OptMAS-DCR 2014
Tutorial
CONTENT

- Focus: Optimization in multi-agent systems
- Distributed Constraint Optimization Techniques by William Yeoh
- Game-theoretic Optimization Techniques by Archie Chapman
- Market-based Resource Allocation and Group Formation by Meritxell Vinyals
Part 1: Distributed Constraint Optimization Techniques
WHAT YOU WILL LEARN

• A high-level overview of the different areas
  • Which is the right model for your problem?
• Some representative algorithms for each of the different areas
  • How do these algorithms look like? Can I use them out of the box?
• Pointers to relevant literature
  • Where can I find more information?
MOTIVATING DOMAIN: SENSOR NETWORK

Model the problem as a Max-DCSP
MOITIVATING DOMAIN: SENSOR NETWORK

Model the problem as a Max-DCSP

<table>
<thead>
<tr>
<th>x</th>
<th>x</th>
<th>x</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>E</td>
<td>W</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>W</td>
<td>0</td>
</tr>
</tbody>
</table>

DCOP
DCOP MODEL

\(<A, V, D, F, \alpha>\)

- A: finite set of agents
- V: finite set of variables
- D: finite set of domain values for each variable
- F: finite set of constraints between variables
- \(\alpha\): mapping of variables to agents
**DCOP Model**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
DCOP MODEL

Assumptions:
• Agents are operating distributedly
• Agents need to decide the values that they take on
• Agents know only the constraints that they are involved in

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
DCOP MODEL

Motivating Domain: Sensor Network
- Sensors → Agents
- Directions → Values
- Sensor coordination → Constraints
- Target preferences → Constraint utilities
DCOP MODEL
DCOP ALGORITHMS

DCOP Algorithms

Complete Algorithms
- Search Algorithms
  - e.g., SBB, ADOPT, AFB
- Inference Algorithms
  - e.g., DPOP, Action-GDL

Incomplete Algorithms
- Search Algorithms
  - e.g., MGM, DBA, DSA
- Inference Algorithms
  - e.g., max-sum
- Sampling Algorithms
  - e.g., DUCT, D-Gibbs
DPOP

- Distributed Pseudotree Optimization Procedure (DPOP)
  - extension of Bucket Elimination (BE), which is used to solve COPs
  - dynamic programming based algorithm
DPOP

- **Phase 1: Pseudo-tree construction phase**
  - Constructs a pseudo-tree to partially order all the variables
  - Most complete DCOP algorithms require this

- **Phase 2: UTIL propagation phase**
  - Propagates utilities up the pseudo-tree similar to dynamic programming

- **Phase 3: VALUE propagation phase**
  - Each agent chooses its best value based on the propagated utilities and its ancestors’ values
## DPOP

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Pseudo-tree:

- Ordering of variables into a “tree” such that no two variables in neighboring subtrees share a constraint in the constraint graph

DPOP: PSEUDO-TREE CONSTRUCTION
DPOP: PSEUDO-TREE CONSTRUCTION

- Pseudo-tree:
  - Ordering of variables into a “tree” such that no two variables in neighboring subtrees share a constraint in the constraint graph
• Pseudo-tree:
  • Ordering of variables into a “tree” such that no two variables in neighboring subtrees share a constraint in the constraint graph

• Pseudo-tree construction algorithms:
  • Distributed DFS [Youssef Hamadi, Christian Bessière, Joël Quinqueton: Distributed Intelligent Backtracking. ECAI 1998: 219-223]
  • Distributed greedy algorithm that orders variables according to some heuristic (e.g., most constrained variable)
DPOP

• Phase 1: Pseudo-tree construction phase
  • Constructs a pseudo-tree to partially order all the variables
  • Most complete DCOP algorithms require this

• Phase 2: UTIL propagation phase
  • Propagates utilities up the pseudo-tree similar to dynamic programming

• Phase 3: VALUE propagation phase
  • Each agent chooses its best value based on the propagated utilities and its ancestors’ values
DPOP: UTIL PROPAGATION

- Agent C
  - joins its constraints (sums up the utilities for all possible combinations of its ancestors)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>U(A,C)</th>
<th>U(B,C)</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th></th>
<th></th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
DPOP: UTIL PROPAGATION

- Agent C
  - joins its constraints (sums up the utilities for all possible combinations of its ancestors)
  - projects out itself (takes the max over all its values for each combination)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>5</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>20</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
DPOP: UTIL PROPAGATION

- Agent C
  - joins its constraints (sums up the utilities for all possible combinations of its ancestors)
  - projects out itself (takes the max over all its values for each combination)
  - sends the projected utility matrix to its parent agent B

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>
DPOP: UTIL PROPAGATION

- Agent B
  - joins its constraints (also sums its child's utilities)
  - projects out itself
  - sends the projected utility matrix to its parent agent A

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>U(A,B)</th>
<th>U(C)</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>25</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td>25</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>40</td>
<td>43</td>
</tr>
</tbody>
</table>
DPOP: UTIL PROPAGATION

- Agent B
  - joins its constraints (also sums its child's utilities)
  - projects out itself
  - sends the projected utility matrix to its parent agent A

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>U(A,B)</th>
<th>U(C)</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>25</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td>25</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>40</td>
<td>43</td>
</tr>
</tbody>
</table>
• Agent B
  • joins its constraints (also sums its child's utilities)
  • projects out itself
  • sends the projected utility matrix to its parent agent A
DPOP: UTIL PROPAGATION

- Agent A
  - joins its constraints (if necessary)

<table>
<thead>
<tr>
<th></th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>45</td>
</tr>
</tbody>
</table>
DPOP

• Phase 1: Pseudo-tree construction phase
  • Constructs a pseudo-tree to partially order all the variables
  • Most complete DCOP algorithms require this

• Phase 2: UTIL propagation phase
  • Propagates utilities up the pseudo-tree similar to dynamic programming

• Phase 3: VALUE propagation phase
  • Each agent chooses its best value based on the propagated utilities and its ancestors’ values
DPOP: VALUE PROPAGATION

- Agent A
  - chooses its value that maximizes its utilities
  - sends its value to its child agent B and pseudo-child agent C
DPOP: VALUE PROPAGATION

- Agent B
  - chooses its value that maximizes its utilities given its parent's value
  - sends its value to its child agent C
### DPOP: VALUE PROPAGATION

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>U(A,C)</th>
<th>U(B,C)</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- Agent C
  - chooses its value that maximizes its utilities given its parent and pseudo-parent's value
### DPOP: VALUE PROPAGATION

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>U(A,C)</th>
<th>U(B,C)</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Agent C**
  - chooses its value that maximizes its utilities given its parent and pseudo-parent's value
- each agent knows its own value in the optimal solution
DPOP

• Properties:
  • Requires a linear number of messages
  • UTIL messages can be exponentially large

• Common Extensions:
  • MB-DPOP: Bounds the amount of memory used but requires more messages [Adrian Petcu, Boi Faltings: MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization. IJCAI 2007: 1452-1457]
  • A-DPOP: Speeds up DPOP by sacrificing optimality [Adrian Petcu, Boi Faltings: Approximations in Distributed Optimization. CP 2005: 802-806]
  • DPOP + Function Filtering: Speeds up DPOP by propagating bounds [Ismel Brito, Pedro Meseguer: Improving DPOP with function filtering. AAMAS 2010: 141-148]
EXTENSIONS + FUTURE DIRECTIONS

• Preserving Privacy:
  • A main motivation for distributed approaches is privacy.
• ...

A Privacy-Preserving Algorithm for Distributed Constraint Optimization by Tal Grinshpoun, Tamir Tassa
EXTENSIONS + FUTURE DIRECTIONS

• Dynamic DCOPs:
  • Real-world coordination problems occur in dynamic environments
  • …

A New Analysis Method for Dynamic, Distributed Constraint Satisfaction by Roger Mailler, Huimin Zheng
EXTENSIONS + FUTURE DIRECTIONS

• Applying DCOPs to Real-World Applications:
  • Sensor Networks [Ruben Stranders, Alessandro Farinelli, Alex Rogers, Nicholas R. Jennings: Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm. IJCAI 2009: 299-304]
  • Smart Grids [Sam Miller, Sarvapali D. Ramchurn, Alex Rogers: Optimal decentralised dispatch of embedded generation in the smart grid. AAMAS 2012: 281-288]
  • Logistic Operations [Thomas Léauté, Boi Faltings: Coordinating Logistics Operations with Privacy Guarantees. IJCAI 2011: 2482-2487]
  • ...

Explorative Max-sum for Teams of Mobile Sensing Agents
by Harel Yedidsion, Roie Zivan, Alessandro Farinelli

Exploiting Max-Sum for the Decentralized Assembly of High-Valued Supply Chains
by Toni Penya-Alba, Meritxell Vinyals, Jesus Cerquides, Juan Antonio Rodriguez-Aguilar
EXTENSIONS + FUTURE DIRECTIONS

• Combining search and inference approaches:
  • Search approaches typically require a small amount of memory but a large amount of computation time.
  • Inference approaches typically require a large amount of memory but a small computation time.

• Combining ADOPT and DPOP [James Atlas, Matt Warner, Keith Decker: A Memory Bounded Hybrid Approach to Distributed Constraint Optimization, DCR 2008]


• ...
GOOD READS

• **AI Magazine Article: Overview of DCOPs and DCSPs** [William Yeoh, Makoto Yokoo: Distributed Problem Solving. AI Magazine 33(3): 53-65 (2012)]

• **Book Chapter: Overview of DCOPs and DCSPs** [Alessandro Farinelli, Meritxell Vinyals, Alex Rogers, Nicholas Jennings: Chapter: Distributed Search and Constraint Handling. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, 2nd Edition. Editor: Gerhard Weiss, (2013)]


• **Dissertation: DPOP** [Adrian Petcu: A Class of Algorithms for Distributed Constraint Optimization]

• **Dissertation: Max-Sum** [Ruben Stranders: Decentralised Coordination of Information Gathering Agents]

• **Dissertation: Action-GDL** [Meritxell Vinyals: Exploiting the structure of Distributed Constraint Optimization Problems to assess and bound coordinated actions in Multi-Agent Systems]
OPEN-SOURCE SIMULATOR PLATFORMS

- DCOPolis [http://www.dcopolis.org/]

- FRODO [http://frodo2.sourceforge.net/]

- DisChoco [http://dischoco.sourceforge.net/]

- Agent Zero [https://code.google.com/p/azapi-test/]
OptMas–DCR tutorial 2014, Part II:
Game theoretic Approaches to
Distributed Optimisation

Archie Chapman
Centre for Future Energy Networks
School of Electrical and Information Engineering
University of Sydney

5 May 2014, Paris, France
Introduction

In this presentation, I will go through game-theoretic models and results that can be applied to problems of distributed optimisation. I will focus on non-cooperative game models — cooperative games are used in a different context.
Game Theory

Game theory is the study of strategic interaction between self-interested entities.

So what does that mean?

It is the study of how an individual “agent”

▶ choses between the actions, strategies and policies available to it,
▶ given its preferences and the information available to it,

in the context of other agents doing the same.

We will look at how these agents can be designed to compute solutions to optimisation problems in a distributed fashion.
Basics — Game theoretic terminology

\( i \in N \quad \text{agents, players, participants} \ldots \)

\{ a^i \in A^i \}_{i \in N} \quad \text{actions, strategies, policies} \ldots

\text{a} \in A = \prod_{i \in N} A^i \quad \text{joint action, action profile, outcome} \ldots

\{ u^i(\text{a}) : i \in N \} \quad u^i : A \to \mathbb{R}, \text{utility, preference, payoff, incentive} \ldots
The agent design problem

The problem:
Design a team of agents that collectively compute a solution to an optimisation problem.

Assumption:
An agent makes choices to maximise its own utility function $u^i(a)$.

The problem, again:
Design utility functions for a team of self-interested agents so that they collectively compute a solution to an optimisation problem.
Team games

In a team game, all agents have the same utility for every outcome; that is: \( u^i(a) = u^j(a) \) for all \( i, j \in N \)

E.g.:

\[
\begin{array}{ccc}
   & a_1^2 & a_2^2 \\
 a_1^1 & (0,0) & (2,2) \\
 a_2^1 & (3,3) & (1,1) \\
\end{array}
\]

Definition: Player \( i \)'s best response set is:

\[
BR_i(a^{-i}) = \{ a^i \in A^i : a^i = \arg\max_{\tilde{a}^i \in A^i} u^i(\tilde{a}^i, a^{-i}) \}
\]

A pure Nash equilibrium is an outcome satisfying:

\[
a^* \text{ such that } a^{i^*} \in BR^i(a^{-i^*}), \forall i \in N.
\]
Potential games (congestion games)

**Definition:** A function $\phi : A \rightarrow \mathbb{R}$ is a **potential** for a game if $\forall i \in I, \forall a^i, \tilde{a}^i \in A^i$, and $\forall a^{-i} \in A^{-i}$:

$$\phi(a^i, a^{-i}) - \phi(\tilde{a}^i, a^{-i}) = u^i(a^i, a^{-i}) - u^i(\tilde{a}^i, a^{-i}),$$

and a game that admits such a function is a **potential game**.

All team games are potential games, with $\phi(a) = u^i(a)$.

E.g.: Stag hunt game (J.-J.Rousseau, 1755):

<table>
<thead>
<tr>
<th></th>
<th>Stag</th>
<th>Hare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stag</td>
<td>(2,2)</td>
<td>(0,1)</td>
</tr>
<tr>
<td>Hare</td>
<td>(1,0)</td>
<td>(3,3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Stag</th>
<th>Hare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stag</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Hare</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Potential games

**Lemma**: Local maxima of a potential function are all pure Nash equilibria of the potential game.

**Theorem**: All finite potential games have a pure Nash equilibrium.  
*Proof*: $\phi$ is single-valued function on a finite space $\therefore$ have a maximal value $\equiv$ pure Nash equilibrium [Monderer and Shapley, 1996].

**Connection to optimisation**: If $\phi$ is an objective function, partition $a$ and construct $u^i(\cdot)$ so that the agents play a potential game, then the system can be used as a distributed method to find local maxima.

(See: Tumer & Wolpert, 1999; Marden, Arslan & Shamma 2009; Chapman, 2009; and more recent work by Jason Marden, et al.)
Potential games — A how-to-build guide.

Given an objective function, to derive a potential game between the agents, give each agent an exclusive subset of variables, and:

**Method 1:** Set all agents’ utilities are the objective function (this is a team game),

**Method 2:** Scale each agent’s utility to a *reference action*,

**Method 3:** Compute each agent’s *marginal contribution* to the target function,

**Method 4:** Decouple “unconnected” agents from one another, in order to exploit sparse graphical structure...
DCOP games and potential games.

Min conflicts graph colouring: 3 nodes, two actions (B&W), two constraints. Aim: minimise number of conflicts.

\[ f^1 = f^2 = \begin{cases} 1 & \text{if arguments } \neq \\ 0 & \text{if arguments } = \end{cases} \]

\[ u^i(a) : \]

| \( s_C \) | \( s_A, s_B \) |
|-------|-------|-------|-------|-------|
| B, B  | (0, 0, 0) | (0, 1, 1) | (1, 0, 1) | (1, 1, 2) |
| B, W  | (1, 1, 2) | (1, 0, 1) | (0, 1, 1) | (0, 0, 0) |
| W, B  |       |       |       |       |
| W, W  |       |       |       |       |

\[ \phi(a) : \]

| \( s_C \) | \( s_A, s_B \) |
|-------|-------|-------|-------|-------|
| B, B  | 0     | 1     | 1     | 2     |
| B, W  | 2     | 1     | 1     | 0     |
| W, B  |       |       |       |       |
| W, W  |       |       |       |       |
Congestion games (= potential games)

Let $G$ be a graph, with edges $M$ that have congestion costs \{ $c_j(\#_j) : j \in M$ \}, where $\#_j$ is the number of users of edge $j$.

Agent $i \in N$ wants to move from one vertex to another using $G$, so
- $A^i \subseteq \mathcal{P}(M)$, and
- $u^i = \sum_{j \in a^i} c_j(\#_j(a))$.

A potential for this game is: $\phi(a) = \sum_{j \in \bigcup_{i=1}^{N} a^i} \sum_{k=1}^{\#_j(a)} c_j(k)$

These are widely used in resource allocation, congestion management, load balancing, scheduling and queueing problems, e.g. target allocation for tracking: Arslan, Marden & Shamma (2006).
Two relevant measures of the solutions efficiency for multi-agent allocation problems are:

- the *price of stability*, and
- the (pure) *price of anarchy*

**Price of stability**: the ratio of the best NE to the optimum.

**Price of anarchy**: the ratio of the worst-case pure NE to the optimum.
Congestion games — quality bounds

Definition: A set function \( f : 2^{\mathbb{Z}} \rightarrow \mathbb{R} \) is submodular if:

\[
 f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y), \quad \forall \; X, Y \subseteq \mathbb{Z},
\]

and is non-decreasing if:

\[
 f(X) \leq f(Y), \quad \forall \; X \subseteq Y \subseteq \mathbb{Z}.
\]

Theorem: If the objective is a non-decreasing submodular function, and agents’ utilities are their marginal contribution to \( f(X) \), then:

- Price of stability is 1,
- Price of anarchy is 1/2.

That is, every NE solution is at least half as good as the global optimum (Vetta, 2002; Marden and Wierman, 2008).
Potential games — Distributed algorithms for selfish agents

Many different algorithms converge to NE in potential games, but all have same structure. Agents share their current action, then compute the following:

State Evaluation: Each algorithm has a target function that it uses to evaluate its prospective variable values.

Decision Rule: The decision rule refers to the procedure by which an agent uses its evaluation of values to decide upon an action to take.

Adjustment Schedule: In many the scheduling mechanism is implicitly random. while some algorithms are identified by their use of specific adjustment schedules that allow for preferential adjustment or parallel execution.

Examples: Best-reply, dist. stochastic algorithm, fictitious play, regret matching, MGM... you can mix-and-match to come up with your own.
Potential games — Distributed algorithms

Example: Distributed Stochastic Algorithm vs Fictitious Play

<table>
<thead>
<tr>
<th></th>
<th>Dist. Stoch. Alg.</th>
<th>Fictitious Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation:</td>
<td>Expected reward wrt previous joint action</td>
<td>Expected reward wrt average joint action</td>
</tr>
<tr>
<td>Decision:</td>
<td>Argmax (Exp. reward)</td>
<td>Argmax (Exp. reward)</td>
</tr>
<tr>
<td>Schedule:</td>
<td>$p$-Flood</td>
<td>Flood</td>
</tr>
</tbody>
</table>

Both DSA and FP are effectively hill-climbing algorithms, but their convergence to NE is proved in different ways.

However, DSA is part of a larger class of algorithms with *finite* memory, which are all subject to a general convergence result.
Infinite action sets — Convex games

Continuous actions means infinitely many joint actions. Actions may also be multi-valued, i.e. vectors.

How to guarantee pure Nash equilibrium exists?
— Convex games:
  - All action spaces $A^i$ are closed and convex, and
  - all utility functions $u^i$ are continuous and concave

Theorem: Convex games have a pure Nash equilibrium (Rosen 1965). Can be solved using (sequential, asynchronous) best reply.
Application of convex games — load scheduling

Mohsenian-Rad, Wong, et al. (2010),

- Divide a day into 24 one-hour slots $H$.
- $A^i$ is a set of feasible allocations of energy use across $H$. Total energy use is fixed, so this is a convex set.
- $u^i$ is the average cost of generating a unit of energy multiplied by the amount of energy $i$ uses in a day. If the generation cost is quadratic, then every $u^i$ is concave.

The unique Nash equilibrium minimises the total generation cost, subject to the agents’ energy use constraints. Solved using a best-reply algorithm.
Infinite action sets — infinite potential games

A function $\phi : A \to \mathbb{R}$ is a potential for an infinite game if
\[ \forall i \in I, \forall a^i, \tilde{a}^i \in A^i, \text{ and } \forall a^{-i} \in A^{-i} : \]
\[ \frac{\partial \phi(a^i, a^{-i})}{\partial a^i} = \frac{\partial u^i(a^i, a^{-i})}{\partial a^i} \]

and a game that admits such a function is a infinite potential game.

**Theorem:** All infinite potential games with closed and convex action spaces have a pure Nash equilibrium [Monderer and Shapley 1996].

Developed for analysis of non-atomic traffic flow (Wardrop equilibrium).

Applied to electrical demand side management problems:
— Ma, Callaway and Hiskens (2013), Plug-in electric vehicle charging.
— Atzeni, Ordonez, et al. (2012), a demand response mechanism.
Extensions: Unknown utility functions

Shameless self promotion: Can extend all of these algorithms to settings where the utility functions are initially unknown, are are estimated on-line from noisy samples using reinforcement learning.

- Use Q–learning within each agent’s reward structure.
- Decaying rate of independent random sampling to learn utility function, i.e. greedy-in-the-limit-with-infinite-exploration (Borel–Cantelli).
- Sufficient conditions for almost-sure convergence of play to NE and of reward estimates to true mean values.

(See: Chapman, Leslie, Rogers and Jennings, 2013a, 2013b, for theoretical results and an application to sensor networks, resp.).
Extensions: Dynamic and stochastic games

So far, we have considered only static games. The models can be extended to allow for (stochastic) state transitions:

- Action spaces and utility functions are made state-dependent.
- Can construct dynamic programming-based values of states to evaluate current action choices.

Team and potential game analysis can be applied to both exogenous and endogenous state-transition cases:

- Potters et al., (2009) for team games,
Good reads


Other references:

- Vetta (2002), Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions, *FOCS 2002*.
Part III: Market-based resource allocation and coalition formation
What you will learn

- A high-level overview of auction mechanisms
  - The Winner Determination Problem (WDP)
  - The complexity of the WDP in different classes of auction
  - DCOP algorithms for the WDP
- A high-level overview of coalition formation
  - The Coalition Structure Generation (CSG) problem
  - The equivalence between the WDP and CSG
  - A compact representation for CSG based on DCOP
- Pointers to relevant literature
Market-based resource allocation
- The Winner Determination Problem (WDP)
- WDP in Canonical auction families
- WDP in Combinatorial auctions
- WDP in Supply Chain formation auctions

Coalition Formation
- Coalition Structure Generation (CSG)
- Compact representations
Market-based resource allocation

From all the mechanisms to solve multi-agent resource allocation problems, we restrict our focus to **auctions**.

- **Auctions** are any mechanisms for **allocating resources among self-interested agents**
- An **auction** is a **protocol** that allows agents to **indicate** their **interests** in one or more **resources** and that uses these indications of interest to **determine** both an **allocation of resources** and a **set of payments**.
- Very **widely used**: government sale of resources, eBay, stock market . . .
Market-based resource allocation

- increasing importance of studying **distributed systems** with **heterogeneous agents**
- markets for:
  - computational resources
  - P2P systems
  - network bandwidth
- currency needn’t be real money, just something **scarce**
  - that said, real money trading agents are also an important motivation
Auctions generic protocol

All auctions follow a **common pattern** typically divided into four sequential steps:

- **Bid call.** The auctioneer broadcasts a call for bids to serve customer(s) and declare when the auction will close.

- **Bid collection.** Bidders transmit their bids to the auctioneer, and the auctioneer collects the bids.

- **Winner determination.** Upon collection of all bids, the auctioneer determines, depending on the auction mechanism, what gets each customer (allocation)

- **Clearing.** The auctioneer informs the depots about the resulting allocation.
The Winner Determination Problem (WDP)

*The goal of an auction is to allocate the goods to those who value them most.*

We focus on the *optimization problem* faced by an auctioneer that must identify the *winner(s)* of an auction

▸ **Winner Determination Problem (WDP):** find an *allocation* of resources to bidders that *maximizes* the *auctioneer’s revenue.*

We do not discuss the concern that bidders in an auction might not *report* their *true valuations* because of strategic considerations.

▸ Bidders are assumed to bid truthfully.
Some Canonical Auctions

English Auction

- auctioneer starts the bidding at some “reservation price”
- bidders then shout out ascending prices
- once bidders stop shouting, the high bidder gets the good at that price

Japanese Auction

- An English auction except that the auctioneer calls out the prices
- all bidders start out standing
- when the price reaches a level that a bidder is not willing to pay, that bidder sits down
- the last person standing gets the good
Dutch Auction

- the auctioneer starts a clock at some high value; it descends
- at some point, a bidder shouts “mine!” and gets the good at the price shown on the clock

Sealed-Bid First-Price Auction

- bidders write down bids on pieces of paper
- auctioneer awards the good to the bidder with the highest bid
- that bidder pays the amount of his bid
Computational complexity of WDP in single-unit

In all these (single-unit) auctions solving the WDP is trivial:

- **English**: Last bid wins
- **Japanese**: Last remaining bidder wins
- **Dutch**: First bid wins
- **Sealed-bid**: Highest bid wins
Computational complexity of WDP in multi-unit

- It exists **multi-unit** versions of the single-unit
  - English, Japanese, Dutch, Sealed-bids
- Bidders bid on **quantity** $q_i$ and price $p_i$
- WDP requires solving a **weighted knapsack problem**
  - **NP-Hard** but **good approximation schemes**
Computational complexity of WDP in multi-unit

Example

- An auctioneer wants to sell 15 limes maximizing the revenue
- The auctioneer receives the following buying offers:
  - A: 12 limes for 4 €
  - B: 2 limes for 2 €
  - C: 1 lime for 1 €
  - D: 1 lime for 1 €
  - E: 4 limes for 10 €
- and solves:

$$\begin{align*}
\text{max } & 4 \cdot x_A + 4 \cdot x_B + 4 \cdot x_C + 4 \cdot x_D + 4 \cdot x_E \\
\text{subject to } & 12 \cdot x_A + 2 \cdot x_B + x_C + x_D + 4 \cdot x_E \leq 15 \\
& x_i \in \{0, 1\} \quad \forall i \in \{A, B, C, D, E\}
\end{align*}$$
Valuations for heterogeneous goods

- now consider **multiple, heterogeneous goods** are being sold.
- bidders could have in this case **different sorts of evaluations**:
  - **complementarity**: for sets $S$ and $T$, $v(S \cup T) > v(S) + v(T)$
    - e.g., a left shoe and a right shoe
  - **substitutability**: $v(S \cup T) < v(S) + v(T)$
    - e.g., two tickets to different movies playing at the same time
- **substitutability** is relatively **easy to deal** with
  - e.g., just sell the goods sequentially, or allow bid withdrawal
- **complementarity** is **trickier**...
Combinatorial Auctions (CA’s)

Complementarity requires Combinatorial Auctions (CA’s) where:
- different goods are traded simultaneously
- bidders express their preferences not just for particular goods but for sets of bundles of goods
Complexity of WDP for CA’s

Let there be \( n \) goods . . .

- The number of possible bids \( m \) is exponential to the number of goods \((2^n)\)
- The WDP for a CA is a NP-complete problem with respect to the number of goods

\[
\max \sum_{i=1}^{m} x_ip_i \\
\text{subject to } \sum_{i|g \in S_i} x_i \leq 1 \quad \forall g \\
x_i \in \{0, 1\} \quad \forall i
\]

(Equivalent to the weighted set packing problem)
Winner determination problem for CAs

How do we deal with the computational complexity of the WDP?

- Use heuristic methods to solve the problem.
  - This works pretty well in practice, making it possible to solve WDPs with many hundreds of goods and thousands of bids.
- Require bids to come from a restricted set, guaranteeing that the WDP can be solved in polynomial time.
  - Problem: these restricted sets are very restricted...
Winner determination problem for CAs

How do we deal with the computational complexity of the WDP?

- Use heuristic methods to solve the problem
  - this works pretty well in practice, making it possible to solve WDPs with many hundreds of goods and thousands of bids.
- Require bids to come from a restricted set, guaranteeing that the WDP can be solved in polynomial time
  - problem: these restricted sets are very restricted...
Winner determination problem for CAs

How do we deal with the computational complexity of the WDP?

- Use **heuristic methods** to solve the problem
  - this works pretty well in practice, making it possible to solve WDPs with many hundreds of goods and thousands of bids.
- Require **bids** to come from a restricted set, guaranteeing that the WDP can be solved in polynomial time
  - problem: these restricted sets are very restricted...
Tractable CA instances: total ordering

It exists a total ordering among the goods

- The most interesting combinations would be contiguous, in the bidders eyes.

Example

- Objects to be auctioned: parcels of land along a shore line
  - Shore line is important: it imposes a linear order on the parcels
  - Make a restriction to bid only on contiguous parcels
Tractable CA instances: total ordering

It exists a total ordering among the goods

- The most interesting combinations would be contiguous, in the bidders eyes.

Example

- Objects to be auctioned: parcels of land along a shore line
  - Shore line is important: it imposes a linear order on the parcels
  - Make a restriction to bid only on contiguous parcels
Tractable CA instances: total ordering

It exists a total ordering among the goods

- The most interesting combinations would be contiguous, in the bidders eyes.

Example

- Objects to be auctioned: parcels of land along a shore line
  - Shore line is important: it imposes a linear order on the parcels
  - Make a restriction to bid only on contiguous parcels

Two computational consequences:

- Number of distinct bids would be limited by a polynomial in the number of objects
- WDP can be solved in polynomial time.
Tractable CA instances: circular ordering

Tractability results also hold in circular orderings

Two computational consequences:

- Number of **distinct bids** would be **limited** by a polynomial in the number of objects
- **WDP** can be **solved** in **polynomial time**.
Supply chain formation

- A supply chain is a set of organizations directly linked by the flow of products, services, finances, and information from a source to a customer.

- Supply Chain formation (SCF) is the process of determining the participants in the supply chain, who will exchange what with whom, and the term of the exchanges.
Supply chain formation

- A **supply chain** is a set of organizations directly linked by the flow of products, services, finances, and information from a source to a customer.

- **Supply Chain formation** (SCF) is the process of determining the participants in the supply chain, who will exchange what with whom, and the term of the exchanges.

- The concept of buyer or seller is generalised to a participant.
- Each participant agent $a_i$ is represented as a set of input goods, a set of output goods and a price: $(I_i, O_i, p_i)$.
- The price (which can be negative or positive) is the value required by the participant to produce the output goods given the input goods.
Supply chain formation model Walsh and Wellman 2011
Supply chain formation model Walsh and Wellman 2011

Optimal solution

\[ v(\{\text{Alice, Dave, Frank}\}) = -5 + (-10) + 22 = 7 \]
Decentralised SCF Winsper and Chli 2013

[Winsper and Chli 2013, “Decentralized Supply Chain Formation using Max-Sum Loopy Belief Propagation”] proposed to solve the SCF problem in a decentralised way by:

✓ Mapping the SCF problem to a DCOP
✓ Solving distributedly using Max-Sum
✓ Finally post-process the solution to turn the result into a feasible supply chain (value propagation).
The problem representation employed by Winsper and Chli 2013 leads to **exponential memory and communication requirements**

[Penya-Alba et al. 2012, “A Scalable Message-Passing Algorithm for Supply Chain Formation”] proposes an alternative mapping to DCOP that dramatically lowers max-sum requirements for SCF

- Experiments solving SCF problems with thousands of participants!!!
Mixed Multi Unit Combinatorial Auction (MMUCA)

- SCF may require the inclusion of directed cycles

- Captured by Mixed Multi Unit Combinatorial Auctions (Giovannucci et al. 2007)

- The WDP for MMUCA is more complex since a solution to the SC is not a set of participants but a sequence of participants (Fionda and Greco 2013)
References of good introductions (and for which I reused some examples) on market-based resource allocation:

- [Vries and Vohra 2003, “Combinatorial Auctions: A Survey”]
- [Leyton-Brown and Shoham 2013, “Mechanism Design and Auctions”]
- [Cerquides et al. 2013, “A Tutorial on Optimization for Multi-Agent Systems”]
Outline

Market-based resource allocation
  The Winner Determination Problem (WDP)
  WDP in Canonical auction families
  WDP in Combinatorial auctions
  WDP in Supply Chain formation auctions

Coalition Formation
  Coalition Structure Generation (CSG)
  Compact representations
A definition of coalition formation

- The coming together of a set of (self-interested?) agents that cooperate to perform joint actions where the reward from performing such actions is attributed to the group.
We focus on the **Coalition Structure Generation** problem:

*The goal is to **partition the agents** into groups such that the **sum of the coalition values** is maximized.*
Coalition Structure Generation (CSG)

Example: given 3 agents, the possible coalitions are:
\{ a_1 \}, \{ a_2 \}, \{ a_3 \}, \{ a_1, a_2 \}, \{ a_1, a_3 \}, \{ a_2, a_3 \}, \{ a_1, a_2, a_3 \}  

The possible coalition structures are:
\{\{ a_1 \}, \{ a_2 \}, \{ a_3 \}\},  \{\{ a_1, a_2 \}, \{ a_3 \}\},  \{\{ a_2 \}, \{ a_1, a_3 \}\},  
\{\{ a_1 \}, \{ a_2, a_3 \}\},  \{\{ a_1, a_2, a_3 \}\}

The input is the characteristic function:
\begin{align*}
v(\{ a_1 \}) &= 20 \\
v(\{ a_2 \}) &= 40 \\
v(\{ a_3 \}) &= 30 \\
v(\{ a_1, a_2 \}) &= 70 \\
v(\{ a_1, a_3 \}) &= 40 \\
v(\{ a_2, a_3 \}) &= 65 \\
v(\{ a_1, a_2, a_3 \}) &= 95
\end{align*}

What we want as output is an optimal coalition structure in which the sum of the values is maximized:
\begin{align*}
V(\{\{ a_1 \}, \{ a_2 \}, \{ a_3 \}\}) &= 20 + 40 + 30 = 90 \\
V(\{\{ a_1, a_2 \}, \{ a_3 \}\}) &= 70 + 30 = 100 \\
V(\{\{ a_2 \}, \{ a_1, a_3 \}\}) &= 40 + 40 = 80 \\
V(\{\{ a_1 \}, \{ a_2, a_3 \}\}) &= 20 + 65 = 85 \\
V(\{\{ a_1, a_2, a_3 \}\}) &= 95
\end{align*}
Coalition Structure Generation (CSG)

Example: given 3 agents, the possible coalitions are:
\{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}  

The possible coalition structures are:
\{\{a_1\}, \{a_2\}, \{a_3\}\}, \{\{a_1, a_2\}, \{a_3\}\}, \{\{a_2\}, \{a_1, a_3\}\},
\{\{a_1\}, \{a_2, a_3\}\}, \{\{a_1, a_2, a_3\}\}

The input is the characteristic function:
\[v(\{a_1\}) = 20\]
\[v(\{a_2\}) = 40\]
\[v(\{a_3\}) = 30\]
\[v(\{a_1, a_2\}) = 70\]
\[v(\{a_1, a_3\}) = 40\]
\[v(\{a_2, a_3\}) = 65\]
\[v(\{a_1, a_2, a_3\}) = 95\]

What we want as output is an optimal coalition structure in which the sum of the values is maximized:
\[V(\{\{a_1\}, \{a_2\}, \{a_3\}\}) = 20 + 40 + 30 = 90\]
\[V(\{\{a_1, a_2\}, \{a_3\}\}) = 70 + 30 = 100\]
\[V(\{\{a_2\}, \{a_1, a_3\}\}) = 40 + 40 = 80\]
\[V(\{\{a_1\}, \{a_2, a_3\}\}) = 20 + 65 = 85\]
\[V(\{\{a_1, a_2, a_3\}\}) = 95\]
Coalition Structure Generation (CSG)

Example: given 3 agents, the possible coalitions are:
\{ a_1 \}, \{ a_2 \}, \{ a_3 \}, \{ a_1, a_2 \}, \{ a_1, a_3 \}, \{ a_2, a_3 \}, \{ a_1, a_2, a_3 \}
The possible coalition structures are:
\{ \{ a_1 \}, \{ a_2 \}, \{ a_3 \} \}, \{ \{ a_1, a_2 \}, \{ a_3 \} \}, \{ \{ a_2 \}, \{ a_1, a_3 \} \},
\{ \{ a_1 \}, \{ a_2, a_3 \} \}, \{ \{ a_1, a_2 \}, \{ a_3 \} \}

The input is the characteristic function:
\[ v(\{ a_1 \}) = 20 \]
\[ v(\{ a_2 \}) = 40 \]
\[ v(\{ a_3 \}) = 30 \]
\[ v(\{ a_1, a_2 \}) = 70 \]
\[ v(\{ a_1, a_3 \}) = 40 \]
\[ v(\{ a_2, a_3 \}) = 65 \]
\[ v(\{ a_1, a_2, a_3 \}) = 95 \]

What we want as output is an optimal coalition structure in which the sum of the values is maximized:
\[ V(\{ \{ a_1 \}, \{ a_2 \}, \{ a_3 \} \}) = 20 + 40 + 30 = 90 \]
\[ V(\{ \{ a_1, a_2 \}, \{ a_3 \} \}) = 70 + 30 = 100 \]
\[ V(\{ \{ a_2 \}, \{ a_1, a_3 \} \}) = 40 + 40 = 80 \]
\[ V(\{ \{ a_1 \}, \{ a_2, a_3 \} \}) = 20 + 65 = 85 \]
\[ V(\{ \{ a_1, a_2, a_3 \} \}) = 95 \]
Coalition Structure Generation (CSG)

Example: given 3 agents, the possible coalitions are:
\{ a_1 \}, \{ a_2 \}, \{ a_3 \}, \{ a_1, a_2 \}, \{ a_1, a_3 \}, \{ a_2, a_3 \}, \{ a_1, a_2, a_3 \}

The possible coalition structures are:
\{ \{ a_1 \}, \{ a_2 \}, \{ a_3 \} \}, \{ \{ a_1, a_2 \}, \{ a_3 \} \}, \{ \{ a_2 \}, \{ a_1, a_3 \} \},
\{ \{ a_1 \}, \{ a_2, a_3 \} \}, \{ \{ a_1, a_2, a_3 \} \}

The input is the characteristic function:
\begin{align*}
v(\{ a_1 \}) &= 20 \\
v(\{ a_2 \}) &= 40 \\
v(\{ a_3 \}) &= 30 \\
v(\{ a_1, a_2 \}) &= 70 \\
v(\{ a_1, a_3 \}) &= 40 \\
v(\{ a_2, a_3 \}) &= 65 \\
v(\{ a_1, a_2, a_3 \}) &= 95
\end{align*}

What we want as output is an optimal coalition structure in which the sum of the values is maximized:
\begin{align*}
V(\{ \{ a_1 \}, \{ a_2 \}, \{ a_3 \} \}) &= 20 + 40 + 30 = 90 \\
V(\{ \{ a_1, a_2 \}, \{ a_3 \} \}) &= 70 + 30 = 100 \\
V(\{ \{ a_2 \}, \{ a_1, a_3 \} \}) &= 40 + 40 = 80 \\
V(\{ \{ a_1 \}, \{ a_2, a_3 \} \}) &= 20 + 65 = 85 \\
V(\{ \{ a_1, a_2, a_3 \} \}) &= 95
\end{align*}
The CSG problem in CF is akin to the WDP in CAs by taking:

- Agents as goods
- Coalitional values as bids

(Equivalent to the weighted set packing problem)

\[
\max \sum_{i=1}^{m} x_i p_i \\
\text{subject to } \sum_{i | g \in S_i} x_i = 1 \quad \forall g \\
x_i \in \{0, 1\} \quad \forall i
\]

Any algorithm for solving the WDP in CAs can be used for CSG in CF and viceversa!!! This also means the same combinatorial problem complexity!!!
The CSG problem in CF is akin to the WDP in CAs by taking:

- Agents as goods
- Coalitional values as bids

(Equivalent to the weighted set packing problem)

$$\max \sum_{i=1}^{m} x_i p_i$$

subject to

$$\sum_{i \mid g \in S_i} x_i = 1 \quad \forall g$$

$$x_i \in \{0, 1\} \quad \forall i$$

Any algorithm for solving the WDP in CAs can be used for CSG in CF and vice versa!!! This also means the same combinatorial problem complexity!!!
The CSG problem in CF is akin to the WDP in CAs by taking:

- Agents as goods
- Coalitional values as bids

(Equivalent to the weighted set packing problem)

\[
\max \sum_{i=1}^{m} x_i p_i \\
\text{subject to } \sum_{i | g \in S_i} x_i = 1 \quad \forall g \\
x_i \in \{0, 1\} \quad \forall i
\]

Any algorithm for solving the WDP in CAs can be used for CSG in CF and vice versa!!! This also means the same combinatorial problem complexity!!!
Compact representations

But we really need to explicitly specify the value of every possible coalition?

- Solution: Compact representations
- In what follows, we briefly discuss a compact representation in terms of a DCOP
DCOP Characteristic function

[Ueda et al. 2010, “Coalition Structure Generation based on Distributed Constraint Optimization”] proposed a CSG framework where:

- Each agent \( a_i \) has a choice of actions: \( d_{i,1}, d_{i,2}, d_{i,3}, \ldots \)
- A (possible negative) reward is assigned to every combination of actions
- Every agent must choose an action to maximise the sum of rewards
DCOP compact representation

- The characteristic function is represented as one big DCOP
- Every coalition’s value is computed as the optimal solution of a DCOP among the agents of a coalition
DCOP compact representation

- The characteristic function is represented as one big DCOP
- Every coalition’s value is computed as the optimal solution of a DCOP among the agents of a coalition
But solving a DCOP is NP-Hard.

Then, computing the values of all coalitions in principle requires solving $2^n$ NP-Hard problems!!!

However, Ueda et al. 2010 shows how to reformulate the CSG problem so that an optimal coalition structure can be found by solving a single DCOP.

Two computational consequences:

✓ The input of the CSG problem can be compactly represented by means of a factor graph/DCOP.

✓ If the DCOP/factor graph is acyclic, the CSG problem can be solved in polynomial time.
But solving a DCOP is NP-Hard.

Then, computing the values of all coalitions in principle requires solving $2^n$ NP-Hard problems!!!

However, Ueda et al. 2010 shows how to reformulate the CSG problem so that an optimal coalition structure can be found by solving a single DCOP.

Two computational consequences:

- The input of the CSG problem can be compactly represented by means of a factor graph/DCOP.
- If the DCOP/factor graph is acyclic, the CSG problem can be solved in polynomial time.
But solving a DCOP is NP-Hard.

Then, computing the values of all coalitions in principle requires solving $2^n$ NP-Hard problems!!!

However, Ueda et al. 2010 shows how to reformulate the CSG problem so that an optimal coalition structure can be found by solving a single DCOP.

Two computational consequences:

- The input of the CSG problem can be compactly represented by means of a factor graph/DCOP.
- If the DCOP/factor graph is acyclic, the CSG problem can be solved in polynomial time.
To continue learning . . .

Pointers to good introductions on coalition formation:

- [Elkind, Rahwan, and Jennings 2013, “Computational Coalition Formation”]
- [Airiau 2013, “Cooperative games and multiagent systems”]
The End

Thank you for your attention!!!


Penya-Alba, Toni et al. (2012). “A Scalable Message-Passing Algorithm for Supply Chain Formation”. In: AAAI.

Ueda, Suguru et al. (2010). “Coalition Structure Generation based on Distributed Constraint Optimization”. In: AAAI.

