Introduction to Multi-Agent Programming

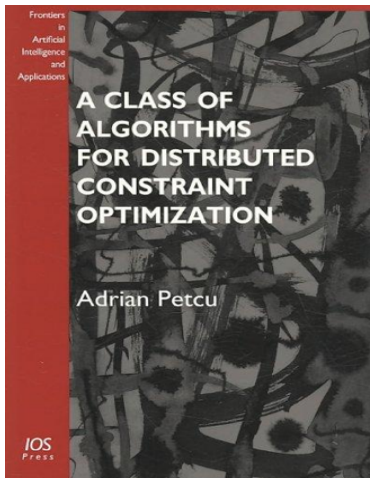
9. Working Together - Part II

Centralized and Decentralized Assignment Problem

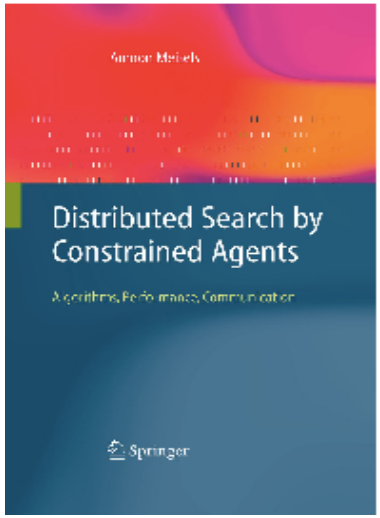
Alexander Kleiner, Bernhard Nebel

Contents

Literature



Adrian Petcu:
A Class of Algorithms for Distributed Constraint Optimization



Amnon Meisels:
Distributed Search by Constrained Agents
- Algorithms, Performance, Communication

The Assignment Problem

Introduction

- Consider the situation of assigning *n* jobs to *n* machines
  - For example, passengers to drivers in a car sharing domain
  - Cleaning robots to rooms
  - Wedding problem with individual sympathy measure
  - …
- When assigning workers *i (=1,2,....,n)* to machine *j (=1,2, ...n)* costs c_{ij} occur
- The objective is to assign the workers to machines at the least possible total cost
- Note when there *k* machines and *n* persons *(k<n)* the matrix can artificially be made square by adding *n-k* dummy persons ("no job") that are initialized with zeros
- Decentralized / Distributed assignment:
  - No central instance for decision making
  - Agents only communicate with their immediate neighbors (e.g. communication range)

Linear Sum Assignment Problem (LSAP)
Introduction

- Given an *n x n* cost matrix $\mathbf{C} = (c_{ij})$ match each row to a different column in such a way that the sum of the corresponding entries is minimized
- Example:
  - Four persons have to be assigned to 4 cleaning jobs, each one has different costs:

	Bathroom	Floors	Windows	Kitchen
Gabi	8 €	10 €	17 €	9 €
Malte	3 €	8 €	5 €	6 €
Tom	10 €	12 €	11 €	9 €
Patrick	6 €	13 €	9 €	7 €

- Costs c_{ij} are assumed non-negative. Negative costs can be converted by adding to each element of **C** the value $v = -\min_{i,j}\{c_{ij}\}$

Linear Sum Assignment Problem (LSAP)

Mathematical Formulation

Assignment Matrix: $x_{ij} = \begin{cases} 1 & \text{if row } i \text{ is assigned to column } j, \\ 0 & \text{otherwise,} \end{cases}$

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$s.t. \quad \sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1,2,...,n),$$

i^{th} person will do only one job

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1,2,...,n),$$

j^{th} work will be done by only one person

$$x_{ij} \in \{0,1\} \quad (i, j = 1,2,...,n).$$

Note: There are n! valid solutions. For example n=10 → 3.63×10^6

Hungarian Method

Procedure

The Hungarian Method is dependent upon two theorems:

Theorem 1:
When adding (or subtracting) a constant to every element of any row (or column) of the cost matrix (c_{ij}) then an assignment which minimizes the total cost for the new matrix will also minimize the total cost matrix.

Theorem 2:
If all $c_{ij} \geq 0$ and there exists a solution such that $\sum c_{ij} x_{ij} = 0$ then the solution is optimal.

→ The Hungarian Method (orig. version) solves LSAPs in $O(|V^3|)$ (Kuhn and Munkers)

→ Stepwise computational procedure on the cost matrix

Hungarian Method

Stepwise Procedure on the cost matrix

Step 1*: Row Reduction*

Subtract the minimum entry of each row from all entries in this row

Step 2: *Column Reduction*

Subtract the minimum entry of each column from all entries in this column

Step 3: *Zero Assignment*

(a) Rows: Examine each row (starting with the first one) until finding a row that contains exactly one zero. Mark this zero as *temporary assignment* and cross all entries in the column where the assignment has been made.
(b) Columns: Examine each column (starting with the first one) until finding a column that contains exactly one zero. Mark this zero as *temporary assignment* and cross all entries in the row where the assignment has been made.

Continue until all zeros are either crossed out or assigned!

Hungarian Method

Stepwise Procedure on the cost matrix

Continue until all zeros have either been assigned or crossed-out!

Then there are *two* possible outcomes:

1. Each column and row contains exactly one marked zero, i.e. total assigned ***zero's = n.*** The assignment is optimal.
2. At least two zeros are found in either a column or row, i.e. total assigned ***zeros < n***. Continue with *Step 4*

Step 4: *Draw the minimum number of lines to cover all zero's*

(a) Mark all rows in which the assignment has not been done
(b) See the position of zero in the marked row and mark the corresponding column
(c) Mark all other rows with a zero in this column

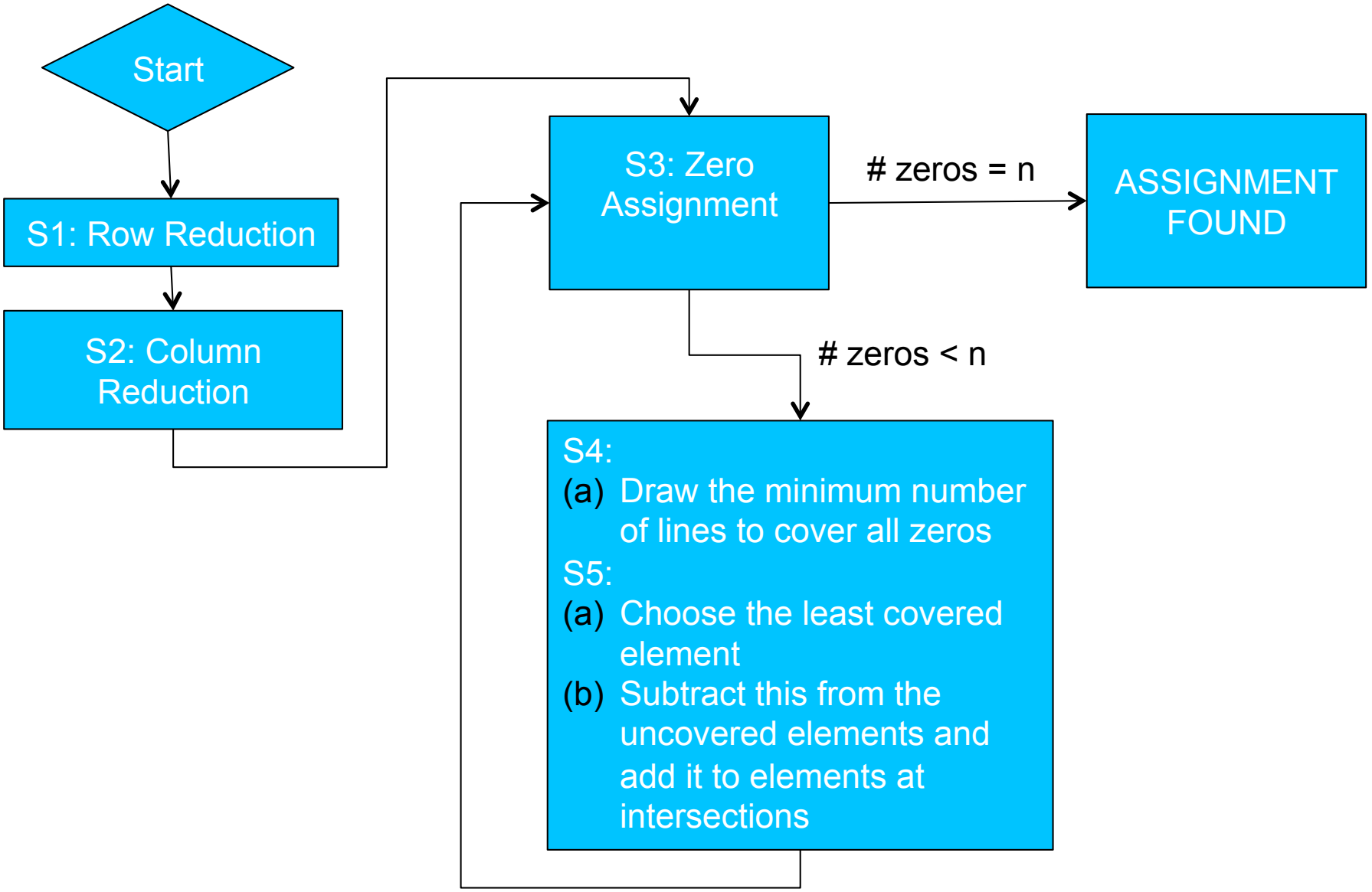
Step 5: *Select the smallest element from the uncovered elements*

(a) Subtract this element from all uncovered ones
(b) Add this element to all elements which are at the intersection of two lines

Step 6: Now we have increased the number of zeros. Repeat *Step 3.*

Hungarian Method

Summary



Hungarian Method

Example I

Example Problem I:

Four persons have to be assigned to 4 cleaning jobs, each one has different costs:

	Bathroom	Floors	Windows	Kitchen
Gabi	8	10	17	9
Malte	3	8	5	6
Tom	10	12	11	9
Patrick	6	13	9	7

Step 1: Row reduction

	Bathroom	Floors	Windows	Kitchen
Gabi	0	2	9	1
Malte	0	5	2	3
Tom	1	3	2	0
Patrick	0	7	3	1

Step 2: Column reduction

	Bathroom	Floors	Windows	Kitchen
Gabi	0	0	7	1
Malte	0	3	0	3
Tom	1	1	0	0
Patrick	0	5	1	1

Hungarian Method

Example I

Step 3: Zero assignment:

	Bathroom	Floors	Windows	Kitchen
Gabi	0	0	7	1
Malte	0	3	0	3
Tom	1	1	0	0
Patrick	0	5	1	1

Temporary assignments

Cross-outs

Finally, all zeros are either crossed out or assigned and the number of assigned zeros is **n** and this the solution optimal!

Result: Gabi → Floors, Malte → Windows, Tom → Kitchen, Patrick → Bathroom

Hungarian Method

Example II

Example Problem II:

5 drivers can pick-up passengers from 6 areas at different costs

	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	6	12	3	11	15
Malte	4	2	7	1	10
Tom	8	11	10	7	11
Patrick	16	19	12	23	21
Robert	9	5	7	6	10

Step 1: Row reduction

	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	3	9	0	8	12
Malte	3	1	6	0	9
Tom	1	4	3	0	4
Patrick	4	7	0	11	9
Robert	4	0	2	1	5

Step 2: Column reduction

	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	2	9	0	8	8
Malte	2	1	6	0	5
Tom	0	4	3	0	0
Patrick	3	7	0	11	5
Robert	3	0	2	1	1

Hungarian Method

Example II

Step 3 (a): Zero row assignment:

	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	2	9	0	8	8
Malte	2	1	6	0	5
Tom	0	4	3	~~0~~	0
Patrick	3	7	~~0~~	11	5
Robert	3	0	2	1	1

Step 3 (b): Zero column assignment:

	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	2	9	0	8	8
Malte	2	1	6	0	5
Tom	0	4	3	~~0~~	~~0~~
Patrick	3	7	~~0~~	11	5
Robert	3	0	2	1	1

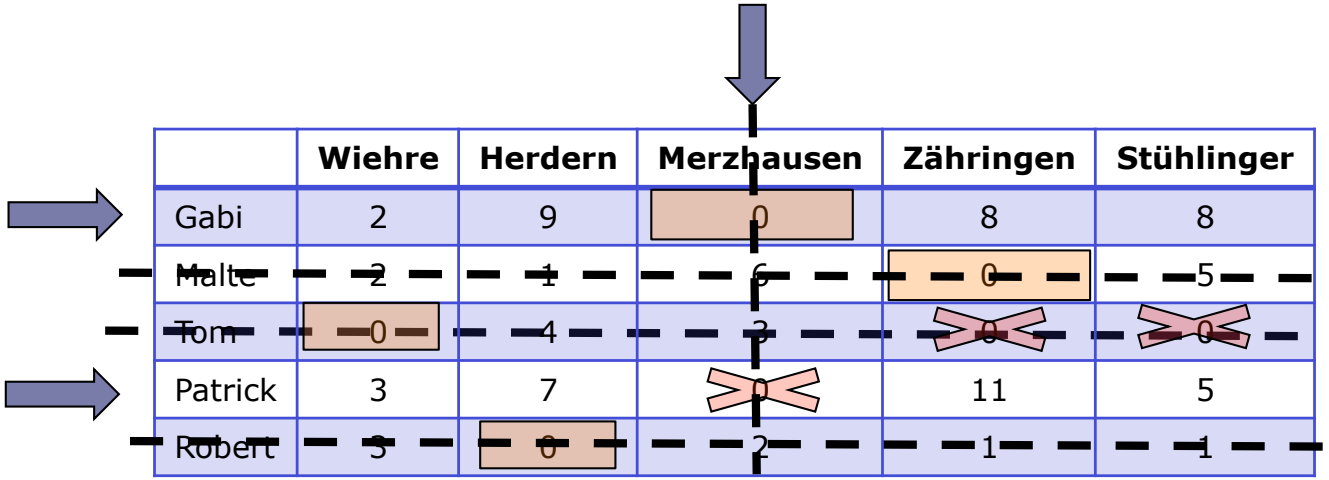
Result:

Now all the zeros are either assigned or crossed out, but the total number assigned **zero's<n** (4<5).
Therefore, we have to follow step 5 as follows:

Hungarian Method

Example II

Step 4: Draw the minimum number of lines to cover all zero's



	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	2	9	0	8	8
Malte	2	1	6	0	5
Tom	0	4	3	0	0
Patrick	3	7	0	11	5
Robert	3	0	2	1	1

Step 5: The smallest element among the uncovered ones is 2 then

(a) subtract 2 from all uncovered elements

(b) add 2 to the entries at the junction of two lines

	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	0	7	0	6	6
Malte	2	1	8	0	5
Tom	0	4	5	0	0
Patrick	1	5	0	9	3
Robert	3	0	4	1	1

Repeat Step 3: Finally we have got five assignments:
Gabi→ Wiehre, Malte → Zähringen, Tom → Stühlinger, Patrick → Merzhausen, Robert → Herdern

	Wiehre	Herdern	Merzhausen	Zähringen	Stühlinger
Gabi	0	7	0	6	6
Malte	2	1	8	0	5
Tom	0	4	5	0	0
Patrick	1	5	0	9	3
Robert	3	0	4	1	1

Distributed Constraint Optimization (DCOP or DisCOP)

"How do a set of agents optimize over a set of alternatives that have varying degrees of global quality?"

- The Distributed Constraint Optimization (DCOP) is a general model for distributed problem solving
- DCOPs are composed of agents, each holding one or more variables
  - Each variable has a domain of possible value assignments
- Constraints among variables (possibly held by different agents) assign costs to combinations of value assignments
- Agents assign values to their variables and communicate with each other, attempting to generate a solution that is globally optimal with respect to the sum of the costs of the constraints

Distributed Constraint Optimization (DCOP or DisCOP)

- *Examples*
  - allocating agents to targets:
    - Assign agents (i.e., fire fighters) to targets (i.e., fires) such that the adequate number of agents is allocated to each target
  - meeting scheduling:
    - arrange a set of meetings with varying participants such that no two meetings involving the same person are scheduled at the same time, while respecting order and deadline constraints
- *Difficulties*
  - No global control/knowledge
  - Localized communication
  - Limited time

Distributed Constraint Optimization (DCOP or DisCOP)

- Why distributed?
  - Cost of formalization:
    - when problem solving is centralized, each participant will have to formulate its constraints on all imaginable options beforehand.
    - in contrast, when using open constraint satisfaction, agents are asked to evaluate only a minimal number of constraints.
  - Privacy issues:
    - in a meeting scheduling scenario, the fact that person A is also meeting with person B may be private information that A wants to keep from another person C. When problem solution is centralized, the solver will see all meetings and constraints that can easily be leaked or stolen.
    - In contrast, a distributed solution can be constructed in such a way that agents only reveal information piecemeal when evaluating constraints.
  - Robustness:
    - a centralized solver creates a central point of failure that leads to brittleness of the entire system.
    - When solving is distributed among different agents, it allows load balancing and redundant and thus fault-tolerant and more efficient computation among different agents, leading to more reliable systems.

Constraint Optimization Problem (COP)

Problem Formulation

A **COP** is a tuple $\langle \mathbf{X}, \mathbf{D}, \mathbf{R} \rangle$ such that:

- $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of variables (e.g. start times of meetings)
- $\mathbf{D} = \{d_1, \ldots, d_n\}$ is a set of discrete, finite variable domains (e.g. time slots)
- $\mathbf{R} = \{r_1, \ldots, r_n\}$ is a set of utility functions, where each r_i is a function with the scope $(X_{i_1}, \ldots, X_{i_k}), r_i : d_{i_1} \times \ldots \times d_{i_k} \rightarrow \Re$ assigning a utility (reward) to each possible combination of values
- Special case hard constraints: They forbid certain value combinations by assigning *0* to feasible and *–inf* to infeasible once
- Goal is to find instantiation X* maximizing the sum of utilities of individual utility functions:

$$X^* = \arg\max_X \left(\sum_{r_i \in \mathbf{R}} r_i(X) \right)$$

Distributed COP (DCOP)

Problem Formulation

A **DCOP** is a tuple $\langle \mathbf{A}, \mathbf{COP}, \mathbf{R}^{ia} \rangle$ such that:

- $\mathbf{A} = \{A_1, ..., A_k\}$ is a set of agents (e.g. people participating in meetings)
- $\mathbf{COP} = \{COP_1, ..., COP_k\}$ is a set of disjoint centralized COPs, where each COP_i is the local sub-problem controlled by agent A_i
- $\mathbf{R}^{ia} = \{r_1, ..., r_n\}$ is a set of inter-agent utility functions defined over variables from several different sub problems COP_i. Each $r_i : scope(r_i) \rightarrow \Re$ expresses the reward obtained by the involved agents for some joint decision. Hard constraints are simulated by assigning 0 to feasible and *–inf* to infeasible once

DCOP Example Applications

Distributed Meeting Scheduling

The Meeting Schedule Problem (MSP) is defined by the tuple $\langle \mathbf{A},\mathbf{M},\mathbf{P},\mathbf{T},\mathbf{C},\mathbf{R} \rangle$ such that:

$\mathbf{A} = \{A_1,\ldots,A_k\}$ is a set of agents.

$\mathbf{M} = \{M_1,\ldots,M_n\}$ is a set of meetings.

$\mathbf{P} = \{p_1,\ldots,p_k\}$ is a set of mappings from agents to meetings: each $p_i \subseteq \mathbf{M}$ is the set of meetings A_i attends.

$\mathbf{T} = \{t_1,\ldots,t_n\}$ is a set of time slots; each meeting can be hold within one available slot.

$\mathbf{R} = \{r_1,\ldots,r_k\}$ is a set of utility functions; a function $r_i : p_i \rightarrow \Re$ expressed by agent A_i represents A_i's utility for each possible schedule of its meetings.

Goal: To find schedule that is feasible (i.e. two meetings sharing an agent must not overlap) and maximizes the sum of agents' utilities. Note the MSP is NP-*hard.*

DCOP Example Applications

Distributed Meeting Scheduling

Example: Three agents want to find optimal schedule for three meetings:

$$A_1 : \{M_1, M_3\}\ ,\ \ A_2 : \{M_1, M_2, M_3\},\ \ A_3 : \{M_2, M_3\}.$$

There are 3 possible time slots: 8AM, 9AM, and 10AM. Each agent has

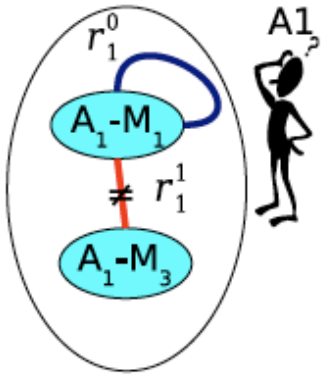
A local scheduling problem COP_i composed of:

- Variables $A_i \_ M_j$: one variable for each meeting Ai wants to participate
- Domains: the time slots 8AM, 9AM, and 10AM
- Hard constrains: No two meetings of A_i may overlap
- Utility functions: A_i's preferences

DCOP Example Applications

Distributed Meeting Scheduling

(a) Local subproblem COP_1 of A_1



r_1^0

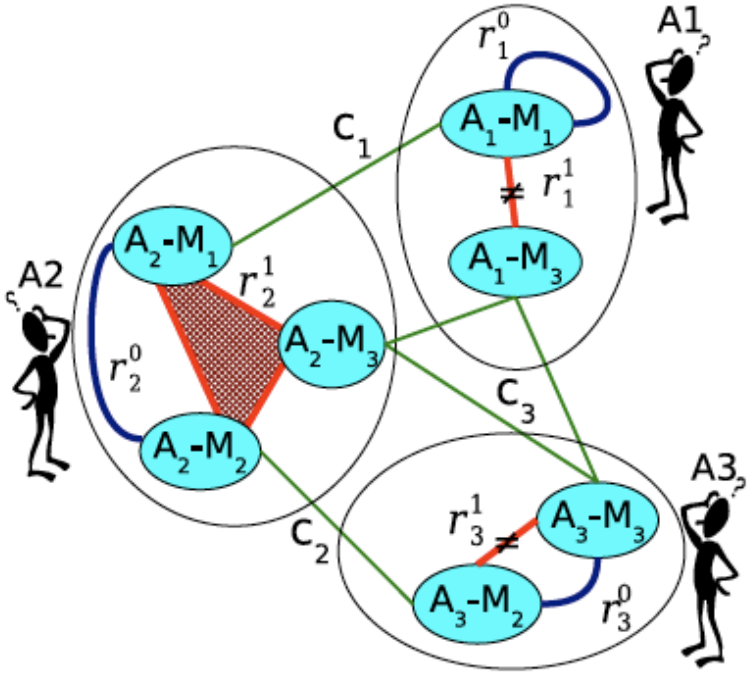
	U(A1-M1)		
Time	8	9	10
Utility	2	5	10

r_1^1

M3 \ M1	8	9	10
8	-inf	0	0
9	4	-inf	0
10	6	5	-inf

- A1 participates in M1 and M3 (non overlap)
- One variable for each meeting = its time slot
- 3 possible values for each variable: 8,9,10
- r_1^1: non overlap of M1 and M3 (-inf on diagonal)
  : A1 prefers M3 to take place *after* M1
- r_1^0: A1 prefers M1 to start late

(b) DCOP with 3 agents and 3 meetings



- Each agent has its own local subproblem
- Inter-agent equality constraints ensure agents agree on the same schedule

c_i: inter-agent constraints with c_i=0 for combinations assigning the same values to variables and –*inf* for different assignments

DCOP Example Applications

Distributed Resource Allocation

- Distributed sensor allocation problem (**SAP**) consists of:
  - a sensor field composed of *n* sensors: $S = \{s_1, s_2, ..., s_n\}$
  - *m* targets that need to be tracked: $T = \{t_1, t_2, ..., t_m\}$
  - *each sensor has a certain "range" (the maximum distance that it can cover)*
  - *in order to successfully track a target, 3 sensors have to be assigned to that target (triangulation can be applied using the data coming from those 3 sensors)*
- *The following restrictions apply:*
  - *Any sensor can only track one target at a time*
  - *the sensors in the field can communicate among themselves, but not necessarily every sensor with every other sensor (the sensor connectivity graph is not fully connected). The 3 sensors tracking a given target must be able to communicate among themselves*

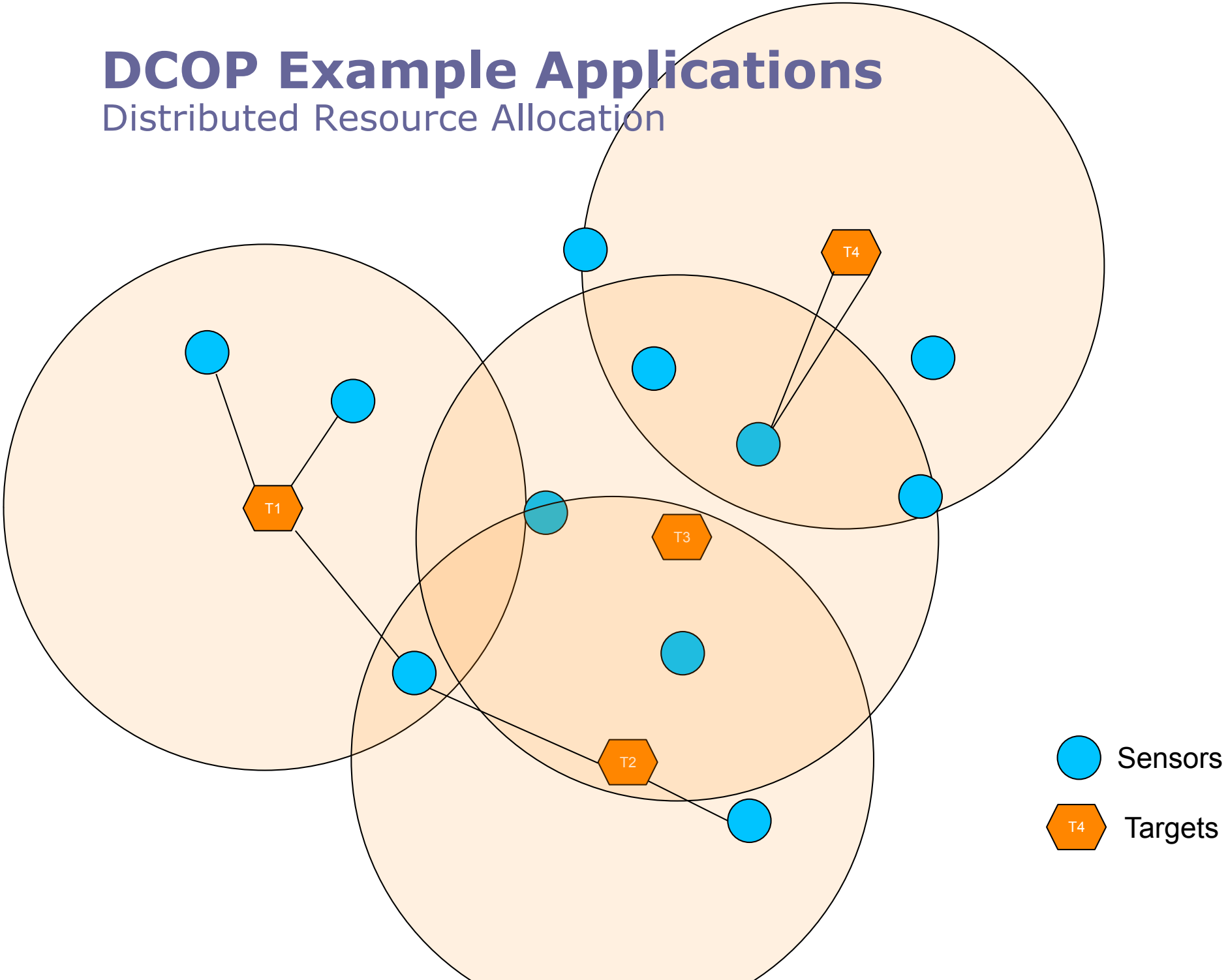
DCOP Example Applications

Distributed Resource Allocation

- We assign one agent for each target with 3 variables, one for each required sensor
  - Example:
    - Agent A_i assigned for target T_i has 3 variables s_1^i, s_2^i, s_3^i representing the sensors that have to be assigned to track the target
    - The domain of each variable consists of the sensors that can *see* the target
- Constraints:
  - Intra-agent:
    - one agent (target) must have 3 different sensors tracking it
    - there must be a communication link between the sensors
  - Inter-agent:
    - No two variables s_k^i, s_l^j from any two agents A_i and A_j can be assigned to the same value, i.e., one sensor can track only a single target.

DCOP Example Applications

Distributed Resource Allocation



DCOP Solvers

- *Synchronous Branch and Bound (SynchB&B)*
  - Hirayama and Yokoo 1997
  - distributed version of the centralized Branch and Bound algorithm
- ADOPT (Modi, Shen, Tambe, and Yokoo 2005)
  - Uses pseudo-tree derived from the structure of the constraints network in order to improve the process of acquiring a solution for the search problem.
  - Asynchronous search algorithm in which assignments are passed down the pseudo-tree. Agents compute upper and lower bounds for possible assignments and send costs which are eventually accumulated by the root agent up to their parents in the pseudo-tree.
- DSA (Zhang, Wang, Xing Wittenburg 2005)
  - Synchronous Stochastic Assignment
- DPOP (Petcu & Faltings 2005)
  - In DPOP, each agent receives from the agents which are its sons in the pseudo-tree all the combinations of partial solutions in their sub-tree and their corresponding costs.
  - The agent generates all possible partial solutions including the partial solutions it received from its sons and its own assignments
  - Once the root agent receives all the information from its sons, it produces the optimal solution and propagates it down the pseudo-tree to the rest of the agents.

Synchronous Branch and Bound (SBB)
Introduction

- Simulates the BB technique for CSPs in a distributed environment
- Starts with a fixed agent ordering, e.g., by agent IDs
  - $A_0, A_1, A_2, \ldots, A_n$
- Current Partial Assignment (CPA) is exchanged as a token among the agents according to the ordering until a solution is found
  - Synchronous and sequential processing!
  - Only the agent holding the CPA message may perform computation
- Pro:
  - Complete solution
- Drawbacks:
  - Slow since agents only perform computations when they hold the CPA
  - Most of the time other agents are idle

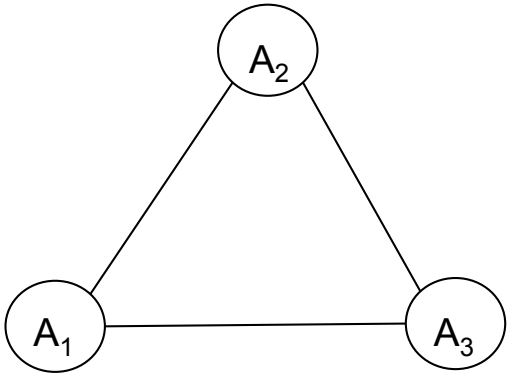
Synchronous Branch and Bound (SBB)
Algorithm

- The CPA starts at the first agent, which assigns its first value to it and sends it to the second agent
- Each agent that receives the CPA extends it by writing on it a value assignment to its variable, as well as the cost it incurred because of constraints with other assignments appearing in the received CPA
- Whenever the CPA reaches a new full assignment at the last agent, the accumulated cost of the CPA is the cost of that full assignment
  - If this cost is smaller than the known upper bound, it is broadcast to all agents as the new upper bound
- Each agent holding the CPA checks whether the CPA's accumulated cost is smaller than the upper bound.
  - If this is false, it assigns the next value in its domain instead of the current value and checks again
- An agent encountering an empty domain of values erases its assignment (and its cost) and sends the CPA back to the previous agent (backtracking)
- When the domain of the first agent is exhausted, the last discovered full assignment is reported as the solution (this requires remembering what that assignment was, which can be done by the last agent).

Synchronous Branch and Bound (SBB)

Example

Problem:



Domain:

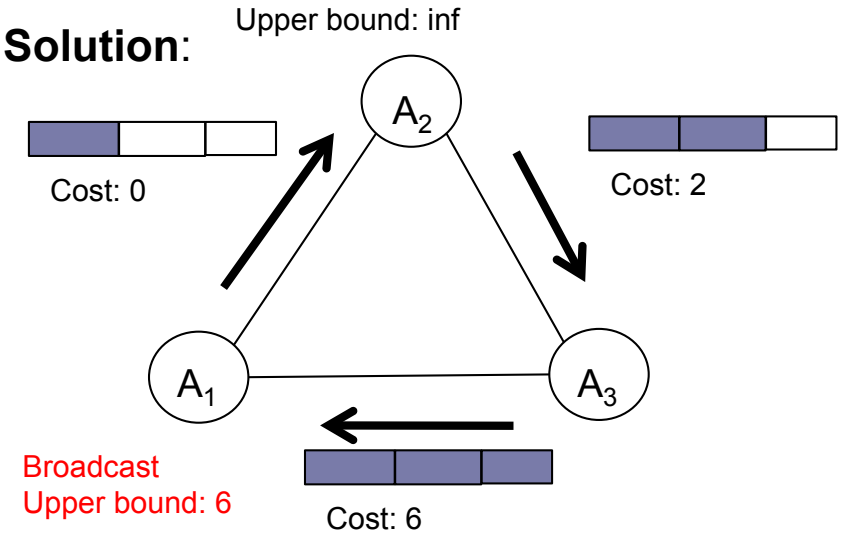
Constraints:

b/o, o/b → 5

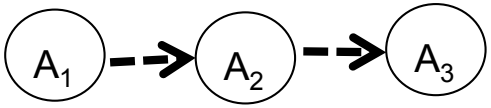
o/o → 0

b/b → 2

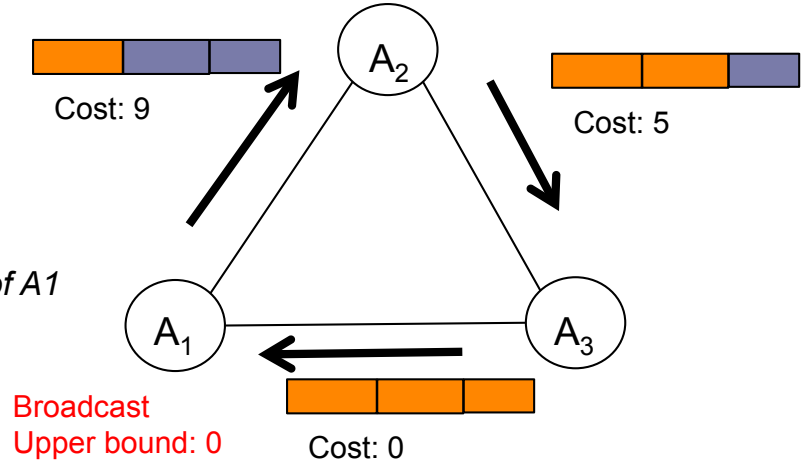
Solution:



Ordering:



Termination since domain of A1 is exhausted!



Distributed Stochastic Algorithm (DSA)

Overview

- The DSA algorithm is
  - synchronous, i.e. each step is executed by all agents at the same time (e.g. via a system clock)
  - uniform, i.e., agents do not require unique IDs
- The procedure
  - starts by assigning random values to variables
  - tries to reduce the number of violated constraints during each step
- Pro:
  - **Simplicity**
  - **Uniformity**, all processes have equal priority for every act, i.e., do not need identities to distinguish one another for breaking ties.
  - **Efficiency**
- Drawbacks
  - Synchronizing messages must be taken into account when measuring the overall communication load

Distributed Stochastic Algorithm (DSA)

Pseudo Code

Algorithm 2 Sketch of DSA, executed by an agent.

```
Randomly choose a value
while (no termination condition is met) do
   if (a new value is assigned) then
      send the new value to neighbors
   end if
   collect neighbors' values, if any; compute the best possible
   conflict reduction Δ
   if (Δ > 0) or (Δ = 0 but there is a conflict) then
      change to a value giving Δ with probability p
   end if
end while
```

Termination: after a defined number of steps

- **C** stands for conflict (yes/no)
- Δ is the best possible conflict reduction between 2 steps
- **v** is the value given Δ
- ***p*** the probability to change the current value
- Notice when Δ>0 there must be a conflict

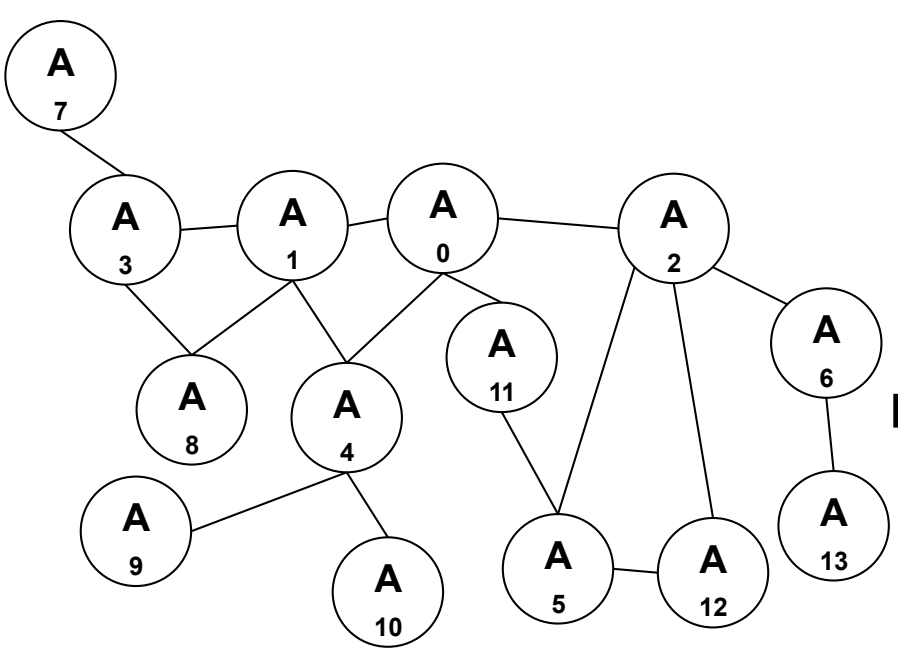
Algorithm	Δ>0	C, Δ=0	no C, Δ=0
DSA-A	v with p	-	-
DSA-B	v with p	v with p	-
DSA-C	v with p	v with p	v with p
DSA-D	v	v with p	-
DSA-E	v	v with p	v with p

Pseudo-tree Generation

- Pseudo-trees allow to split COPs into smaller subproblems
  - There is a single agent (node) that is placed at the root
  - Each agent has zero or more children
  - All agents except the root have a parent
- Constraints are only allowed between an agent and its ancestors!
  - If two agents share a constraint, then one of these agents is an ancestor of the other
  - All constraints are either from parent to son (tree edges), or from an agent to one of its ancestors (back edges)
- Simple solution: Build the depth first search tree (DFS) of the constraint graph, i.e., traverse the graph from a start node in a DFS manner
  - There are many possible DFS trees for a given constraint graph

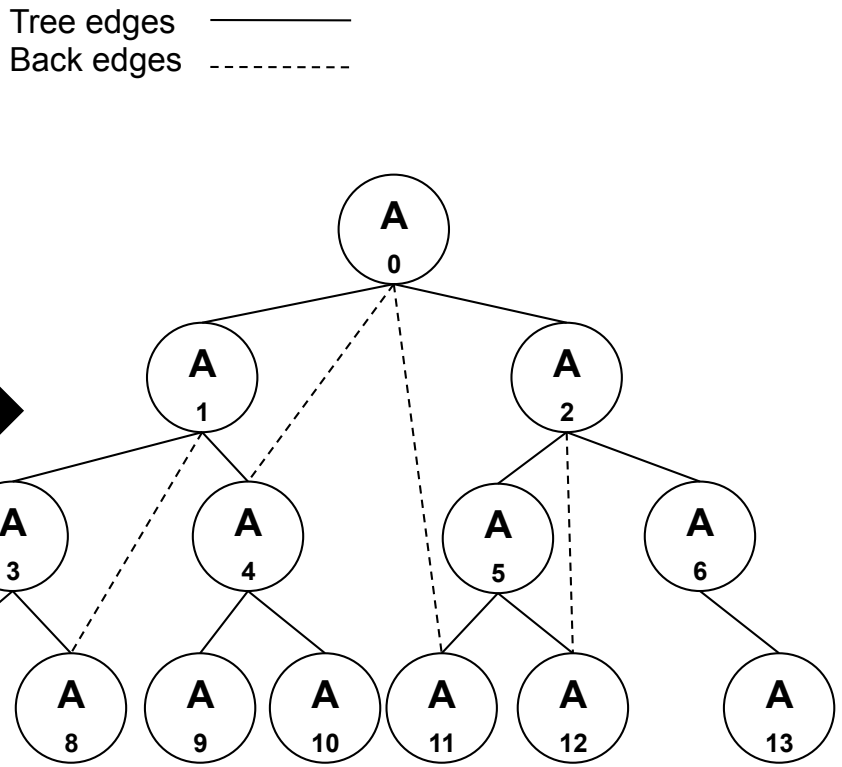
Pseudo-tree Generation

Example



Constraint graph

d_i	d_j	$f(d_i,d_j)$
0	0	1
0	1	2
1	0	2
1	1	0



Pseudo Tree

Distributed Pseudo-tree Optimization (DPOP)

- Composed of three phases:
  1. Generation of the pseudo-tree ordering of the agents
  2. UTIL propagation phase
  3. VALUE propagation phase
- UTIL phase:
  - Starts from the leaves (e.g. A_7 – A_{12}) and propagates UTIL messages up the tree only through tree edges
  - UTIL messages
    - Considers all direct parent dependencies (from tree & back edges) to compute for each value combination of dependent variables the optimal utility
    - For example: message A6 → A2 depends on A2 only, whereas A8 → A3 depends on all combinations of <X3,X1> resulting in the following UTIL message:

$X_8 \rightarrow X_3$	$X_3 = v_3^0$	$X_3 = v_3^1$	...	$X_3 = v_3^{m-1}$
$X_1 = v_1^0$	$u^*_{X_8}(v_1^0)$	$u^*_{X_8}(v_1^0)$	...	$u^*_{X_8}(v_1^0)$
...	...	...	...	...
$X_1 = v_1^{n-1}$	$u^*_{X_8}(v_1^{n-1})$	$u^*_{X_8}(v_1^{n-1})$	...	$u^*_{X_8}(v_1^{n-1})$

Max. Utilities given the configuration of the parents

Distributed Pseudo-tree Optimization (DPOP)

- UTIL phase (cont.):
  - Each node A_i waits until all UTIL messages from the children are received
  - A_i computes the maximal utility for each of its values given the parent dependencies and child messages
  - A_i memorizes its optimal values corresponding to each value assignment of its parents
- VALUE phase:
  - After all UTIL messages arrived at the root node X_R, X_R computes the overall utility corresponding to each of its values and picks the maximal one
  - X_R sends downwards its final choice to all children which also are able then to pick there optimal value and to send their choice further down the tree

Distributed Pseudo-tree Optimization (DPOP)

DPOP$(\mathcal{X}, \mathcal{D}, \mathcal{R})$
Each agent X_i executes:

Phase 1: pseudotree creation
elect leader from all $X_j \in \mathcal{X}$
elected leader initiates pseudotree creation
afterwards, X_i knows P(X_i), PP(X_i), C(X_i) and PC(X_i)
Phase 2: *UTIL* message propagation
if $|Children(X_i)| == 0$ (i.e. X_i is a leaf node) **then**
  $UTIL_{X_i}$(P(X_i)) ← Compute_utils(P(X_i), PP(X_i))
  Send_message(P(X_i), $UTIL_{X_i}$(P(X_i)))
activate UTIL_Message_handler()
Phase 3: *VALUE* message propagation
activate VALUE_Message_handler()
END ALGORITHM

UTIL_Message_handler(X_k,$UTIL_{X_k}(X_i)$)
store $UTIL_{X_k}(X_i)$
if *UTIL* messages from all children arrived **then**
  if Parent(X_i)==null (*that means X_i is the root*) **then**
    $v_i^* \leftarrow Choose\_optimal(null)$
    Send $VALUE(X_i, v_i^*)$ to all $C(X_i)$
  else
    $UTIL_{X_i}$(P(X_i)) ← Compute_utils(P(X_i), PP(X_i))
    Send_message(P(X_i), $UTIL_{X_i}$(P(X_i)))
return

VALUE_Message_handler($VALUE^{X_i}_{P(X_i)}$)
add all $X_k \leftarrow v_k^* \in VALUE^{X_i}_{P(X_i)}$ to agent_view
$X_i \leftarrow v_i^* = Choose\_optimal(agent\_view)$
Send $VALUE^{X_i}_{X_l}$ to all $X_l \in C(X_i)$

Choose_optimal(agent_view)

$$v_i^* \leftarrow argmax_{v_i} \sum_{X_l \in C(X_i)} UTIL_{X_l}(v_i, agent\_view)$$

return v_i^*

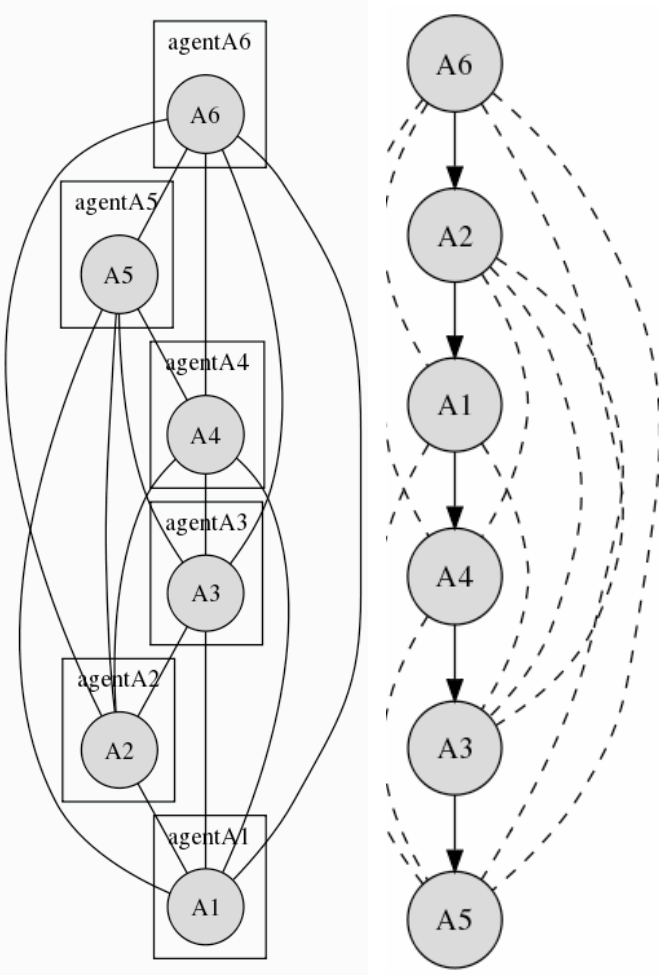
Compute_utils(P(X_i), PP(X_i))
for all combinations of values of $X_k \in PP(X_i)$ **do**
  let X_j be Parent(X_i)
  similarly to DTREE, compute a vector $UTIL_{X_i}(X_j)$ of all $\{Util_{X_i}(v_i^*(v_j), v_j) | v_j \in Dom(X_j)\}$
assemble a hypercube $UTIL_{X_i}(X_j)$ out of all these vectors (totaling $|PP(X_i)| + 1$ dimensions).
return $UTIL_{X_i}(X_j)$

P: Parent (tree edge)
C: Child (tree edge)
PP: Pseudo Parent (back edge)
PC: Pseudo Child (back edge)

Distributed Pseudo-tree Optimization (DPOP)

Drawbacks

- Fully synchronous, i.e., agents must wait for all their children to finish their computation before they can start to compute
- The size of the messages in DPOP can become be exponentially large
  - For example, in a fully connected problem, a pseudo-tree would be a chain, and if there are 10 agents, then the UTIL message from the leaf agent to its parent would include its cost for each assignment combination of the other nine agents. If the domains are of size 10, then this message contains at least 109 such values!

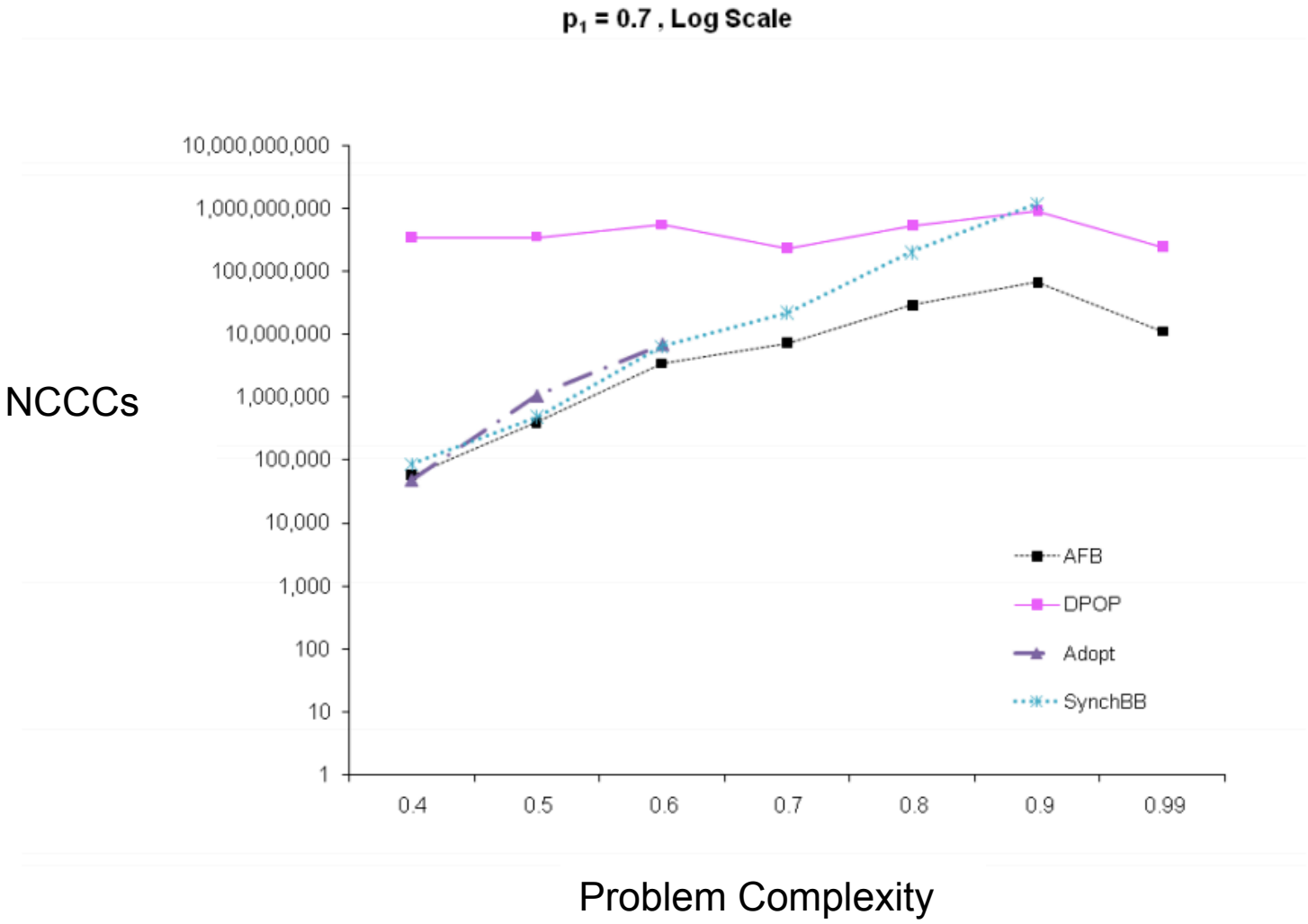


DCOP Performance Metrics

- In general, computation time **and** communication load are of interest! How to determine the right metric?
  - Counting Cycles of computation on each agent?
  - Big-O notation?
  - Counting the number of exchanged messages or the total amount of information sent?
- Non-Concurrent Constraint Checks (NCCCs)
  - Every agent holds a counter of constraint checks performed
  - Every message carries the value of the sending agent's counter
  - When an agent receives a message it updates its counter to the largest value between its own counter and the counter value carried by the message
  - After termination the largest counter held by some agent is reported as the cost of the search

DCOP Performance Metrics

Example: Testing complete methods



Links

Freiburg DCOP Benchmarking:

http://kaspar.informatik.uni-freiburg.de/~rslb

Frodo - An Open-Source Framework for DCOP:

http://liawww.epfl.ch/frodo/

Summary

- The assignment problem is fundamental in many real-world domains
- In general the assignment problem can be solved by distributed and centralized methods
- We discussed the Hungarian method, a centralized approach
  - The Hungarian method computes an optimal assignment in $O(n^3)$ running time
- Distributed Constraint Optimization (DCOP) offers a rich set of algorithms for solving the assignment problem decentralized
  - Here we distinguish between complete and incomplete methods
  - Complete methods, such as SyncBB and DPOP, typically do not scale-up with large agent teams
  - In contrast, we discussed DSA an efficient stochastic algorithm

Literature

- Paper on the Hungarian Method:
  - *Kuhn, Harold W.* The Hungarian Method for the Assignment Problem. In Jünger, Michael, Liebling, Thomas M., Naddef, Denis, Nemhauser, George L., Pulleyblank, William R., Reinelt, Gerhard, Rinaldi, Giovanni and Wolsey, Laurence A., **50 Years of Integer Programming** 1958-2008, pp. 29-47. Springer Berlin Heidelberg 2010.
- Paper on DSA:
  - *Zhang, W., Wang, G., Xing, Z. and Wittenburg, L. Distributed stochastic search and distributed* breakout: properties, comparison and applications to constraint optimization problems in sensor networks. **Artificial Intelligence** 161 (1-2):55-87, 2005.
- Paper on DPOP:
  - *Petcu, A. and Faltings, B.* A scalable method for multi-agent constraint optimization. In **International Joint Conference on Artificial Intelligence**, pp. 26-6, 2005.