11} \land \cdots ) \]
\[ \neg (A_2 \land A_{21} \land \cdots ) \]
\[ \vdots \]
\[ \neg (A_m \land A_{m1} \land \cdots ) \]
\[ \frac{\neg (A_{11} \land \cdots \land A_{21} \land \cdots \land A_{m1} \land \cdots )}{\neg (A_{11} \land \cdots \land A_{21} \land \cdots \land A_{m1} \land \cdots )}. \]
Distributed Constraints
Constraint Satisfaction
Hyper-Resolution Based Consistency Algorithm

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 = \bullet \lor x_1 = \bullet )</td>
<td>( x_2 = \bullet \lor x_2 = \bullet )</td>
<td>( x_3 = \bullet \lor x_3 = \bullet )</td>
</tr>
<tr>
<td>( \neg (x_1 = \bullet \land x_2 = \bullet) )</td>
<td>( \neg (x_1 = \bullet \land x_2 = \bullet) )</td>
<td>( \neg (x_1 = \bullet \land x_2 = \bullet) )</td>
</tr>
<tr>
<td>( \neg (x_1 = \bullet \land x_2 = \bullet) )</td>
<td>( \neg (x_1 = \bullet \land x_2 = \bullet) )</td>
<td>( \neg (x_1 = \bullet \land x_2 = \bullet) )</td>
</tr>
</tbody>
</table>

Diagram:

```
  X1
 / \ \
/   \ \
X2   X3
```

Nodes: \( \bullet \) for true, \( \circ \) for false.
Distributed Constraints
Constraint Satisfaction
Hyper-Resolution Based Consistency Algorithm

\[
\begin{array}{ccc}
\mathbf{X}_1 & \mathbf{X}_2 & \mathbf{X}_3 \\
\mathbf{X}_1 = \bullet \lor \mathbf{X}_1 = \bullet & \mathbf{X}_2 = \bullet \lor \mathbf{X}_2 = \bullet & \mathbf{X}_3 = \bullet \lor \mathbf{X}_3 = \bullet \\
\neg (\mathbf{X}_1 = \bullet \land \mathbf{X}_2 = \bullet) & \neg (\mathbf{X}_1 = \bullet \land \mathbf{X}_2 = \bullet) & \neg (\mathbf{X}_1 = \bullet \land \mathbf{X}_2 = \bullet) \\
\neg (\mathbf{X}_1 = \bullet \land \mathbf{X}_2 = \bullet) & \neg (\mathbf{X}_1 = \bullet \land \mathbf{X}_2 = \bullet) & \neg (\mathbf{X}_1 = \bullet \land \mathbf{X}_2 = \bullet) \\
\neg (\mathbf{X}_2 = \bullet \land \mathbf{X}_3 = \bullet)
\end{array}
\]
Distributed Constraints
Constraint Satisfaction
Hyper-Resolution Based Consistency Algorithm

### Table

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = \bullet \lor x_1 = \bullet$</td>
<td>$x_2 = \bullet \lor x_2 = \bullet$</td>
<td>$x_3 = \bullet \lor x_3 = \bullet$</td>
</tr>
<tr>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
</tbody>
</table>

### Diagram

- **$x_1$** Sends $\neg (x_2 = \bullet \land x_3 = \bullet)$ to $x_2$ and $x_3$. 
Distributed Constraints
Constraint Satisfaction
Hyper-Resolution Based Consistency Algorithm

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = \bullet \lor x_1 = \bullet$</td>
<td>$x_2 = \bullet \lor x_2 = \bullet$</td>
<td>$x_3 = \bullet \lor x_3 = \bullet$</td>
</tr>
<tr>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_3 = \bullet)$</td>
<td>$\neg (x_3 = \bullet)$</td>
<td>$\neg (x_3 = \bullet)$</td>
</tr>
</tbody>
</table>

*Image of a triangle with constraints:*

- $x_1 = \bullet \lor x_1 = \bullet$
- $\neg (x_1 = \bullet \land x_2 = \bullet)$
- $\neg (x_1 = \bullet \land x_2 = \bullet)$
- $\neg (x_1 = \bullet \land x_2 = \bullet)$
- $\neg (x_2 = \bullet \land x_3 = \bullet)$
- $\neg (x_2 = \bullet \land x_3 = \bullet)$
- $\neg (x_2 = \bullet \land x_3 = \bullet)$
- $\neg (x_3 = \bullet)$
<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = \textbullet \lor x_1 = \textbullet$</td>
<td>$x_2 = \textbullet \lor x_2 = \textbullet$</td>
<td>$x_3 = \textbullet \lor x_3 = \textbullet$</td>
</tr>
<tr>
<td>$\neg(x_1 = \textbullet \land x_2 = \textbullet)$</td>
<td>$\neg(x_1 = \textbullet \land x_2 = \textbullet)$</td>
<td>$\neg(x_1 = \textbullet \land x_2 = \textbullet)$</td>
</tr>
<tr>
<td>$\neg(x_1 = \textbullet \land x_2 = \textbullet)$</td>
<td>$\neg(x_1 = \textbullet \land x_2 = \textbullet)$</td>
<td>$\neg(x_1 = \textbullet \land x_2 = \textbullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \textbullet \land x_3 = \textbullet)$</td>
<td>$\neg(x_2 = \textbullet \land x_3 = \textbullet)$</td>
<td>$\neg(x_2 = \textbullet \land x_3 = \textbullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \textbullet \land x_3 = \textbullet)$</td>
<td>$\neg(x_3 = \textbullet)$</td>
<td>$\neg(x_3 = \textbullet)$</td>
</tr>
</tbody>
</table>

**Diagram:**

- $x_1$: $x_1 = \textbullet \lor x_1 = \textbullet$
- $\neg(x_1 = \textbullet \land x_2 = \textbullet)$
- $\neg(x_1 = \textbullet \land x_3 = \textbullet)$
- $\neg(x_2 = \textbullet \land x_3 = \textbullet)$
### Distributed Constraints

**Constraint Satisfaction**

**Hyper-Resolution Based Consistency Algorithm**

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = \bullet \lor x_1 = \bullet$</td>
<td>$x_2 = \bullet \lor x_2 = \bullet$</td>
<td>$x_3 = \bullet \lor x_3 = \bullet$</td>
</tr>
<tr>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg (x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg (x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
</tbody>
</table>

![Diagram](attachment:image.png)

$x_1$ Sends $\neg (x_2 = \bullet \land x_3 = \bullet)$ to $x_2$ and $x_3$. 
### Distributed Constraints

#### Constraint Satisfaction

#### Hyper-Resolution Based Consistency Algorithm

<table>
<thead>
<tr>
<th></th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x_1 = \bullet \lor x_1 = \bullet)</td>
<td>(x_2 = \bullet \lor x_2 = \bullet)</td>
<td>(x_3 = \bullet \lor x_3 = \bullet)</td>
</tr>
<tr>
<td>¬</td>
<td>((x_1 = \bullet \land x_2 = \bullet))</td>
<td>((x_1 = \bullet \land x_2 = \bullet))</td>
<td>((x_1 = \bullet \land x_2 = \bullet))</td>
</tr>
<tr>
<td>¬</td>
<td>((x_1 = \bullet \land x_2 = \bullet))</td>
<td>((x_1 = \bullet \land x_2 = \bullet))</td>
<td>((x_1 = \bullet \land x_2 = \bullet))</td>
</tr>
<tr>
<td>¬</td>
<td>((x_2 = \bullet \land x_3 = \bullet))</td>
<td>((x_2 = \bullet \land x_3 = \bullet))</td>
<td>((x_2 = \bullet \land x_3 = \bullet))</td>
</tr>
<tr>
<td>¬</td>
<td>((x_2 = \bullet \land x_3 = \bullet))</td>
<td>((x_3 = \bullet))</td>
<td>((x_2 = \bullet \land x_3 = \bullet))</td>
</tr>
<tr>
<td>¬</td>
<td>((x_2 = \bullet \land x_3 = \bullet))</td>
<td>((x_3 = \bullet))</td>
<td>((x_2 = \bullet \land x_3 = \bullet))</td>
</tr>
<tr>
<td>¬</td>
<td>((x_3 = \bullet))</td>
<td>((x_3 = \bullet))</td>
<td>((x_3 = \bullet))</td>
</tr>
</tbody>
</table>
Distributed Constraints
Constraint Satisfaction
Hyper-Resolution Based Consistency Algorithm

<table>
<thead>
<tr>
<th>$\mathbf{x}_1$</th>
<th>$\mathbf{x}_2$</th>
<th>$\mathbf{x}_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = \bullet \lor x_1 = \bullet$</td>
<td>$x_2 = \bullet \lor x_2 = \bullet$</td>
<td>$x_3 = \bullet \lor x_3 = \bullet$</td>
</tr>
<tr>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_3 = \bullet)$</td>
</tr>
<tr>
<td></td>
<td>$\neg(x_3 = \bullet)$</td>
<td>$\neg(x_3 = \bullet)$</td>
</tr>
</tbody>
</table>

$x_2$ Sends $\neg(x_3 = \bullet)$ and $\neg(x_3 = \bullet)$ to $x_3$. 
### Hyper-Resolution Based Consistency Algorithm

#### Table

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = \bullet \lor x_1 = \bullet$</td>
<td>$x_2 = \bullet \lor x_2 = \bullet$</td>
<td>$x_3 = \bullet \lor x_3 = \bullet$</td>
</tr>
<tr>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
<td>$\neg(x_1 = \bullet \land x_2 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
<tr>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
<td>$\neg(x_2 = \bullet \land x_3 = \bullet)$</td>
</tr>
</tbody>
</table>

#### Diagram

```
\text{\begin{tikzpicture}
\node (x1) at (0,0) {$x_1$};
\node (x2) at (1,1) {$x_2$};
\node (x3) at (2,0) {$x_3$};
\draw (x1) -- (x2);
\draw (x2) -- (x3);
\draw (x3) -- (x1);
\end{tikzpicture}}
```

**Contradiction**
Backtracking
Backtracking
Backtracking
Backtracking
Backtracking
Agent Variables

- **priority**: the agent’s fixed priority number. All agents are ordered.
- **local-view**: current values of other agents’ variables.
- **current-value**: current value of agent’s variable.
- **neighbors**: initially, the set of agents with whom agent shares a constraint.
HANDLE-OK?$(j, x_j)$ This message asks the receiver if that assignment does not violate any of his constraints.

HANDLE-NOGOOD$(j, nogood)$ which means that $j$ is reporting that it can’t find a value for his variable because of $nogood$.

HANDLE-ADD-NEIGHBOR$(j)$ which requests the agent to add some other agent $j$ as its neighbor.
HANDLE-OK?(j, xj)
1 \textit{local-view} \leftarrow \textit{local-view} + (j, xj)
2 \textsc{check-local-view}()
CHECK-LOCAL-VIEW() 
1 \textbf{if} local-view and } x_i \text{ are not consistent
2 \textbf{then if} no value in } D_i \text{ is consistent with } local-view
3 \textbf{then} BACKTRACK()
4 \textbf{else} select } d \in D_i \text{ consistent with } local-view
5 \quad x_i \leftarrow d
6 \quad \forall k \in \text{neighbors } \text{ } k.\text{HANDLE-OK}(i, x_i)
HANDLE-NOGOOD\((j, nogood)\)
1. record \(nogood\) as a new constraint
2. for \((k, x_k) \in nogood\) where \(k \notin \text{neighbors}\)
3. \hspace{1em} do \(k\).HANDLE-ADD-NEIGHBOR\((i)\)
4. \hspace{1em} neighbors ← neighbors + \(k\)
5. \hspace{1em} local-view ← local-view + \((k, x_k)\)
6. old-value ← \(x_i\)
7. CHECK-LOCAL-VIEW()
8. if \(old-value \neq x_i\)
9. \hspace{1em} then \(j\).HANDLE-OK?\((i, x_i)\)
BACKTRACK()
1  nogoods ← \{V | V = inconsistent subset of local-view using hyper-resolution rule\}
2  if an empty set is an element of nogoods
3     then broadcast that there is no solution
4          terminate this algorithm
5  for V ∈ nogoods
6     do select \((j, x_j)\) where \(j\) has lowest priority in \(V\)
7         \(j\).HANDLE-NOGOOD\((i, V)\)
8         local-view ← local-view \(−(j, x_j)\)
9  CHECK-LOCAL-VIEW()
Example

\[ \text{Added link: } \text{ok}(x_2), \text{local-view} = (x_2), (x_3) \]

\[ \text{nogood}(x_2) \land x_3 = \]
Example

$$x_1 \xrightarrow{\text{ok}} \left(x_2, x_3\right)$$

$$\text{local-view} = \left(x_2, x_3\right)$$

$$\text{nogood} (x_2, x_3)$$
Distributed Constraints
Constraint Satisfaction
Asynchronous Backtracking

Example

\[ \text{local view} = (x_2, x_3) \]
\[ \text{nogood} (x_2 = \land x_3 = \) \]
\[ \text{ok?}(x_2, ) \]
\[ \text{ok?}(x_3, ) \]
Example

\[ local-view = (x_2, \bullet), (x_3, \bullet) \]
Example

\[ \text{local-view} = (x_2, \bullet), (x_3, \bullet) \]
Example

(local-view = (x₂, •), (x₃, •))
Example

\[ \text{local-view} = (x_2, \bullet), (x_3, \bullet) \]

\[ \text{local-view} = (x_2, \bullet) \]
Example

\[
\text{Added link}
\]

\[
\text{local-view} = (x_2, \bullet), (x_3, \bullet)
\]

\[
\text{nogood}(x_2 = \bullet \land \neg x_3 = \bullet)
\]
Theorem (ABT is Complete)

The ABT algorithm always finds a solution if one exists and terminates with the appropriate message if there is no solution.

Proof.

By induction. First show that the agent with the highest priority never enters an infinite loop. Then show that given that all the agents with lower priority that $k$ never fall into an infinite loop then $k$ will not fall into an infinite loop. □
Asynchronous Weak-Commitment (AWC)

- Use dynamic priorities.
- Change **ok?** messages to include agent’s current priority.
- Use min-conflict heuristic.
CHECK-LOCAL-VIEW

1  if $x_i$ is consistent with local-view then return
2  then BACKTRACK()
3  if no value in $D_i$ is consistent with local-view then BACKTRACK()
4  else select $d \in D_i$ consistent with local-view and which minimizes constraint violations with lower priority agents.
5
6  $x_i \leftarrow d$
7  $\forall k \in \text{neighbors} \ k.\text{HANDLE-OK}\left(i, x_i, priority\right)$
BACKTRACK

1. generate a nogood $V$
2. if $V$ is empty nogood
   3. then broadcast that there is no solution
      4. terminate this algorithm
5. if $V$ is a new nogood
   6. then $\forall_{(k,x_k) \in V} k.\text{HANDLE-NOGOOD}(i,j,priority)$
6. $priority \leftarrow 1 + \max\{\text{neighbors' priorities}\}$
8. select $d \in D_i$ consistent with local-view
   and which minimizes constraint violations with lower priority agents.
9. $x_i \leftarrow d$
10. $\forall_{k \in \text{neighbors}} k.\text{HANDLE-OK}?(i,x_i,priority)$
Example

priority = 0

priority = 0

priority = 0

ok(?(x2, , 0))

local-view = (x2, )

nogood(x2, )

priority = 1

Added link

ok(?(x2, , 0))

local-view = (x2, )

nogood(x2, )

priority = 2
Example

\[ \text{priority} = 0 \]

\[ x_1 \]

\[ x_2 \]
\[ \text{priority} = 0 \]

\[ x_3 \]
\[ \text{priority} = 0 \]

\( \text{ok}(x_2, 0) \)

\( \text{ok}(x_3, 0) \)

\( \text{local-view} = (x_2), (x_3) \)

\( \text{nogood}(x_2, 0) \)

\[ \text{priority} = 1 \]

\( \text{ok}(x_2, 0) \)

\( \text{local-view} = (x_2) \)

\( \text{nogood}(x_2) \)

\[ \text{priority} = 2 \]
Example

\[ \text{priority} = 0 \]

\[ \text{OK}(x_2, 0) \]

\[ \text{OK}(x_3, 0) \]

\[ \text{local view} = (x_2, x_3) \]

\[ \text{nogood} (x_2, x_3) \]

\[ \text{priority} = 1 \]

\[ \text{OK}(x_2, 0) \]

\[ \text{local view} = (x_2) \]

\[ \text{nogood} (x_2) \]

\[ \text{priority} = 2 \]
Example

\[ x_1 \]

\[ \text{local-view} = (x_2, \bullet), (x_3, \bullet) \]

\[ priority = 0 \]

\[ x_2 \]

\[ x_3 \]

\[ \text{priority} = 0 \]

\[ \text{priority} = 0 \]
Example

\[ \text{local-view} = (x_2, \bullet), (x_3, \bullet) \]

\[ \text{nogood}(x_2 = \top \land x_3 = \bot), 1 \]

\[ priority = 1 \]

\[ priority = 0 \]

\[ priority = 0 \]
Example

Distributed Constraints
Constraint Satisfaction
Asynchronous Weak-Commitment Search

Example

Priority: $x_1$ (priority = 1)

Local-view: $(x_2, \bullet), (x_3, \bullet)$

Nogood: $(x_2 = \text{false} \land x_3 = \text{false})$

Priority: $x_2$ (priority = 0)

Added link

Priority: $x_3$ (priority = 0)
Example

\[\text{local-view} = (x_2, \bullet), (x_3, \bullet)\]

\[\text{priority} = 1\]

\[\text{ok?}(x_2, \bullet, 0)\]

\[\text{Added link}\]

\[\text{priority} = 0\]
Example

\[ \text{priority} = 1 \]

\[ \text{local-view} = (x_2, \bullet), (x_3, \bullet) \]

\[ \text{nogood}(x_2 = 0 \land x_3 = 0) \]

\[ \text{Added link} \]

\[ \text{local-view} = (x_2, \bullet) \]

\[ \text{priority} = 2 \]

\[ \text{nogood}(x_2, \bullet) \]

\[ \text{priority} = 0 \]
Theorem (AWC is complete)

*The AWC algorithm always finds a solution if one exists and terminates with the appropriate message if there is no solution.*

Proof.

The priority values are changed if and only if a new nogood is found. Since the number of possible nogoods is finite the priority values cannot be changed indefinitely. When the priority values are stable AWC becomes ABT, which is complete. □
Hill Climbing
Hill Climbing
Hill Climbing
Hill Climbing
Hill Climbing
Hill Climbing
Definition (Quasi-local-minimum)

An agent $x_i$ is in a quasi-local-minimum if it is violating some constraint and neither it nor any of its neighbors can make a change that results in lower cost for all.
Remote Procedure Calls

- `HANDLE-OK?(i, x_i)` where $i$ is the agent and $x_i$ is its current value,
- `HANDLE-IMPROVE(i, improve, eval)` where `improve` is the maximum $i$ could gain by changing to some other color and `eval` is its current cost.
HANDLE-OK?(j, x_j)

1  received-ok[j] ← TRUE
2  agent-view ← agent-view + (j, x_j)
3  if \( \forall k \in \text{neighbors} \) received-ok[k] = TRUE
4     then SEND-IMPROVE()
5     \( \forall k \in \text{neighbors} \) received-ok[k] ← FALSE
SEND-IMPROVE()

1. $cost \leftarrow$ evaluation of $x_i$ given current weights and values.
2. $my\text{-}improve \leftarrow$ possible maximal improvement
3. $new\text{-}value \leftarrow$ value that gives maximal improvement
4. $\forall k \in \text{neighbors} \ k \cdot \text{HANDLE\text{-}IMPROVE}(i, my\text{-}improve, cost)$
HANDLE-IMPROVE($j$, improve, eval)

1. $\text{received-improve}[j] \leftarrow \text{improve}$
2. if $\forall_{k \in \text{neighbors}} \text{received-improve}[k] \neq \text{NONE}$
   then $\text{SEND-OK}$
3. $\text{agent-view} \leftarrow \emptyset$
4. $\forall_{k \in \text{neighbors}} \text{received-improve}[k] \leftarrow \text{NONE}$
SEND-OK( )

1 if $\forall k \in \text{neighbors} \ my\text{-}improve \geq received\text{-}improve[k]$
2 then $x_i \leftarrow \text{new\text{-}value}$
3 if $\text{cost} > 0 \land \forall k \in \text{neighbors} \ received\text{-}improve[k] \leq 0 \triangleright \text{quasi-local opt.}$
4 then increase weight of constraint violations
5 $\forall k \in \text{neighbors} \ k.\text{HANDLE-OK}(i, x_i)$
Example
Example
Theorem (Distributed Breakout is not Complete)

Distributed breakout can get stuck in local optima. Therefore, there are cases where a solution exists and it cannot find it.

Proof.

By example.
Theorem (Distributed Breakout is not Complete)

Distributed breakout can get stuck in local optima. Therefore, there are cases where a solution exists and it cannot find it.

Proof.

By example.

In practice, its really good.
Outline
Definition (Constraint Optimization Problem (COP))

Given variables $x_1, x_2, \ldots, x_n$ with domains $D_1, D_2, \ldots, D_n$ and a set of constraints $P$ of the form $pk(x_{k1}, x_{k2}, \ldots, x_{kj}) \to \mathbb{R}$, find assignments for all the variables such that the sum of the constraint values is minimized.
BRANCH-AND-BOUND-COP()

1. $c^* \leftarrow \infty$  \> Minimum cost found. Global variable.
2. $g^* \leftarrow \emptyset$  \> Best solution found. Global variable.
3. BRANCH-AND-BOUND-COP-HELPER(1, $\emptyset$)
4. return $g^*$

BRANCH-AND-BOUND-COP-HELPER($i$, $g$)

1. if $i = n$
2. then if $P(g) < c^*$
3. then $g^* \leftarrow g$
4. $c^* \leftarrow P(g)$
5. return
6. for $v \in D_i$
7. do $g' \leftarrow g + \{x_i \leftarrow v\}$
8. if $P(g') < c^*$
9. then BRANCH-AND-BOUND-COP-HELPER($i + 1$, $g'$)
Definition (Distributed Constraint Optimization Problem (DCOP))

Give each agent one of the variables in a COP. Agents are responsible for finding a value for their variable and can find out the values of their neighbors’ via communication.
Outline
Remote Procedure Calls

- **threshold** tell children how much cost they can incur, ignore anything that costs more than that.
- **value** tell descendants what value agent sets itself to.
- **cost** tell parent lower and upper bounds of cost given the current value assignments of ancestors.
Distributed Constraints

Distributed Constraint Optimization

Adopt

\[
\begin{array}{|c|c|c|}
\hline
 d_i & d_j & p(d_i, d_j) \\
\hline
 0 & 0 & 1 \\
 0 & 1 & 2 \\
 1 & 0 & 2 \\
 1 & 1 & 0 \\
\hline
\end{array}
\]

\[
\begin{array}{c}
x_1 = 0 \\
x_2 = 1 \\
x_3 = 0 \\
x_4 = 1
\end{array}
\]
Distributed Constraints
Distributed Constraint Optimization
Adopt

\[
\begin{array}{|c c| c|}
\hline
d_i & d_j & p(d_i, d_j) \\
\hline
0 & 0 & 1 \\
0 & 1 & 2 \\
1 & 0 & 2 \\
1 & 1 & 0 \\
\hline
\end{array}
\]

\[
x_1 = 0 \\
x_2 = 0 \\
x_3 = 0 \\
x_4 = 0 
\]
Distributed Constraint Optimization

\[
\begin{array}{c|c|c}
 d_i & d_j & p(d_i, d_j) \\
 0 & 0 & 1 \\
 0 & 1 & 2 \\
 1 & 0 & 2 \\
 1 & 1 & 0 \\
\end{array}
\]

\[x_1 = 0\]
\[x_2 = 0\]
\[x_3 = 0\]
\[x_4 = 0\]

Value
\[x_1 = 0\]
\[x_2 = 0\]
Distributed Constraints

Distributed Constraint Optimization

Adopt

<table>
<thead>
<tr>
<th>$d_i$</th>
<th>$d_j$</th>
<th>$p(d_i, d_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$x_1 = 0$

$x_2 = 0$

$x_3 = 0$

$x_4 = 0$

$\text{cost } 1,\infty$

$x_1 = 0$

$\text{cost } 2,2$

$x_1 = 0$

$x_2 = 0$

$\text{cost } 1,1$

$x_2 = 0$
Distributed Constraints

Distributed Constraint Optimization

Adopt

<table>
<thead>
<tr>
<th>$d_i$</th>
<th>$d_j$</th>
<th>$p(d_i, d_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ x_1 = 1 \]

\[ x_2 = 0 \]

\[ x_3 = 0 \]

\[ x_4 = 0 \]
### Distributed Constraints

**Distributed Constraint Optimization**

**Adopt**

<table>
<thead>
<tr>
<th>$d_i$</th>
<th>$d_j$</th>
<th>$p(d_i, d_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{cost} & : 0, \infty \quad x_1 = 1 \\
\text{value} & : x_2 = 0 \quad x_1 = 1 \\
& : x_2 = 1 \\
\end{align*}
\]
Distributed Constraint Optimization

\[
\begin{array}{c|c|c}
   d_i & d_j & p(d_i, d_j) \\
\hline
   0 & 0 & 1 \\
   0 & 1 & 2 \\
   1 & 0 & 2 \\
   1 & 1 & 0 \\
\end{array}
\]

\[
x_1 = 1 \\
x_2 = 1 \\
x_3 = 1 \\
x_4 = 1 \\
\]

\[
\text{cost} \\
0,0 \\
x_1 = 1 \\
\text{cost} \\
0,0 \\
x_2 = 1 \\
\text{cost} \\
0,3 \\
x_1 = 1
\]
Distributed Constraints

Distributed Constraint Optimization

Adopt

<table>
<thead>
<tr>
<th>$d_i$</th>
<th>$d_j$</th>
<th>$p(d_i, d_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$x_1 = 1$

$x_2 = 1$

$x_3 = 1$

$x_4 = 1$

cost

0, 0

$x_1 = 1$
Outline
Theorem (APO worst case is centralized search)

*In the worst case APO (OptAPO) will make one agent do a completely centralized search of the complete problem space.*

Proof.

By example. □
Adopt versus OptAPO

- Adopt is better when communications are fast.
- OptAPO is better when communications are slow.
- Both have very bad worst-case but seem to perform well.