Distributed Constraint Optimization for the Internet-of-Things

Pierre Rust  Gauthier Picard

Orange Labs
pierre.rust@orange.com

MINES Saint-Étienne
LaHC UMR CNRS 5516
picard@emse.fr
Internet-of-Things (IoT) and its Control

- Huge (marketing?) trend today
- 25 billion of connected objects in 2020? (Gartner)
- Hardware and communication is cheaper and cheaper
- Constrained devices
  - limited CPU and memory resources
  - limited communication capabilities

- Connected things’ actions should be **coordinated**
- Current approach: centralizing decisions
  - Communications
  - Resilience
  - Scalability
  - Cost
Distributed Coordination and Decision Making

Autonomous and spontaneous

Coordinating objects to achieve objectives

- Coordination
  - Decentralized
  - Spontaneous
  - Autonomous

- No central point
- Self-adaption to environmental changes
- Self-repair in case of one component failure
About decisions

\[ x_i \]
About decisions

\[ x_i \ ? \]

s.t. “I’m happy with \( x_i \)”
About decisions

\[ x_i \quad ? \quad \text{s.t. “I’m happy with } x_i \text{”} \]

\[ x_j \quad ? \quad \text{s.t. “agent } i \text{ is fine with } x_j \text{”} \]
About decisions

$x_i \ ?$  
\text{s.t. “I’m happy with } x_i \text{”}

$x_j \ ?$  
\text{s.t. “agent } i \text{ is fine with } x_j \text{”}

How can agents autonomously make their decisions in a coordinated way, without external control?
About decisions

\( x_i \) ?

s.t. “I’m happy with \( x_i \)”

\( x_j \) ?

s.t. “agent \( i \) is fine with \( x_j \)”

How can agents autonomously make their decisions in a coordinated way, without external control?

⇒ Decentralized decision making
About decisions

$x_i \quad ?$

s.t. “I’m happy with $x_i$”

$x_j \quad ?$

s.t. “agent $i$ is fine with $x_j$”

How can agents autonomously make their decisions in a coordinated way, without external control?

⇒ Decentralized decision making

- Agents have to coordinate to perform best actions
- Agents form a team → best actions for the team
Application Domains

[Images of various application domains: a cityscape, a circuit board, a road with connected cars, and a field of connected devices.]
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
DCOP\textsuperscript{1}
Distributed Constraints Optimization Problem

Definition (DCOP)
A DCOP is a tuple \((\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu)\), where:
- \(\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}\) is a set of agents
- \(\mathcal{X} = \{x_1, \ldots, x_n\}\) are variables
- \(\mathcal{D} = \{D_{x_1}, \ldots, D_{x_n}\}\) is a set of finite domains, for the \(x_i\) variables
- \(\mathcal{C} = \{f_1, \ldots, f_m\}\) is a set of soft constraints, where each \(c_i\) defines a cost \(\in \mathbb{R} \cup \{\infty\}\) for each combination of assignments to a subset of variables
- \(\mu\) is a function mapping variables to their associated agent

Definition (Solution)
A solution to the DCOP is an assignment \(\mathcal{A}\) to all variables that minimizes \(\sum_i f_i\)

\textsuperscript{1}Some contents taken from OPTMAS 2011 and OPTMAS-DCR 2014
Objective Function

\[ F(A) = \sum_{x_i, x_j \in X} f_{ij} \quad \text{where} \quad f_{ij} = (x_i + x_j + 1) \mod 3 \]

In figure (a):
- \( F(\{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0)\}) = 4 \)
- \( F(\{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1)\}) = 0 \)
But first: how to solve DCOPs?

DCOP Algorithms

Complete

Partially Decentralized

Synchronous

Search

OPTApo

Inference

PC-DPOP

Fully Decentralized

Synchronous

Search

SyncBB

Inference

DPOP and variants

Asynchronous

Search

AFB; ADOPT and variants

Incomplete

Fully Decentralized

Synchronous

Search

D-Gibbs

Inference

Region Optimal DSA; MGM

Sampling

Max-Sum and variants

[Fioretto et al., 2018]
Some issues related to IoT

Internet-of-Things is a physical network infrastructure

- Things are interconnected and very heterogeneous
- Where to place computations (variables and constraints/factors)?
Some issues related to IoT

Internet-of-Things is a physical network infrastructure

- Things are interconnected and very heterogeneous
- Where to place computations (variables and constraints/factors)?

Decision problem by itself
- Constrained by the things’ capacities (memory, communication, CPU, ...)

Pierre Rust, Gauthier Picard
Distributed Constraint Optimization for the Internet-of-Things
Some issues related to IoT (cont.)

Internet-of-Things is an open system

- How to cope with things’ (dis)appearance?
Some issues related to IoT (cont.)

Internet-of-Things is an open system

- How to cope with things’ (dis)appearance?
Some issues related to IoT (cont.)

Internet-of-Things is an open system

- How to cope with things’ (dis)appearance?

- Disappearance: one solution is to replicate computations
  - Where replicas are placed?
  - Which replicate to activate following a disappearance?

- Newcoming things: opportunity to load balance, but...
  - Which computations to move?
This tutorial will thus focus on...

- Using DCOPs to model IoT applicative problems
- Modeling the specific problem of distributing decisions/computations
- Using distributed algorithms and DCOPs to equip IoT applications with resilience
This tutorial will thus focus on...

- Using DCOPs to model IoT applicative problems
- Modeling the specific problem of distributing decisions/computations
- Using distributed algorithms and DCOPs to equip IoT applications with resilience
- All that will be illustrated using the PyDCOP framework
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
Hands on PyDCOP I

- Install VirtualBox
- Import the pyDCOP Virtual Machine (https://bit.ly/2L0Kqns)
  - It’s a Debian image with everything preinstalled:
    - python3, pyDCOP, matplotlib, glpk, etc.
- Alternatively, follow

Hands on PyDCOP I
Virtual machine Setup

Before starting the VM:
- "Bridged adapter" mode
- Select wifi network adapter
- Reset MAC Address

Then
- Start the VM
- Launch a terminal
- Note down the IP with `ip address`
Hands on PyDCOP I

Virtual machine Setup

dcop@debian:$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
default scope host lo
   inet 127.0.0.1 scope host
     valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
     valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
   group default qlen 1000
   link/ether 00:00:27:ec:1d:7c brd ff:ff:ff:ff:ff:ff
   inet 192.168.1.22/24 brd 192.168.1.255 scope global dynamic enp0s3
     valid_lft 86326sec preferred_lft 86326sec
     inet6 fe80:a00:27ff:feec:1d7c/64 scope link
     valid_lft forever preferred_lft forever

dcop@debian:$
Hands on PyDCOP I

Files for the tutorials are in /home/dcop/tutorials.

$ cd /home/dcop/tutorials/hands-on_1
Hands on PyDCOP I
DCOP - Graph Coloring

(a) constraints graph

(b) factor graph

- **Objective:** minimize
- **Domain:** 2 colors $R$ and $B$
- **Variables:** $V_1$, $V_2$, $V_3$
- **Constraints:** neighbors must have different colors + preferences
- **Agents:** 3 agents

Yaml representation

---

Pierre Rust, Gauthier Picard
Distributed Constraint Optimization for the Internet-of-Things
Hands on PyDCOP I
pyDCOP yaml format

graph_coloring.yaml

name: graph coloring
objective: min

domains:
  colors:
    values: [R, G]

variables:
  v1:
    domain: colors
  v2:
    domain: colors
  v3:
    domain: colors

constraints:
  pref_1:
    type: extensional
    variables: v1
    values:
      -0.1: R
      0.1: G

  pref_2:
    type: extensional
    variables: v2
    values:
      -0.1: G
      0.1: R

  pref_3:
    type: extensional
    variables: v3
    values:
      -0.1: G
      0.1: R

  diff_1_2:
    type: intention
    function: 10 if v1 == v2 else 0

  diff_2_3:
    type: intention
    function: 10 if v3 == v2 else 0

agents: [a1, a2, a3, a4, a5]
Hands on PyDCOP I
Solving the Graph Coloring DCOP

Command:

```
$ pydcop solve --algo dpop graph_coloring.yaml
```

Output:

```
...
"assignment": {
  "v1": "R",
  "v2": "G",
  "v3": "R"
},
"cost": -0.1,
...
```

With other algorithms:

```
$ pydcop --timeout 2 solve --algo dsa graph_coloring.yaml
$ pydcop solve --algo mgm --algo_params stop_cycle:20 graph_coloring.yaml
```
Hands on PyDCOP I

Results

Full results :

```json
{
    "agt_metrics": {
        ...
    },
    "assignment": {
        "v1": "R",
        "v2": "G",
        "v3": "R"
    },
    "cost": -0.1,
    "cycle": 20,
    "msg_count": 158,
    "msg_size": 158,
    "status": "FINISHED",
    "time": 0.03201029699994251,
    "violation": 0
}
```

Look at results from mgm and dsa, compared to dpop’s results!
Hands on PyDCOP I

Logs

**Simple:**
use `-v 0..3`

```
$ pydcop -v 2 solve --algo dpop graph_coloring.yaml
```

**Precise:**
use `-log <log.conf>`

```
$ pydcop --log log.conf -t 1 solve --algo dsa graph_coloring.yaml
```

Now, look at algo.log
**Periodic**: "--collect_on period --period <p>"

```bash
$ pydcop --log log.conf -t 10 solve \  
   --collect_on period --period <p> \  
   --algo dsa graph_coloring.yaml
```

**Cycle**: "--collect_on cycle_change"

Only supported with synchronous algorithms!

```bash
$ pydcop solve --algo mgm --algo_params stop_cycle:20 \  
   --collect_on cycle_change --run_metric ./metrics.csv \  
   graph_coloring_50.yaml
```

**Value**: "--collect_on value_change"

```bash
$ pydcop -t 5 solve --algo mgm --collect_on value_change \  
   --run_metric ./metrics_on_value.csv \  
   graph_coloring_50.yaml
```
Hands on PyDCOP I
Run-time metrics

With a bigger graph coloring problem

```
$ pydcop solve --algo mgm --algo_params stop_cycle:20 \ 
    --collect_on cycle_change \ 
    --run_metric ./metrics.csv \ 
    graph_coloring_50.yaml
```

Plotting with matplotlib

```
$ python3 plot_cost.py ./metrics.csv
```

Do the same thing with DSA, look at the result, what do you see?
Hands on PyDCOP I
Run-time metrics

MGM (1720) and DSA (1647), both with 30 cycles
Web-base agent graphical interface:

- Run the web application

  $ cd ~/pydcop-ui
  $ python3 -m http.server

- Launch a browser on http://127.0.0.1:8000

- Solve the dcop with the option --uiport <port> (also, use --delay <delay>)

  $ pydcop -v 3 solve -a mgm -d adhoc --delay 2 --uiport 10000
  ./graph_coloring_3agts_10vars.yaml

- Each agent exposes a web-socket, the web application connects to these websockets and display the agents’ state.
Hands on PyDCOP I

Web-ui
## Hands on PyDCOP I

### Web-ui

![PyDCOP Interface](image)

**Computations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Algo</th>
<th>Type</th>
<th>Size</th>
<th>Value</th>
<th>Cycles</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>v8</td>
<td>dsa</td>
<td>variable</td>
<td>5</td>
<td>G</td>
<td>12</td>
<td>11 / 11</td>
</tr>
<tr>
<td>v4</td>
<td>dsa</td>
<td>variable</td>
<td>20</td>
<td>G</td>
<td>13</td>
<td>48 / 48</td>
</tr>
<tr>
<td>v3</td>
<td>dsa</td>
<td>variable</td>
<td>10</td>
<td>R</td>
<td>12</td>
<td>22 / 22</td>
</tr>
<tr>
<td>v7</td>
<td>dsa</td>
<td>variable</td>
<td>10</td>
<td>R</td>
<td>12</td>
<td>24 / 24</td>
</tr>
</tbody>
</table>
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods
  DPOP
  Max-Sum
  DSA
  MGM

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs
### Distributed Pseudotree Optimization Procedure (DPOP)

[Distributed Pseudotree Optimization Procedure (DPOP) by Petcu and Faltings, 2005](#)

<table>
<thead>
<tr>
<th>PHASES</th>
<th>MESSAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. DFS Tree construction</td>
<td>token passing</td>
</tr>
<tr>
<td>2. Utility phase: from leaves to root</td>
<td><strong>util</strong> (child → parent, constraint table [-child])</td>
</tr>
<tr>
<td>3. Value phase: from root to leaves</td>
<td><strong>value</strong> (parent → children ∪ pseudochildren, parent value)</td>
</tr>
</tbody>
</table>
DFS Tree Phase

■ **Distributed DFS graph traversal:** token, ID, $neighbors(X)$
  1. $X$ owns the token: adds its own ID and sends it in turn to each of its neighbors, which become children
  2. $Y$ receives the token from $X$: it marks $X$ as visited. First time $Y$ receives the token then $parent(Y) = X$. Other IDs in token which are also $neighbors(Y)$ are **pseudoparent**. If $Y$ receives token from neighbor $W$ to which it was never sent, $W$ is pseudochild.
  3. When all $neighbors(X)$ visited, $X$ removes its ID from token and sends it to $parent(X)$.

■ A node is selected as root, which starts

■ When all neighbors of root are visited, the DFS traversal ends
DFS Tree Phase: Example

root

\[ x_1 \]

\[ x_2 \]

\[ x_3 \]

\[ x_4 \]

\[ [x_1] \]

\[ \downarrow \]

\[ x_1 \] parent of \( x_2 \)
DFS Tree Phase: Example

- $x_1$ parent of $x_2$
- $x_2$ parent of $x_3$
- $x_1$ pseudoparent of $x_3$
DFS Tree Phase: Example

- $x_1$ parent of $x_2$
- $x_2$ parent of $x_3$
- $x_1$ pseudoparent of $x_3$
- $x_3$ parent of $x_4$
- $x_3$ pseudoparent of $x_1$
DFS Tree Phase: Example

1. Root:

   - $x_1$ parent of $x_2$
   - $x_2$ parent of $x_3$
   - $x_1$ pseudoparent of $x_3$
   - $x_3$ parent of $x_4$
   - $x_3$ pseudoparent of $x_1$
Util Phase

Agent $X$:

- receives from each child $Y_i$ a cost function: $C(Y_i)$
- combines (adds, joins) all these cost functions with the cost functions with $\text{parent}(X)$ and $\text{pseudoparents}(X)$
- projects $X$ out of the resulting cost function, and sends it to $\text{parent}(X)$

From the leaves to the root
Util Phase: Example

\[ X \]

\[ X \quad Y \quad Z \quad T \]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>
Util Phase: Example

\[
\begin{array}{c|c|c}
X & T \\
\hline
a & a & 1 \\
\hline
a & b & 2 \\
\hline
b & a & 2 \\
\hline
b & b & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
X & Y \\
\hline
a & a & 1 \\
\hline
a & b & 2 \\
\hline
b & a & 2 \\
\hline
b & b & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
X & Z \\
\hline
a & a & 1 \\
\hline
a & b & 2 \\
\hline
b & a & 2 \\
\hline
b & b & 0 \\
\end{array}
\]

parent

children

Remove

Remove duplicates

Keep the min cost

Project out

Parent

Children

Add

All value combinations

Costs are the sum of applicable costs
Util Phase: Example

\[
\begin{array}{cccc}
X & T \\
\hline
a & a & 1 \\
a & b & 2 \\
b & a & 2 \\
b & b & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
X & Y & Z & T \\
\hline
a & a & a & a & 3 \\
a & a & a & b & 4 \\
a & a & b & a & 4 \\
a & a & b & b & 5 \\
a & b & a & a & 4 \\
a & b & a & b & 5 \\
a & b & b & a & 5 \\
a & b & b & b & 6 \\
b & a & a & a & 6 \\
b & a & a & b & 4 \\
b & a & b & a & 4 \\
b & a & b & b & 2 \\
b & b & a & a & 4 \\
b & b & a & b & 2 \\
b & b & b & a & 2 \\
b & b & b & b & 0 \\
\end{array}
\]

Add

Remove

Remove duplicates

Keep the min cost

Project out

All value combinations
Costs are the sum of applicable costs
**Util Phase: Example**

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>2</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>2</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
</tbody>
</table>

**Add**

**Parent**

**Children**

**All value combinations**

Costs are the sum of applicable costs

**Remove**

Remove duplicates

Keep the min cost

**Project out $X$**
1. The root finds the **value that minimizes the received cost function** in the util phase, and informs its descendants (children $\cup$ pseudochildren)

2. Each agent **waits to receive** the value of its parent / pseudoparents

3. Keeping fixed the value of parent/pseudoparents, finds **the value that minimizes the received cost function** in the Util phase

4. Informs of this value to its children/pseudochildren

This process starts at the root and ends at the leaves
DTREE: DPOP for DCOPs without backedges

Optimal solution: linear number of messages, message size: linear
DTREE : DPOP for DCOPs without backedges

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Y</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>
DTREE: DPOP for DCOPs without backedges

Optimal solution: linear number of messages, message size: linear
DTREE: DPOP for DCOPs without backedges

Optimal solution: linear number of messages, message size: linear
DTREE : DPOP for DCOPs without backedges

Optimal solution:

- Linear number of messages
- Message size: linear
DTREE: DPOP for DCOPs without backedges

Optimal solution: linear number of messages
message size: linear
DTREE: DPOP for DCOPs without backedges

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

Optimal solution:
- Linear number of messages
- Message size: linear
DTREE: DPOP for DCOPs without backedges

Optimal solution:
- Linear number of messages
- Message size: linear

Pierre Rust, Gauthier Picard
DTREE: DPOP for DCOPs without backedges

Optimal solution:
- linear number of messages
- message size: linear
DPOP for any DCOP
DPOP for any DCOP

<table>
<thead>
<tr>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>b</td>
</tr>
</tbody>
</table>

Optimal solution: linear number of messages, message size: exponential
DPOP for any DCOP

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

Optimal solution:
linear number of messages
message size: exponential
DPOP for any DCOP

Optimal solution: linear number of messages, message size: exponential
DPOP for any DCOP

### Example Constraints

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Z</th>
<th></th>
<th>Y</th>
<th>W</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
<td></td>
<td>a</td>
<td>a</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
<td></td>
<td>a</td>
<td>b</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
<td></td>
<td>a</td>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
<td></td>
<td>b</td>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td></td>
<td>a</td>
<td>a</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
<td></td>
<td>a</td>
<td>b</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
<td></td>
<td>b</td>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
<td></td>
<td>b</td>
<td>b</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The optimal solution is: linear number of messages, message size: exponential.
DPOP for any DCOP

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>0</td>
</tr>
</tbody>
</table>
DPOP for any DCOP

\[
\begin{array}{ccc}
X & Z \\
a & a & 1 \\
a & b & 2 \\
b & a & 2 \\
b & b & 0 \\
Y & W \\
a & a & 1 \\
a & b & 2 \\
b & a & 2 \\
b & b & 0 \\
Y & Z \\
a & a & 1 \\
a & b & 2 \\
b & a & 2 \\
b & b & 0 \\
X & Y \\
a & a & 1 \\
a & b & 2 \\
b & a & 2 \\
b & b & 0 \\
X & X \\
a & a & 1 \\
a & b & 2 \\
b & a & 2 \\
b & b & 0 \\

\end{array}
\]
DPOP for any DCOP

<table>
<thead>
<tr>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

Optimal solution:
- linear number of messages
- message size: exponential

Pierre Rust, Gauthier Picard
DPOP for any DCOP

Optimal solution:
- linear number of messages
- message size: exponential
GDL-based approaches

- **Generalized Distributive Law** [Aji and McEliece, 2000]
  - Unifying framework for inference in Graphical models
  - Builds on basic mathematical properties of semi-rings
  - Widely used in Info theory, Statistical physics, Probabilistic models

- **Max-sum**
  - DCOP settings: maximise social welfare

<table>
<thead>
<tr>
<th>( K )</th>
<th>( (+, 0) )</th>
<th>((\cdot, 1))</th>
<th>short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( A )</td>
<td>( (+, 0) )</td>
<td>((\cdot, 1))</td>
<td>sum-product</td>
</tr>
<tr>
<td>2. ( A[x] )</td>
<td>( (+, 0) )</td>
<td>((\cdot, 1))</td>
<td>min-product</td>
</tr>
<tr>
<td>3. ( A[x, y, \ldots] )</td>
<td>( (+, 0) )</td>
<td>((\cdot, 1))</td>
<td>max-product</td>
</tr>
<tr>
<td>4. ( [0, \infty) )</td>
<td>( (+, 0) )</td>
<td>((\cdot, 1))</td>
<td>min-sum</td>
</tr>
<tr>
<td>5. ( (0, \infty] )</td>
<td>( (\min, \infty) )</td>
<td>((\cdot, 1))</td>
<td>max-sum</td>
</tr>
<tr>
<td>6. ( [0, \infty) )</td>
<td>( (\max, 0) )</td>
<td>((\cdot, 1))</td>
<td>Boolean</td>
</tr>
<tr>
<td>7. ( (-\infty, \infty) )</td>
<td>( (\max, -\infty) )</td>
<td>((+, 0))</td>
<td></td>
</tr>
<tr>
<td>8. ( (-\infty, \infty) )</td>
<td>( (\text{union}, \emptyset) )</td>
<td>((\cap, S))</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>( (\text{OR}, 0) )</td>
<td>((\text{AND}, 1))</td>
<td></td>
</tr>
<tr>
<td>10. ( 2^S )</td>
<td>( (\cup, \emptyset) )</td>
<td>((\cap, S))</td>
<td></td>
</tr>
<tr>
<td>11. ( \Lambda )</td>
<td>( (\lor, 0) )</td>
<td>((\land, 1))</td>
<td></td>
</tr>
<tr>
<td>12. ( \Lambda )</td>
<td>( (\land, 1) )</td>
<td>((\lor, 0))</td>
<td></td>
</tr>
</tbody>
</table>
Agents iteratively computes local functions that depend only on the variable they control.
Agents iteratively computes local functions that depend only on the variable they control
Agents iteratively compute local functions that depend only on the variable they control.

\[
m_{1 \rightarrow 2}(x_2) = \max_{x_1} (F_{12}(x_1, x_2) + m_{4 \rightarrow 1}(x_1))
\]
Agents iteratively computes local functions that depend only on the variable they control

\[ m_{1 \rightarrow 2}(x_2) = \max_{x_1} \left( F_{12}(x_1, x_2) + m_{4 \rightarrow 1}(x_1) \right) \]
Agents iteratively computes local functions that depend only on the variable they control.
Agents iteratively computes local functions that depend only on the variable they control
Agents iteratively compute local functions that depend only on the variable they control.

\[ m_{1 \rightarrow 2}(x_2) = \max_{x_1} (\text{message terms}) + \text{message terms} \]

\[ z_1(x_1) = m_{4 \rightarrow 1}(x_1) + m_{2 \rightarrow 1}(x_1) \]
Agents iteratively computes local functions that depend only on the variable they control.

\[ m_{1 \rightarrow 2}(x_2) = \max_{x_1} (\text{message 1} + \text{message 2}) \]

\[ z_1(x_1) = m_{4 \rightarrow 1}(x_1) + m_{2 \rightarrow 1}(x_1) \]
Agents iteratively compute local functions that depend only on the variable they control.

\[
m_1 \rightarrow 2(x_2) = \max_{x_1} (\text{Shared constraint}) + m_2 \rightarrow 1(x_1)
\]

\[
z_1(x_1) = \text{All incoming messages}
\]

\[
m_4 \rightarrow 1 \to m_2 \rightarrow 1(x_1)
\]

\[
\text{All incoming messages except } x_2
\]
Agents iteratively computes local functions that depend only on the variable they control.
Max-Sum on acyclic graphs

- Max-sum Optimal on acyclic graphs
  - Different branches are independent
  - Each agent can build a correct estimation of its contribution to the global problem ($z$ functions)

- Message equations very similar to Util messages in DPOP
  - Sum messages from children and shared constraint
  - Maximize out agent variable
  - GDL generalizes DPOP [Vinyals et al., 2011]

$$m_{1 \rightarrow 2}(x_2) = \max_{x_1} (F_{12}(x_1, x_2) + m_{4 \rightarrow 1}(x_1))$$
Max-Sum Performance

- **Good performance on loopy networks** [Farinelli et al., 2008]
  - When it converges very good results
  - Interesting results when only one cycle [Weiss, 2000]
  - We could remove cycle but pay an exponential price (see DPOP)
Max-Sum for low power devices

- Low overhead
  - Msgs number/size
- Asynchronous computation
  - Agents take decisions whenever new messages arrive
- Robust to message loss
Local Greedy Approaches

- **Greedy local search**
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one
Local Greedy Approaches

- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one

![Diagram showing a network of nodes and connections with values and changes indicated.](image-url)
Local Greedy Approaches

- **Greedy local search**
  - Start from random solution
  - Do local changes if global solution improves
  - **Local**: change the value of a subset of variables, usually one
Local Greedy Approaches

- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one
Local Greedy Approaches

- **Greedy local search**
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one
Local Greedy Approaches

- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing
Local Greedy Approaches

- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing
Local Greedy Approaches

Problems
- Local minima
- Standard solutions: Random Walk, Simulated Annealing
Local Greedy Approaches

- Problems
  - Local minima
  - Standard solutions: Random Walk, Simulated Annealing
Distributed Local Greedy approaches

- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination
Distributed Local Greedy approaches

- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination
Distributed Local Greedy approaches

- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination
Distributed Local Greedy approaches

- Local knowledge
- Parallel execution
  - A greedy local move might be harmful/useless
  - Need coordination
Distributed Stochastic Search Algorithm (DSA)

[Zhang et al., 2005]

- Greedy local search with activation probability to mitigate issues with parallel executions
- DSA-1: change value of one variable at time
- Initialize agents with a random assignment and communicate values to neighbors
- Each agent:
  - Generates a random number and execute only if rnd less than activation probability
  - When executing changes value maximizing local gain
  - Communicate possible variable change to neighbors
DSA-1: Execution Example
DSA-1: Execution Example
DSA-1: Execution Example

-1 -1 -1

-2 0
DSA-1: Execution Example

\[ -1 \quad -1 \quad -1 \quad -1 \quad -1 \]
DSA-1: Execution Example
DSA-1: Discussion

- Extremely “cheap” (computation/communication)
- Good performance in various domains
  - e.g. target tracking [Fitzpatrick and Meertens, 2003; Zhang et al., 2003]
  - Shows an anytime property (not guaranteed)
  - Benchmarking technique for coordination
- Problems
  - Activation probability must be tuned [Zhang et al., 2003]
  - No general rule, hard to characterise results across domains
Maximum Gain Message (MGM-1)
[MAHESWARAN et al., 2004]

- Coordinate to decide who is going to move
  - Compute and exchange possible gains
  - Agent with maximum (positive) gain executes

- Analysis
  - Empirically, similar to DSA
  - More communication (but still linear)
  - No Threshold to set
  - Guaranteed to be monotonic (Anytime behavior)
MGM-1: Example
MGM-1: Example

\[ \begin{array}{c}
-2 \\
0
\end{array} \]
MGM-1: Example

\[
G = -2
\]
MGM-1: Example

\[ G = -2 \]
MGM-1: Example

\[
G = -2
\]

\[
G = 0
\]
MGM-1: Example

\[ G = -2 \]

\[ G = 0 \]
MGM-1: Example

\[ G = -2 \]

\[ G = 0 \]

\[ G = 2 \]
MGM-1: Example

\[ G = -2 \]

\[ G = 0 \]

\[ G = 2 \]
MGM-1: Example

\[ G = -2 \]

\[ G = 0 \]

\[ G = 2 \]

\[ G = 0 \]
MGM-1: Example

\[ G = -2 \]

\[ G = 0 \]

\[ G = 2 \]

\[ G = 0 \]
MGM-1: Example

\[ G = -2 \]

\[ G = 0 \]

\[ G = 2 \]

\[ G = 0 \]
MGM-1: Example

```
-1
```

```
-1
```
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
pyDCOP is designed to make it easy to implement new DCOP algorithms

- All the infrastructure is provided:
  - agents,
  - messaging,
  - metrics,
  - etc.

- Base classes and utility functions for
  - constraints,
  - variables,
  - domains,
  - etc.

- Plugin mechanism to define new algorithms for DCOP, distribution and replication.
Create a new python module in pydcop.algorithms

- Define a constant indicating the graphical representation used by your algorithm: `GRAPH_TYPE = 'constraints_hypergraph'`
- Define your messages:
  ```python
  message_type(<name>, [fields]);
  ```
- Subclass `VariableComputation`:
  Register your message handlers with `self.msg_handlers`;
  Send messages to your neighbors using `self.post_msg` or `self.post_to_all_neighbors`,
  Select a new value with `self.value_selection`
  Start a new cycle with `self.new_cycle()`.
Hands on PyDCOP II
Simple DSA implementation

One class, 5 methods:

```python
GRAPH_TYPE = 'constraints_hypergraph'
algo_params = []

DsaMessage = message_type("DsaMessage", ["value"])

class DsaTutoComputation(VariableComputation):

    def __init__(self, variable, constraints, computation_definition):
        ...

    def on_start(self):
        ...

    def on_value_msg(self, variable_name, recv_msg, t):
        ...

    def evaluate_cycle(self):
        ...

    def compute_best_value(self) -> Tuple[Any, float]:
        ...
```
class DsaTutoComputation(VariableComputation):
    def __init__(self, variable, constraints, computation_definition):
        super().__init__(variable, computation_definition)
        self._msg_handlers['DsaMessage'] = self.on_value_msg
        self.constraints = constraints
        self.current_cycle = {}
        self.next_cycle = {}

    def on_start(self):
        self.random_value_selection()
        self.post_to_all_neighbors(DsaMessage(self.current_value))
        self.evaluate_cycle()

    def on_value_msg(self, variable_name, recv_msg, t):
        if variable_name not in self.current_cycle:
            self.current_cycle[variable_name] = recv_msg.value
            self.evaluate_cycle()
        else:  # The message is for the next cycle
            self.next_cycle[variable_name] = recv_msg.value
def evaluate_cycle(self):
    if len(self.current_cycle) == len(self.neighbors):
        # Values from all neighbors received for this cycle:
        self.current_cycle[self.variable.name] = self.current_value
        current_cost = assignment_cost(self.current_cycle, self.constraints)
        arg_min, min_cost = self.compute_best_value()
        # Change value?
        if current_cost > min_cost and 0.5 > random.random():
            self.value_selection(arg_min)
        # Start a new cycle:
        self.new_cycle()
        self.current_cycle, self.next_cycle = self.next_cycle, {}
        self.post_to_all_neighbors(DsaMessage(self.current_value))

def compute_best_value(self) -> Tuple[Any, float]:
    # Find the value from our domain that yields the best cost:
    arg_min, min_cost = None, float('inf')
    for value in self.variable.domain:
        self.current_cycle[self.variable.name] = value
        cost = assignment_cost(self.current_cycle, self.constraints)
        if cost < min_cost:
            arg_min, min_cost = value, cost
    return arg_min, min_cost
We can now use this new algorithm directly through the command line interface (except for `stop_cycle: 20`):

```
$ pydcop --log log.conf -t 20 solve --algo dsatuto \ 
   --collect_on value_change \ 
   --run_metric ./metrics_tuto.csv \ 
   graph_coloring_50.yaml
```

Of course, it also works with the metrics, web-ui, etc.
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
SECP model

Smart Environment Configuration Problem [Rust et al., 2016]

- Example of applying DCOPs to a "real" problem
- Coordinate objects in the building
- Model
  - objects
  - relations between objects and environment
  - user objectives and requirements
- Formulate the problem as an optimization problem
SECP model

*Smart Environment Configuration Problem* [Rust et al., 2016]

Focus on smart lighting use cases

- **Objects:** anything that can produce light: light bulbs, windows with rolling shutter, etc.
- **User preferences:** having a predefined luminosity level in a room, under some conditions

**Energy efficiency**

Linking objects and user preferences:

- How to model the luminosity in a room? **variable**
- How to model the dependency between the light sources and the luminosity? **function / constraint**


**SECP model**

Example application to ambient intelligence scenario

- **Actuators**
  - Connected light bulbs, TV, Rolling shutters, ...

- **Sensors**
  - Presence detector, Luminosity Sensor, etc.

- **Physical Dependency Models**
  - E.g. Living-room light model

- **User Preferences**
  - Expressed as rules:

    IF presence_living_room = 1
    AND light_sensor_living_room < 60
    THEN light_level_living_room ← 60
    AND shutter_living_room ← 0
### SECP model

Example application to ambient intelligence scenario

- **Actuators**
  - *Decision variable* $x_i$, domain $\mathcal{D}_{x_i}$
  - *Cost function* $c_i : \mathcal{D}_{x_i} \rightarrow \mathbb{R}$

- **Sensors**
  - *Read-only variable* $s_l$, domain $\mathcal{D}_{s_l}$

- **Physical Dependency Models** $\langle y_j, \phi_j \rangle$
  - Give the expected state of the environment from a set of actuator-variables influencing this model
  - Variable $y_j$ representing the *expected* state of the environment
  - Function $\phi_j : \prod_{\varsigma \in \sigma(\phi_j)} \mathcal{D}_{\varsigma} \rightarrow \mathcal{D}_{y_j}$

- **User Preferences**
  - *Utility function* $u_k$
  - Distance from the current expected state to the target state of the environment
Formulating SECP as a DCOP

Multi-objective optimization problem

\[
\begin{align*}
\min_{x_i \in \nu(A)} & \sum_{i \in A} c_i \\
\text{and} & \sum_{i \in A} c_i \\
\max_{x_i \in \nu(A)} & \sum_{y_j \in \nu(\Phi)} u_k \\
\text{s.t.} & \phi_j(x_1^j, \ldots, x_j^j) = y_j \quad \forall y_j \in \nu(\Phi)
\end{align*}
\]

Then mono-objective DCOP formulation

\[
\begin{align*}
\max_{x_i \in \nu(A)} \quad \omega_u \sum_{k \in R} u_k - \omega_c \sum_{i \in A} c_i + \sum_{\varphi_j \in \Phi} \varphi_j \\
\text{with reformulation of hard constraints } \phi_j \text{ into soft ones:}
\end{align*}
\]

\[
\varphi_j(x_1^j, \ldots, x_j^{|\sigma(\phi_j)|}, y_j) = \begin{cases} 
0 & \text{if } \phi_j(x_1^j, \ldots, x_j^{|\sigma(\phi_j)|}) = y_j \\
-\infty & \text{otherwise}
\end{cases}
\]
Formulating SECP as a DCOP

Representing a DCOP as a factor graph
SECP Factor Graph
in a house (without rules)
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
Distribution of computations
Allocating computations to agents

- DCOP: \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu \rangle
- \mu: function mapping variables to their associated agent

Why is distribution needed?

Common assumptions:
- computation \equiv variable
- each agent controls exactly one variable (bijection)
- binary constraints

Real distributed problems:
- agents must be hosted on real devices
- the set of devices might be given by the problem
- for some variables the relation with an agent is obvious, but not always
Distribution of computations

Allocating computations to agents

- Distributing computations
  - computations depends on the graph model
  - variables and / or factors

- Distribution impacts the system characteristics:
  - speed,
  - communication load,
  - hosting costs, etc.
Distribution of computations
Allocating computations to agents

- Simple heuristic
  - No computation on sleepy devices (sensors)
  - Computation should be close the the impacted variables
  - Spread the computation load amongst agents

- How good is it?
Optimal distribution?

- Problem dependent
- Optimization problem: find the best distribution, for your problem’s criteria
- Optimal distribution $\equiv$ graph partitioning, NP-hard in general [Boulle, 2004]
Distribution of computations
Better definition

SECP distribution problem
- Devices have limited memory
- Communication is expensive and has limited bandwidth
- Variable related to an actuator are hosted by it
- Objective: **minimize overall communication between agents**

Optimization problem: define an ILP for it!
Binary ILP for computation distribution

- $x^k_i$, binary variables that map computations to agents and $\alpha^m_{ij} = x^m_i \cdot f^m_j$

\[ \forall x_i \in X, \sum_{a_m \in A} x^m_i = 1 \quad (1) \]

- Message's size between variable $x_i$ and factor $f_j$: $msg(i, j)$

\[ \text{minimize} \sum_{x^m_i} \sum_{(i, j) \in D} \sum_{(m, n) \in A^2} \text{msg}(i, j) \cdot \alpha^m_{ij} \quad (2) \]

- Memory footprint of a computation: $\text{weight}(e)$, and memory capacity for a device: $\text{cap}(a_k)$

\[ \forall a_m \in A, \sum_{x_i \in D} \text{weight}(x_i) \cdot x^m_i \leq \text{cap}(a_m) \quad (3) \]

- and a few linearization constraints
Binary ILP for computation distribution

More generic case:

- Add route cost: \( \text{com}(i, j, m, n) \)

\[
\forall x_i, x_j \in X, \forall a_m, a_n \in A,
\text{com}(i, j, m, n) = \begin{cases} 
\text{msg}(i, j) \cdot \text{route}(m, n) & \text{if } (i, j) \in D, m \neq n \\
0 & \text{otherwise}
\end{cases}
\] (4)

\[
\text{minimize} \sum_{x_i^m} \sum_{(i,j) \in D} \sum_{(m,n) \in A^2} \text{com}(i, j, m, n) \cdot \alpha^m_{ij}
\] (5)

- Add hosting costs: \( \text{host}(a_m, x_i) \)

\[
\text{minimize} \sum_{x_i^m} \sum_{(x_i, a_m) \in X \times A} x_i^m \cdot \text{host}(a_m, x_i)
\] (6)
Binary ILP for computation distribution

\[
\begin{align*}
\text{minimize} & \quad \omega_{\text{com}} \cdot \sum_{(i,j) \in D} \sum_{(m,n) \in A^2} \text{com}(i,j,m,n) \cdot \alpha_{ij}^{mn} \\
& \quad + \omega_{\text{host}} \cdot \sum_{(x_i, a_m) \in X \times A} x_i^m \cdot \text{host}(a_m, x_i) \\
\text{subject to} & \\
\forall a_m \in A, \quad & \sum_{x_i \in D} \text{weight}(x_i) \cdot x_i^m \leq \text{cap}(a_m) \\
\forall x_i \in X, \quad & \sum_{a_m \in A} x_i^m = 1 \\
\forall x_i \in X, \quad & \alpha_{ij}^{mn} \leq x_i^m \\
\forall x_j \in X, \quad & \alpha_{ij}^{mn} \leq x_j^m \\
\forall x_i, x_j \in X, a_m \in A, \quad & \alpha_{ij}^{mn} \geq x_i^m + x_j^m - 1
\end{align*}
\]
Solving the ILP for computation deployment

- NP-hard, but can be solved with branch-and-cut
  LP solvers are very good at this
- Yet, only possible for small instances
- Gives us a reference for optimality: benchmarking
- When not solvable, still gives us a metrics to compare heuristics
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
A Very simple SECP: single room

- 3 light bulbs, 1 model and 3 rules
- /tutorials/hands-on_3/single_room.yaml
- Solve with

```
pydcop --log log.conf -t 10 solve \
   --algo maxsum --algo_params damping:0.8 \ 
   --dist adhoc single_room.yaml
```

- Result: "cost": 702.3000000000004, ...
  - not that good ...
  - Look at the yaml definition
  - the rules contradict each other!

- Change the yaml definition
  - comment out rules to keep only one active
  - could be done with 'read-only' variables
  - solve it again
■ We used `solve`:
  ▶ great for testing
  ▶ everything run locally, in the same process

■ Launching several agents:
  ▶ One agent for each light bulb `a1`, `a2` and `a3` (change port for each agent)

```bash
pydcop -v3 agent -n a1 -p 9001 \  --orchestrator 127.0.0.1:9000
```

▶ an orchestrator

```bash
pydcop --log log.conf -t 10 orchestrator \  --algo maxsum --algo_params damping:0.8 \  --dist adhoc single_room.yaml
```

▶ run the agents on different Virtual machines, different computers
in /tutorials/hands-on_3/SimpleHouse.yml
13 light bulbs, 6 models
$ pydcop --output dist_house_fg_ilp.yaml distribute -d ilp_compref \ 
    -a maxsum SimpleHouse.yml

Need to specify the algorithm, used to deduce:

- the computation graph
- the computations’ weight
- the size of computations’ messages

On such a small system, we can compute the optimal distribution!
Hands on PyDCOP III
Distributing a SECP

cost: 8725.0
distribution:
  a_d1: [mv_desk, mc_desk, l_d1, r_work, mc_livingroom, mv_livingroom]
  a_d2: [l_d2]
  a_e1: [mv_entry, r_entry, mv_stairs, l_e1, mc_entry, mc_stairs]
  a_e2: [l_e2]
  a_k1: [l_k1]
  a_k2: [l_k2]
  a_k3: [l_k3]
  a_lv1: [l_lv1]
  a_lv2: [mc_kitchen, l_lv2]
  a_lv3: [l_lv3]
  a_tv1: [l_tv1]
  a_tv2: [l_tv2]
  a_tv3: [r_lunch, l_tv3, mv_tv, r_cooking, r_homecinema, mc_tv, mv_kitchen]
inputs:
  algo: maxsum
dcop: [SimpleHouse.yml]
dist_algo: ilp_compref
graph: factor_graph
Menu

DCOP Framework

Hands on PyDCOP I

Focus on Some Solution Methods

Hands on PyDCOP II

Focus on Smart Environment Configuration Problems

Distributing Computations

Hands on PyDCOP III

Dynamic DCOPs

Conclusion
So far we have:

- Designed a model for SECP
- Formulated this model as a DCOP
- Distributed the computation of the DCOP on devices / agents (bootstrap)
- Run our system to get self-configured devices
But what happens in dynamic environments if objects appear and disappear?
SECP is a dynamic problem

Dynamics in the infrastructure

- Devices can disappear
- New devices can be added to the system

At runtime..

- No powerful device available to solve the ILP
- The deployment must be repaired: self-adaptation
- Only consider a portion of the factor graph: the neighborhood
Definition ($k$-resiliency)

$k$-resiliency is the capacity for a system to repair itself and operate correctly even in the case of the disappearance of up to $k$ agents

- Two parts:
  - Do not lose the definition of the computations: replication
  - Migrate the orphaned computations to another agent: selection / activation

- Apply to any graph of computations, not only DCOP
Replication of computations

Replica distribution

- For each computation, place $k$ replica on $k$ other agents
  replica $equiv$ definition of the computation
- Must be distributed!
- Optimal replication? impact the set of available agents when repairing
  which criteria? too hard (quadratic multiple knapsack problem)...

Distributed Replica Placement Method (DRPM)

- Heuristic: place replica on agents close (network) the active computation,
  while respecting capacity
- Distributed version of iterative lengthening (aka uniform cost search based
  on path costs)
Replication of computations
iterative lengthening on route and hosting costs

Figure: A sample route+host-graph with 4 agents (in gray), where $a_1$ search for hosting computation $x_i$. For $k = 2$, DRMP places a replica on $a_2$ (cost of $1 + 1 = 2$) and another on $a_3$ (cost of $1 + 3 + 1 = 5$) if enough capacity on these two agents, since the minimum cost path to host on $a_4$ is higher ($1 + 5 = 6$).
Migrating computations

Selecting an agent

Migrating a set of $x_i$ computations $X_c$

- set of candidate agents $A_c$
- migrating the computation must not exceed agent’s capacity
- for each computation, select the agent that minimize hosting and communication cost

Same optimization problem than for initial distribution, but on a subset of the graph

Distributed process!
Distributed optimization problem \( \Rightarrow \) let’s use a DCOP!

- \( \mathcal{A} \) is the set of candidate agents \( A_c \)
- \( \mathcal{X} \) are the binary decision variables \( x_i^m \)
- \( \mathcal{C} \) are the constraints ensuring that all computations are hosted, agent’s capacities are respected and hosting and communication costs are minimized
Migrating computations

Selecting an agent

\[ \sum_{a_m \in A^i_c} x^m_i = 1 \]  \hspace{1cm} (13)

\[ \sum_{x_i \in X^m_c} \text{weight}(x_i) \cdot x^m_i + \sum_{x_j \in \mu^{-1}(a_m) \setminus X_c} \text{weight}(x_j) \leq \text{cap}(a_m) \]  \hspace{1cm} (14)

\[ \sum_{x_i \in X^m_c} \text{host}(a_m, x_i) \cdot x^m_i \]  \hspace{1cm} (15)

\[ \sum_{(x_i, x_j) \in X^m_c \times N_i \setminus X_c} x^m_i \cdot \text{com}(i, j, m, \mu^{-1}(x_j)) \]

\[ + \sum_{(x_i, x_j) \in X^m_c \times N_i \cap X_c} x^m_i \cdot \sum_{a_n \in A^j_c} x^n_j \cdot \text{com}(i, j, m, n) \]  \hspace{1cm} (16)
Decentralized reparation

When agents are removed:

- computation to migrate = computation that were hosted on these agents
- candidate agents = remaining agents that posses a replica of these orphaned computation
Solving the migration DCOP
Which algorithm should we use?

Criteria:
- lightweight
- fast (even if not optimal !)
- monotonic : mix of hard and soft constraints

MGM-2 : like MGM, with 2-coordination
Experimental results

Graph showing the comparison between the average cost with and without perturbation over time. The x-axis represents time in seconds (0 to 240), and the y-axis represents cost (0 to 9000). Two lines are plotted: one in blue for the average cost with perturbation and one in red for the average cost without perturbation.
Menu

DCOP Framework
Hands on PyDCOP I
Focus on Some Solution Methods
Hands on PyDCOP II
Focus on Smart Environment Configuration Problems
Distributing Computations
Hands on PyDCOP III
Dynamic DCOPs

Conclusion
To sum up

What we’ve seen today:

- Some generic concepts
  - How to model coordination problems using DCOP formalism
  - Some solution methods (complete and incomplete) to solve DCOP

- Some specificities of IoT-based apps
  - How to model a specific smart environment configuration problem as a DCOP
  - How to use PyDCOP to model, run, solve, and distribute DCOP
  - How to equip a system with resilience using replication and DCOP-based reparation

- Want to go deeper into DCOPs → OPTMAS-DCR workshop series (AAMAS/IJCAI), other tutorials at AAMAS/IJCAI
The End
References


