LECTURE 6

SUMMARY

1. Distributed CSP

- if the problem is to **allocate tasks or resources to multiple agents** and there exists **inter-agents constraints**
  - $\Rightarrow$ the problem can be formalized as a distributed CSP (DCSPs)
  - each task/resource is a **variable**
  - the **possible assignments** are values
- many application problems in DAI can be formalized as distributed CSPs
  - interpretation problems
  - assignment problems
  - multiagent truth maintenance tasks

How can the CSP formalization be related to DAI?

- the variables of the CSP are distributed among agents
  - each process/agent corresponds to a variable
  - the processes act asynchronously to solve the CSP
  - $\Rightarrow$ **asynchronous search algorithms**
  - the processes that have links to X_i are called neighbors of X_i
  - **assumption**:
    - a process *knows* the identifiers of its neighbors
- the **communication model**:
  - processes/agents communicate by sending messages
    - a process can send messages to other processes iff
      - the process knows the addresses/identifiers of other processes
  - the delay in delivering a message is finite
  - for the transmission between the any pair of processes, messages are received in the order in which there were sent

1.1 Filtering algorithm (FA)

- assures **arc** consistency
- idea of the FA
  - each process communicates its domain to its neighbors
    - and removes values that cannot satisfy constraints from its domain
  - the process X_i performs the following procedure **REVISE**(X_i, X_j) for each neighboring process X_j

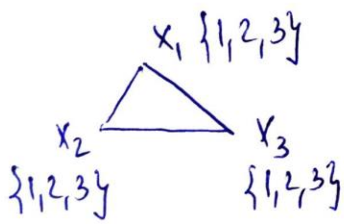
procedure REVISE(X_i, X_j)
for all $v_i \in D_i$ **do**

if there is no value $v_j \in D_j$ such that v_j is consistent with v_i **then**
@delete v_i from D_i
endif
endfor
endREVISE

- if some value of the domain of X_i is removed by performing **REVISE** ⇒
  - process X_i sends the new domain to neighboring processes
  - if X_i receives a new domain from a neighboring process X_j, the procedure **REVISE**(X_i, X_j) is performed again

!!! the execution order of the processes is arbitrary

- e.g., for the **3-queens** problem



a) X_1 : REVISE(X_1, X_2) removes 2
X_2 : REVISE(X_2, X_3) removes 2
X_3 : REVISE(X_3, X_2) removes 2

b) X_1 : REVISE(X_1, X_3) removes 1, 3
⇒ the domain of $X_1 = \emptyset$
overconstrained

In general, the FA cannot solve a DCSP

- it should be considered a preprocessing procedure
- is invoked before the application of other search methods
  - reduces the domains of variables for the search procedure

1.2 Hyper-Resolution based Consistency Algorithm

- a CSP is ***k*-consistent** iff the following condition is satisfied
  - given any instantiation of any k - 1 variables satisfying all the constraints among those variables, it is possible to find an instantiation of any k-th variable such that these k variable values satisfy all the constraints among them.
- the FA achieves **2- consistency**
- a ***k*-consistency** algorithm transform a given problem into an equivalent ***k*-consistent** CSP
- a CSP is **strongly *k*-consistent** if
  - the problem is ***k*-consistent**

  and

  - is ***j*-consistent** for all $j < k$

Theorem

- if the CSP has n variables and is strongly n-consistent

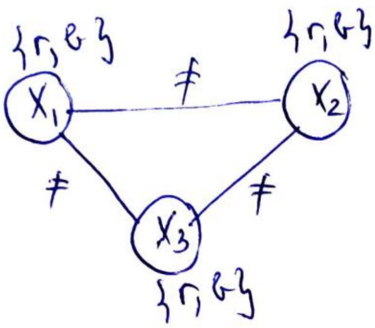
$\Rightarrow$

a solution can be obtained without any trial-and-error exploration

  - for a any instantiation of k-1 variables, one can always find at least one consistent value for the k-th variable

The HR-based consistency algorithm

- uses the **hyperresolution rule** (HR)
- all constraints are represented as a **nogood**
  - a prohibited combination of variables values
  - e.g.,



- a constraint between X_1 and X_2 can be represented as two nogoods:

1. $\{X_1=red, X_2=red\} \leftrightarrow \neg(X_1=red \wedge X_2=red)$
2. $\{X_1=blue, X_3=blue\} \leftrightarrow \neg(X_1=blue \wedge X_3=blue)$

since the domain of X_1 is {red, blue} $\Rightarrow$

3. $(X_1=red \vee X_1=blue)$

- the HR rule combines
  - nogoods (1., 2.) and
  - the condition that one variable takes one value from its domain (3.)

  and generates a new nogood

4. $\{X_2=red, X_3=blue\} \leftrightarrow \neg(X_2=red \wedge X_3=blue)$

  - if the following propositions hold

$\neg(p_1 \wedge p_2)$
$\neg(p_3 \wedge p_4)$
$(p_1 \vee p_3)$

$\Rightarrow$

$\neg(p_2 \wedge p_4)$

- the HR rule is described as follows

$(X_1=x_{11} \vee X_1=x_{12} \vee X_1=x_{13} \vee \ldots \vee X_1=x_{1m})$
$\neg(X_1=x_{11} \wedge X_{i1}=x_{i1} \wedge \ldots),$
$\neg(X_1=x_{12} \wedge X_{i2}=x_{i2} \wedge \ldots),$
...
$\neg(X_1=x_{1m} \wedge X_{im}=x_{im} \wedge \ldots)$

$$\rceil\ (X_{i1}=x_{i1} \wedge \ldots \wedge X_{i2}=x_{i2} \wedge \ldots X_{im}=x_{im} \wedge \ldots)$$

- in the **HR based consistency algorithm**
  - each process represents its constraints as nogoods
  - the process generates new nogoods using the HR rule
    - combining the information about its domain

    and
    - existing nogoods
  - a newly obtained nogood
    - is communicated to related processes
    - if a new nogood is communicated, the process tries to generate further nogoods
      - using the communicated nogood)
  - if an empty set becomes a nogood

    $\Rightarrow$ the problem is over-constrained
    - a superset of a nogood cannot be a solution

- **disadvantage**
  - the HR rule can generate a very large number of nogoods
- if we restrict the application of the rules so that
  - only the nogoods whose lengths (# of variables) are less than *k* are produced

  $\Rightarrow$ the problem becomes strongly *k*-consistent
- the HR rules generation is used in the **asynchronous backtracking** algorithm

1.3 Asynchronous Backtracking (AB)

- is an asynchronous version of the classical BT algorithm
- in the AB
  - the priority order of the variables/processes is determined by the alphabetical order of the variables identifiers $X_1, X_2, \ldots X_n$
  - each process communicates its tentative value assignment neighboring processes
  - a process **changes** its assignment if its current value assignment is **not consistent** with the higher priority processes
  - if there is no value that is consistent with higher priority processes

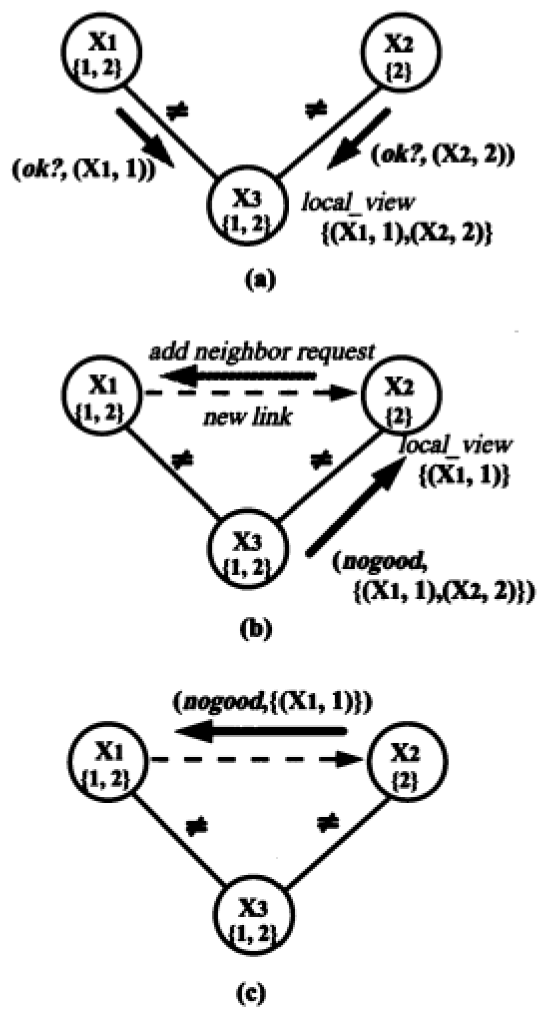
    $\Rightarrow$
    - the process generates a new **nogood**
    - communicates the nogood to a higher priority process

      $\Rightarrow$
      - the higher priority process changes its value
    - the generation of a nogood is identical to the HR rule
      - the AB algorithm generates only the nogoods that actually occur in the AB
  - **each process**
    - maintains the current value assignment of other processes from its viewpoint (***local_view***)

        - since the processes acts asynchronously and concurrently

          ⇒

            - the ***local_view*** may contain obsolete information
    - if a process X_i does not have a consistent value with the higher priority processes
        - it generates a **nogood**
        - the receiver of the nogood must check whether the nogood is actually violated from its own ***local_view***
- messages communicated between processes
    - ***ok?*** message
        - to communicate the current value
    - ***nogood*** message
        - to communicate a nogood
- the algorithm is described in [1], Section 4.2.4



[1]

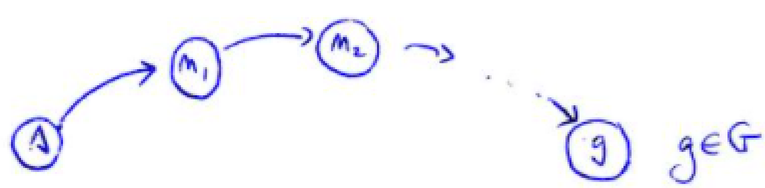
1.4 Other algorithms

- Asynchronous Weak-Commitment Search
    - the limitation of AB is that the process/variable ordering is statically determined
    - **AWCS** uses a value ordering heuristic
        - **min-conflict** heuristic
        - the value that minimizes the number of constraint violations with other variables is preferred
- Distributed Forward Checking
- Distributed BackJumping

2. Path finding problems ([1], Section 4.3)

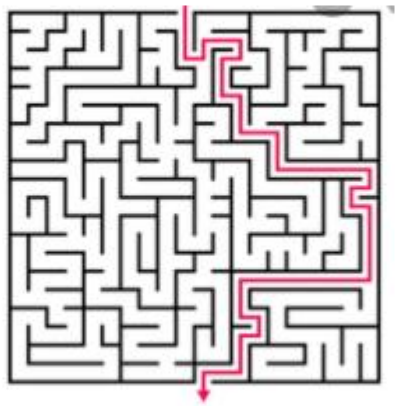
2.1 PFP – classical AI

- **definition**
- a set of nodes **N**, each representing a state;
- a set of directed links **L**, each representing an operator available to a problem solving agent;
- we assume that there exists a unique node ***s*** called the start node
    - representing the initial state
- a set of nodes **G** - each node from **G** represents a goal state;
- for each link, the *weight* of the link is defined
    - it represents the *cost* of applying the operator (is also called the *distance* between the nodes).
- we call the nodes that have directed links from a node ***i*** the *neighbors* of ***i***.
- **goal of PFP**
    - finding a path from ***s*** to a node ***g*** ∈ **G** , following the links between the states
        - any path
        - optimal path

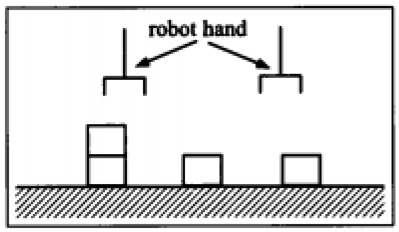


- **Examples**

– **Toy** problems:
  - *n*-puzzle
  - searching a maze



- planning problems for (multiple) robot hands



- **Software Engineering**
  - Many SE problems are NP-complete optimization problems => metaheuristic techniques (GA, SA, Tabu search, HC), evolutionary algorithms, machine learning.
  - Requirements analysis, design, development, maintenance, testing.
    - CITO: Class Integration Test Ordering optimization
- **Bioinformatics**
  - DNA sequencing and reconstruction, DNA fragment assembly, gene finding and identification, gene expression profiling, protein structure prediction (secondary, tertiary), phylogenetic trees.
- **Medicine**
  - Radiation treatment planning for cancer patients, medical data classification, gene mutations in cancer, medical image segmentation, brain tumour tissue segmentation, chemotherapy treatments.
- **Other**
  - Weight optimization in Recurrent Neural Networks, Reinforcement learning, etc.

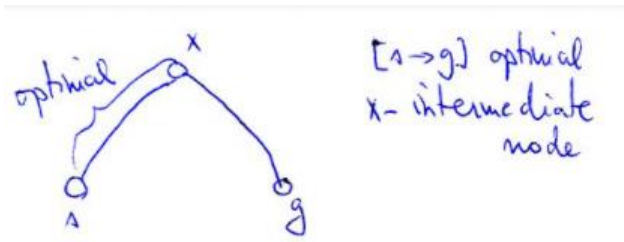
2.2 Distributed PFP (DPFP)

- *How can the PFP formalization be related to DAI?*
  - assume that multiple robots are exploring an unknown environment for finding a certain location (such a problem can be formalized as a DPFP)
- **algorithms for DPFP**
  1. Asynchronous dynamic programming (ADP)
     - the basis of other algorithms
  2. Learning Real time A* - LRTA* special cases of ADP
     Real Time A* - RTA*

Moving Target Search
Real Time Bidirectional Search
Real Time Multiagent Search

2.3 Asynchronous Dynamic Programming (ADP)

- in a PFP, the *principle of optimality* holds (if a path is optimal, then every segment of it is optimal)



- let us denote by $h^*(i)$ - the shortest distance from node i to a goal node
- from the principle of optimality, the shortest distance via a neighbouring node j is given by
    - $f^*(i, j) = k(i, j) + h^*(j)$
        - where $k(i,j)$ represents the cost of the link from node i to node j
- if node i is not a goal node => the path from i to a goal node must visit one of the neighbouring nodes $\Rightarrow$
    - $h^*(i) = \min_{j \text{ neighbour of } i} f^*(i,j)$ holds
- if h^* is given for each node, the optimal path can be obtained by repeating the following procedure
    - for each neighbouring node j of the current node i, compute $f^*(i, j) = k(i, j) + h^*(j)$
    - then, move to the node j that gives $\min_{j \text{ neighbour of } i} f^*(i,j)$.

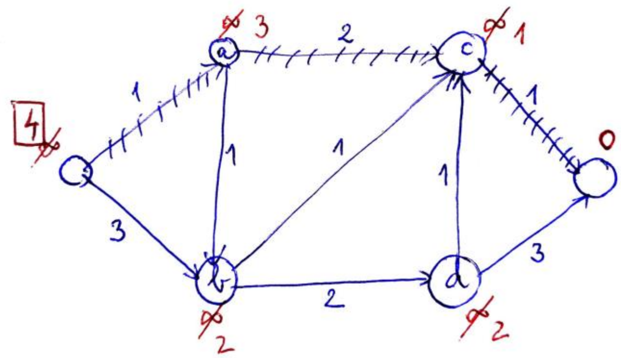
- ADP computes h^* by repeating the local computations for each node in the graph.

Let us assume the following situation

- for each node i, there exists a process/agent corresponding to i
- each process records $h(i)$ - the estimated value of $h^*(i)$
    - the initial value of $h(i)$ is arbitrary (e.g 0 or ∞) for all nodes excepting the goal nodes
    - for the goal nodes, the initial value for $h(i)$ is 0.
- each process can refer to the $h$values of its neighbour nodes (through shared memory/blackboard or message passing).
- the execution order of the processes is arbitrary
    - the processes are executed asynchronously and concurrently.

In this situation, each process i updates $h(i)$ by the following procedure.

- For each neighbouring node *j*, compute $f(i,j) = k(i,j) + h(j)$, where $h(j)$ is the current estimated distance from *j* to a goal node and $k(i,j)$ represents the cost of the link from node *i* to node *j*. Then, update $h(i)$ with $\min\limits_{j \text{ neighbour of } i} f(i,j)$.



<u>Theorem.</u> If the costs of all links are positive, then $h(i)$ will eventually converge to the true value $h^*(i)$.

<u>Remarks.</u>

1. In reality, ADP can not be used for a large PFP
   - if the number *n* of nodes is huge, we can not afford to have processes for all nodes.
2. ADP can be considered a foundation for all the other algorithms (LRTA*, RTA*, etc).
   - in these algorithms instead of allocating processes for all nodes, some kind of control is introduced for enabling the execution by a reasonable number of processes/agents).
3. More about DPFP may be found in [1], Section 4.3.

Bibliography

[1] Weiss, G. (Ed.): Multiagent Systems: *A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999 (available at www.cs.ubbcluj.ro/~gabis/weiss/Weiss.zip) [Ch. 4]

[2] Șerban Gabriela, *Sisteme Multiagent în Inteligenta Artificiala Distribuita. Arhitecturi si Aplicatii*, Editura RisoPrint, Cluj-Napoca, 2006

[3] Czibula Gabriela, *Sisteme inteligente. Instruire automată*, Editura RisoPrint, Cluj-Napoca, 2008