Distributed Constraint Optimisation Problems

Summary

- Multi-Agent Systems
- Distributed COP
- Complete solution technique: DPOP

Multi-Agent Systems

MAS

- Systems composed of multiple computational units (Agents) that can interact among them
- Agent $\rightarrow$ hard to define precisely, main features
  - Relevant degree of autonomy
  - Reactivity
  - Pro-activeness
  - Social Ability $\Rightarrow$ **Multi-Agent Systems**

Motivations

Evolution in CS

- Ubiquity
- Connectivity
- Autonomy and Delegation
- High level programming

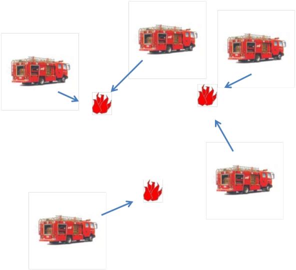
all this components favor the use of MAS technology

Applications

Applications for MAS

- Distributed Problem Solving (GRATE*, CALO, Electric Elves)
  - e-Elves `http://www.isi.edu/e-elves/index.html`
- Energy management on Smart Grids (IDEaS, ORCHID)
  - IDEaS `http://www.ideasproject.info/`
- Cooperative Information Gathering (GlacsWeb, Adaptive Energy-Aware Sensor Networks)
  - AEASN `http://www.ecs.soton.ac.uk/research/projects/AEASN`
  - demo `http://profs.sci.univr.it/~farinelli/WASWebPage/WAS-demo.html`
- E-commerce (Trading Agent Competition)
- Security (DeFACTO, ARMOR) ...

Coordination in MAS



◇ **Coordination**: choose agent's individual actions so to maximize a system-wide objective

- Individual actions: which fire to tackle
- system-wide objective: minimize total extinguish time
- solution: a **joint** action

Decentralized Coordination

◇ **Decentralized Coordination**: local decisions with local information

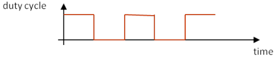
◇ Why Decentralized Coordination ?

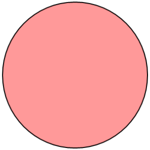
- No benefit for computation or solution quality
- But:
  - Robustness (single point of failure)
  - Scalability (bandwidth to share info)
- Decompose the problem
  - Each agent cares only of **local** neighbours

Example of Decentralized coordination: Wide area Surveillance Problem

- Sensors detect vehicles on a Road Network
- Sensors have different sensing ranges
- Roads have different traffic loads

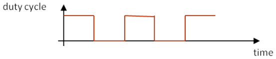
WAS: model

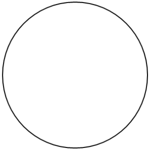




- ♢ Energy Constraints
  - Sense/Sleep modes
  - Recharge when Sleeping
  - Energy neutral operation
  - $\Rightarrow$ Constraints on duty cycle
- ♢ Sensor model
  - activity detected by single sensor $\Rightarrow$ coordination

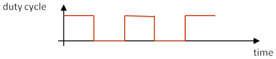
WAS: model

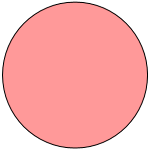




- Energy Constraints
  - Sense/Sleep modes
  - Recharge when Sleeping
  - Energy neutral operation
  - $\Rightarrow$ Constraints on duty cycle
- Sensor model
  - activity detected by single sensor $\Rightarrow$ coordination

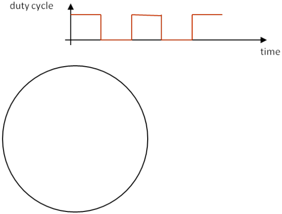
WAS: model





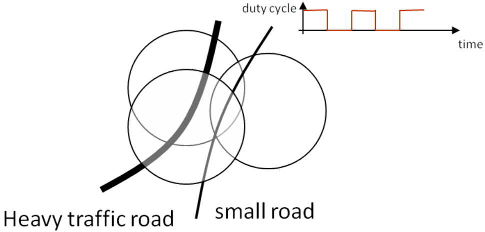
- ◇ Energy Constraints
  - Sense/Sleep modes
  - Recharge when Sleeping
  - Energy neutral operation
  - $\Rightarrow$ Constraints on duty cycle
- ◇ Sensor model
  - activity detected by single sensor $\Rightarrow$ coordination

WAS: model



- ♢ Energy Constraints
  - Sense/Sleep modes
  - Recharge when Sleeping
  - Energy neutral operation
  - $\Rightarrow$ Constraints on duty cycle
- ♢ Sensor model
  - activity detected by single sensor $\Rightarrow$ coordination

WAS: Goal

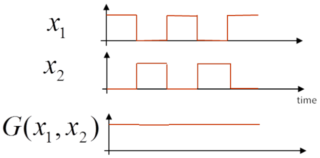


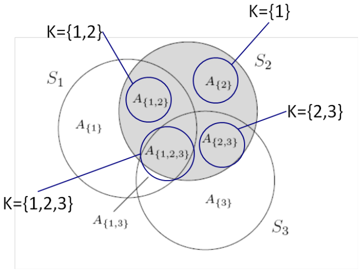
- Coordinate Sensors' duty cycles
  - Achieve Energy neutral operation
  - Minimize probability of missing vehicles

WAS: System Wide Utility

◇ Weighted Probability of event detection for each possible **joint** schedule

$$U(\vec{x}) = \sum_{\vec{k} \subset \mathcal{S}} A_{\vec{k}} \times P(detection | \lambda_d, G(\vec{x}_{\vec{k}}))$$

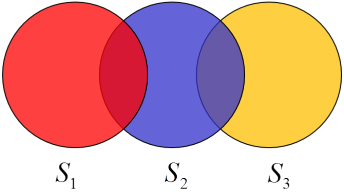




WAS: Interactions among sensors

◇ System wide utility decomposition in individual utilities (avoiding double counting), for example:

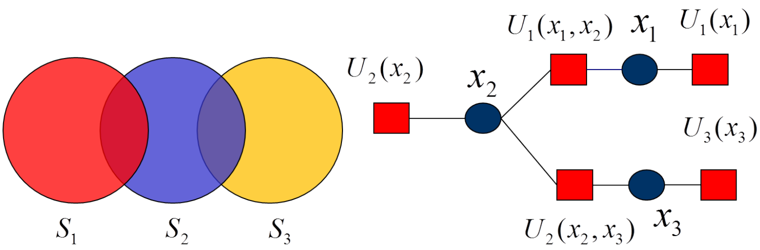
$$U(x_1, x_2, x_3) = U_1(x_1, x_2) + U_2(x_2, x_3) + U_3(x_3)$$



WAS: Factor Graph

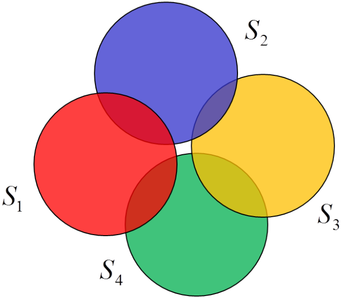
◇ Factor Graph representation

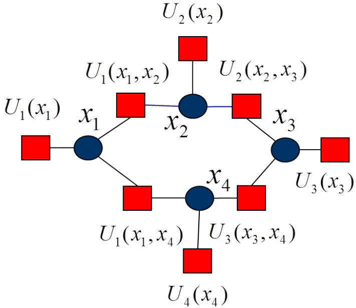
$$U(x_1, x_2, x_3) = U_1^1(x_1) + U_1^2(x_1, x_2) + U_2^1(x_2) + U_2^2(x_2, x_3) + U_3^1(x_3)$$



WAS: Loopy Factor Graph

- Tipically Graph will contains loops





Distributed COP

DCOPs

DCOP: Cost network + Agents

- DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}_h, \mathcal{C}_s \rangle$
- $\mathcal{A} = \{A_1, \ldots, A_k\}$ is a set of agents
- $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, \cdots, D_n\}$ is a set of variable domains
- $\mathcal{C}_h$ and $\mathcal{C}_s$ represent hard and soft constraints
- $\mathcal{C}_s = \mathcal{F} = \{F_1, \ldots, F_m\}$ is a set of constraint functions
- Each function $F_i : D_{i_1} \times \cdots \times D_{i_{r_i}} \rightarrow \Re$ depends on a set of variable $\mathbf{X_i} \subseteq \mathcal{X}$

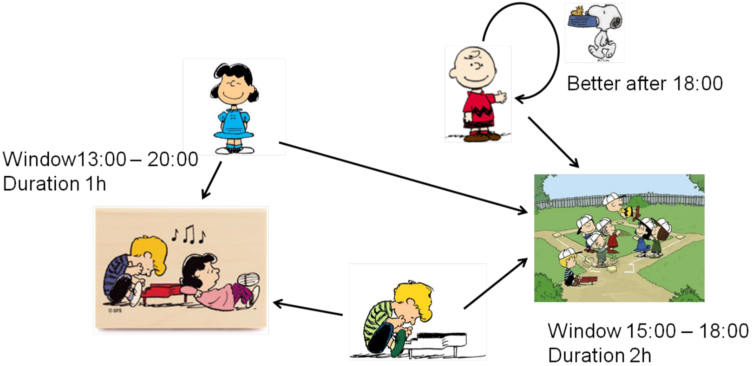
Usual Assumptions and Objectives

Assumptions and Objective

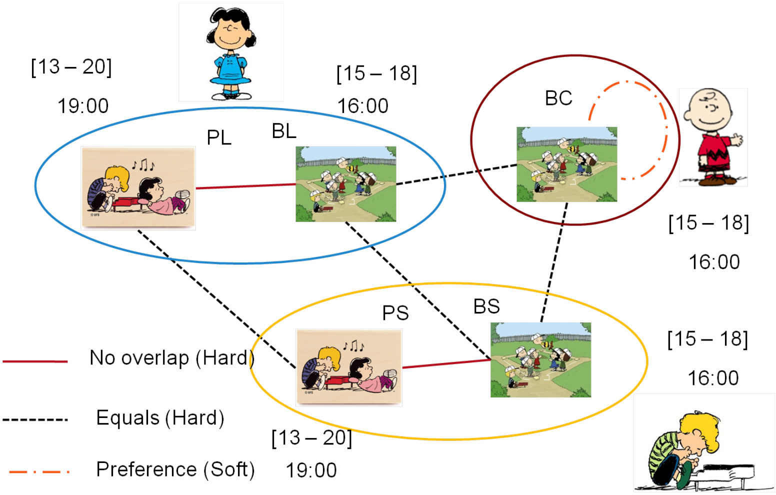
- Each variable X_i is owned by exactly one agent A_i
- An agent can potentially own more than one variable
- The agent A_i is responsible for assigning values to the variables it owns
- Objective: find the variable assignment such that all hard constraints are satisfied and the sum of all constraint functions is maximised:

$$\bar{x}^* = \arg\max_{\bar{x}} \sum_i F_i(\bar{x}_i)$$

Example of Meeting Scheduling: Problem



Example of Meeting Scheduling: DCOP



Example: Meeting Scheduling

Example (DCOP for MS)

A set of PDA agents must set up a set of meetings that PDA owners have to attend

- Agents: PDA of people that must participate to the meeting
- Variables: Meeting time (one variable for each meeting and each agent)
- Domains: slots during work hours (e.g. 8am,...,4pm)
- Constraints: hard and soft
  - Equality between meeting variables that represent same meeting across agents (Hard Constraint)
  - Inequality between meeting variables that represent different meetings within one agent (Hard Constraint)
  - Preference that people have on meeting time (Soft Constraint)

Evaluating DCOP solution techniques

Measures

- Solution quality
- Optimality guarantees
- Coordination overhead
  - Amount of computation each agent execute
  - Number of messages
  - Message size

Solution Techniques for DCOPs

Solution Techniques

- Complete approaches
  - Guarantee to provide optimal solution
  - Exponential coordination overhead
  - ADOPT, DPOP, OptAPO
- Approximate approaches
  - Low coordination overhead
  - No guarantees on optimality
  - DSA, MGM, Max-Sum

Complete Solution Techniques

Solution Techniques

- ADOPT
  - Distributed branch and bound (Search)
  - Partial order based on a DFS search
  - Asynchronous, optimality guarantees
  - Number of messages exponential in the DFS tree height
- OptAPO
  - Based on mediator agents that compute solutions for part of the problem
  - Low communication overhead (size, number)
  - Computation of mediator agents grow exponentially with the size of their partial problem

Dynamic Programming vs. Search

DP vs. Search in MAS

- Search:
  - linear size messages
  - message number is exponential (number of agents)
- Dynamic Programming:
  - linear number of messages
  - message size is exponential (width of DFS tree)
- Usually width is smaller than depth (specially for sparse problems)
- Messages can have large overhead (packet, e-mail, etc.)

Dynamic Programming Optimisation Protocol

DPOP

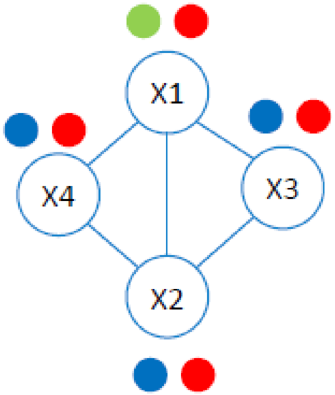
- Distributed
- Dynamic Programming
- Complete (Optimality guarantee)
- Three Phases:
  - Pseudo-tree building with a DFS
  - Utility messages from leaves to root (Util propagation)
  - Value messages from root to leaves (Value propagation)
- Each phase: linear number of message
- Util propagation phase produces messages of exponential size

DPOP running Example

<u>Value of Each Constraint</u>

same color -1

different colors 0



Pseudotrees: basic concepts

Pseudotree arrangement of a graph G

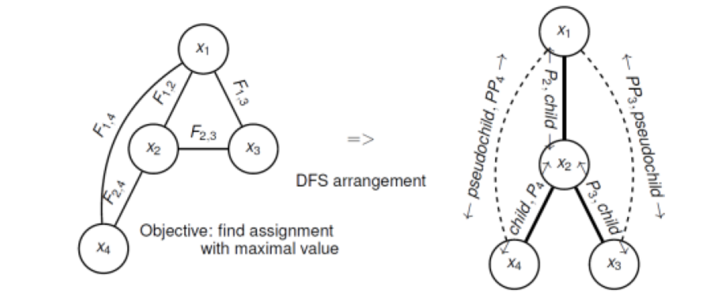
1. A rooted tree with same node as G
2. Adjacent nodes in G falls in the same branch of the Pseudotree

Thanks to 2 once a subset of nodes (separator) are instantiated different subtrees are completely independent

- Tree edges: form a spanning tree of the original graph
- Back edges: represent constraints that are not part of the spanning tree

Example: Pseudotree

Example (Pseudotree)



DFS arrangement and Pseudotrees

DFS and Pseudotree

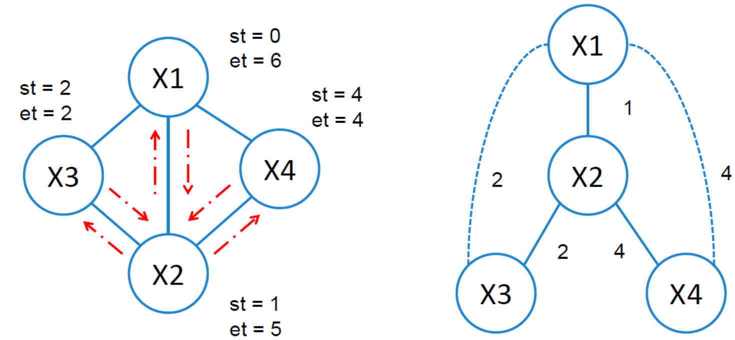
- DFS traversal of a graph generates a pseudotree
- DFS trees are subclass of Pseudotree
- Using DFS trees only neighbouring agents need to communicate
- DFS trees can be easily built using distributed algorithm

DFS traversal and psudotree building

DFS traversal

- Traverse the graph using a recursive procedure.
- Each time we reach a node X_i from a node X_j we mark X_i as visited and establish a parent/child relationship between X_j and X_i
  - $P_i = X_j$ and $C_j = C_j \cup X_i$
- When a node X_i has a visited neighbour X_j which is not its parent we establish a pseudo-parent/pseudo-child relationship between X_j and X_i
  - $PP_i = PP_i \cup X_j$ and $PC_j = PC_j \cup X_i$

Example (Pseudo tree with DFS traversal)



token movement

st: first time node received the token

et: last time node sent the token

time++ = each time token moves

Basic concepts for DFS trees

basic concepts

- Children C_i/ Parent P_i of node X_i: descendants / ancestor of X_i through tree edges
- Pseudo-Children PC_i / Pseudo-Parents PP_i of X_i: descendants / ancestor of X_i through back edges
- Sep_i separator of node X_i: all ancestors (though tree and back edges) which are connected with X_i and with any descendant of X_i
- Sep_i minimal set of ancestors that, if removed, completely disconnects the subtree rooted at node X_i from the rest of the problem
- $Sep_i = \cup_{X_j \in C_i} Sep_j \cup P_i \cup PP_i \setminus X_i$

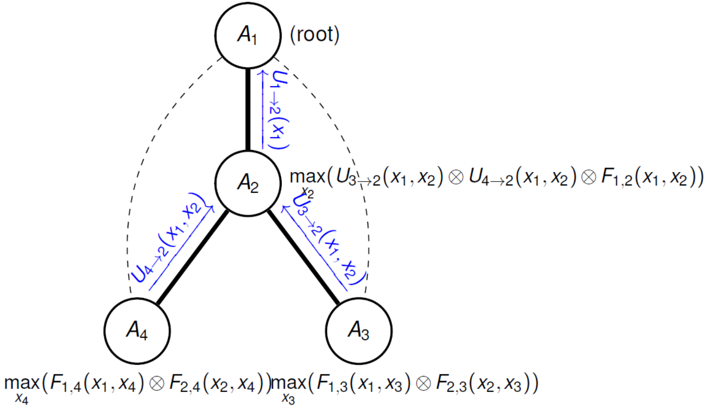
DPOP: Util propagation

Util Propagation

- Start from leaves and goes up the tree
- Each agent computes messages for its parent based on messages received from children and relevant constraints.
- Agent A_i controlling variable X_i with children C_i parent $P_i = X_j$ and pseudoparents PP_i
- $M_{i \to j}(Sep_i) = \max_{X_i}(\sum_{X_k \in C_i} M_{k \to i} + \sum_{X_p \in P_i \cup PP_i} F_i^p)$
- Each message projects out X_i (by maximisation) and aggregates (by summation) functions received from children and all constraints with ancestors (parents and pseudoparents)
- The size and computation of each message is exponential in the size of the separator

Example: Util propagation

Example (message computation for util propagation phase)



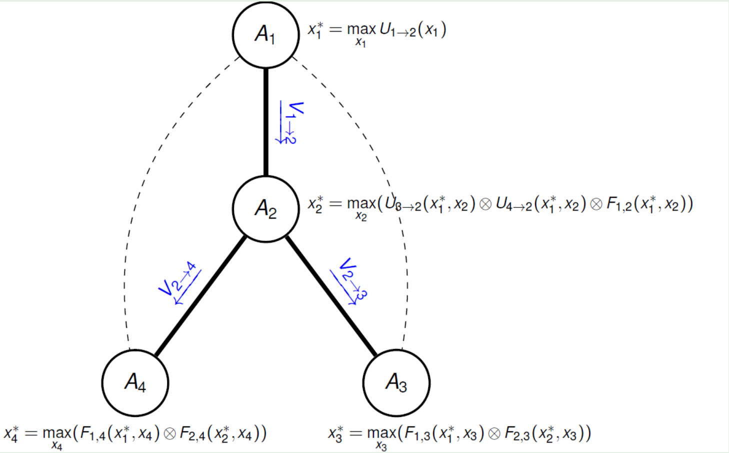
DPOP: Value propagation

Value Propagation

- Proceeds from root to leaves
- Root agent A_r computes x_r^* which is the argument that maximises the sum of messages received by all children (plus all unary relations it is involved in).
- It sends a message $V_{r \to c} = \{X_r = x_r^*\}$ containing this value to all children C_r
- The generic agent A_i computes $x_i^* = \arg\max_{X_i}(\sum_{X_k \in C_i} M_{k \to i}[\bar{x}_p^*] + \sum_{X_p \in P_i \cup PP_i} F_i^p(X_i, \bar{x}_p^*))$, where x_p^* are the optimal values received from the parent.
- The generic agent A_i sends a message to each child A_j $V_{i \to j} = \{X_s = x_s^*\} \cup X_i = x_i^*$, where $X_s \in Sep_i \cap Sep_j$

Example: Value propagation

Example (message computation for value propagation phase)



Pseudotree and induced width

Separator size and Induced Width

The induced width of a graph G along a given DFS arrangement equals the size of the largest separator of any node in the DFS arrangement

- ordering o orders of the DFS traversal
- process the nodes in reverse connecting all ancestors of each node
- width of a node: number of induced ancestors
- recursively connecting ancestors $\Rightarrow$ propagating parents and pseudoparents
- the number of induced ancestors is exactly the size of the separator

Bucket Elimination and DPOP

BE and DPOP

- Util phase of DPOP performs the same computation as BE when using the depth first order related to the DFS tree
- Depth first order related to the DFS tree: linear sequence of nodes visited by the DFS
- DPOP computes the same cost functions and sends it to the same variable as BE
- Message size (and computation) is exponential in the induced width (= max separator size) for both techniques
- Since depth first order is a specific ordering $\rightarrow$ DPOP is part of BE

DFS tree and efficiency

DFS ordering and efficiency

- Depth first order is crucial for DPOP efficiency
- Coordination overhead is exponential in the induced width
- Heuristics to guide the DFS search:
  - Maximum Connected Node MCN
  - Maximum Cardinality Set (for DFS) MCS
- DFS induces only a specific set of orderings thus we might loose good orderings to keep local computation
- Trade off depends on application settings

DFS Heuristics: MCN

Maximum Connected node

- Choose node with maximum number of neighbours as root
- Select the neighbour with the highest number of neighbours
- Brake ties arbitrarily (e.g. lower/higher Id)

DFS Heuristics: MCS

Maximum Cardinality Set for DFS

- Maximum cardinality does not produce a DFS in general, must be adapted to DFS
  - Choose a random node as root
  - Select the neighbour with the highest number of visited neighbours