Distributed Constraint Processing

An Introduction

Gauthier Picard

MINES Saint-Étienne
LaHC UMR CNRS 5516

gauthier.picard@emse.fr

— Some contents taken from OPTMAS 2011 and OPTMAS-DCR 2014 Tutorials—

Contents

Introduction

Motivations

- Multi-agent systems are a way to model decentralised problem solving (privacy, distribution)
- Agents, having personal goals and constraints, negotiate as to reach a global equilibrium

$\Rightarrow$ distributed problem solving using agents

Approaches

- Classical CSP solver extensions
- Classical local search solver extensions

Cooperative Decentralized Decision Making

- Decentralised Decision Making
  - Agents have to coordinate to perform best actions
- Cooperative settings
  - Agents form a team $\rightarrow$ best actions for the team
- Why DDM in cooperative settings is important
  - Surveillance (target tracking, coverage)
  - Robotics (cooperative exploration)
  - Autonomous cars (cooperative traffic management)
  - Scheduling (meeting scheduling)
  - Rescue Operation (task assignment)

Distributed Constraint Optimisation Problems (DCOPs) for DDM

Why DCOPs for Cooperative DDM?

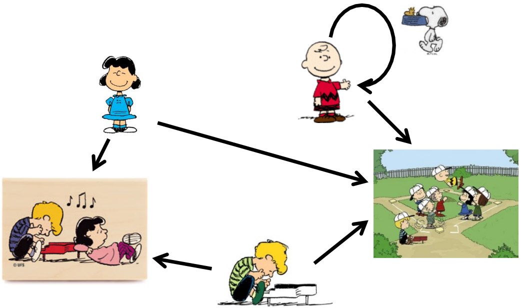
- Well defined problem
  - Clear formulation that captures most important aspects
  - Many solution techniques
    - Optimal: ABT, ADOPT, DPOP, ...
    - Approximate: DSA, MGM, Max-Sum, ...
- Solution techniques can handle large problems
  - compared for example to sequential decision making (MDP, POMDP)

Modeling Problems as DCOP

Target Tracking

- Why decentralize
  - Robustness to failure and message loss

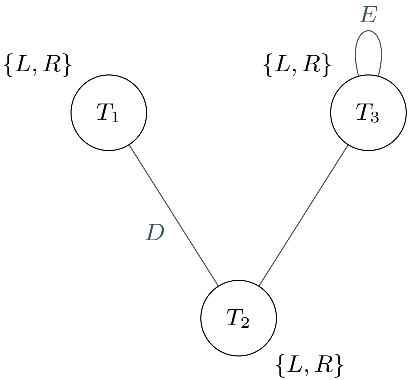
Meeting Scheduling



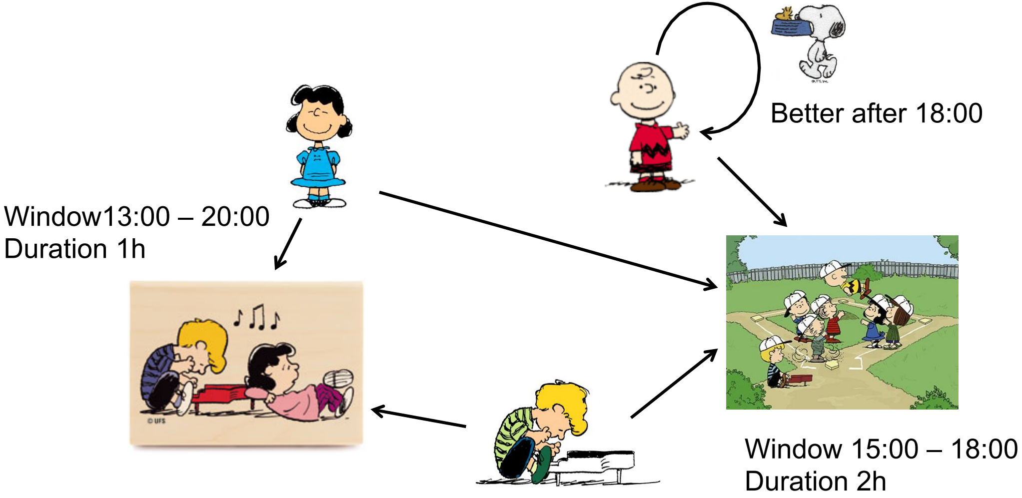
- Why decentralize
  - Privacy

Target Tracking as a DCOP

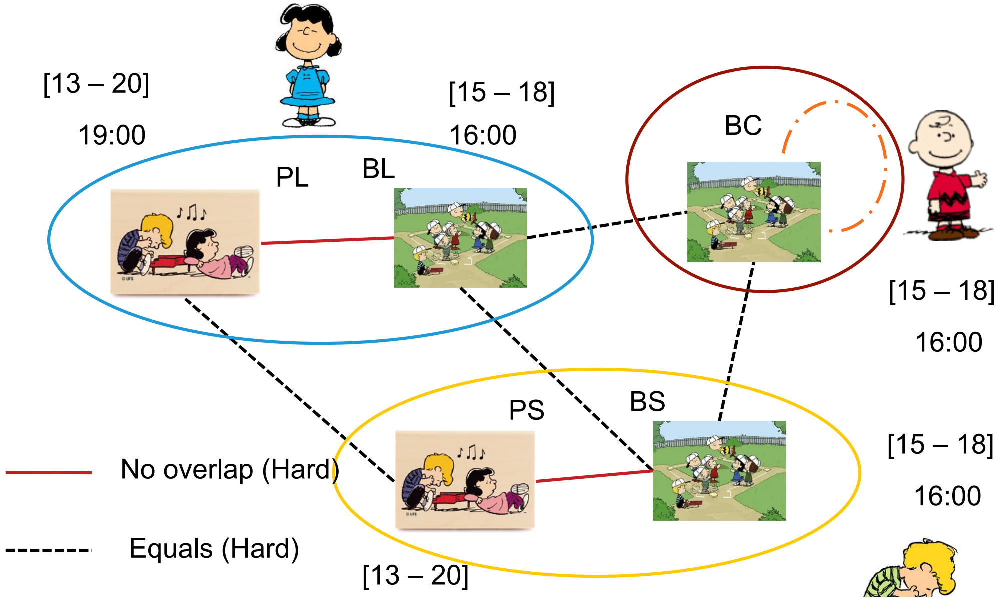
- Variables → Cameras
- Domains → Camera actions
  - look left, look right
- Constraints
  - Overlapping cameras
  - Related to targets
    - Diabolik, Eva
- Maximise sum of constraints



Meeting Scheduling as a DCOP



Meeting Scheduling as a DCOP



Benchmarking problems

Motivations

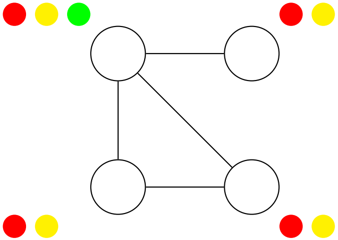
- Analysis of complexity and optimality is not enough
- Need to empirically evaluate algorithms on the same problem

Graph coloring

- Simple to formalise very hard to solve
  - Well known parameters that influence complexity
    - Number of nodes, number of colors, density (number of link/number of nodes)
- Many versions of the problem
  - CSP, MaxCSP, COP

Graph Coloring

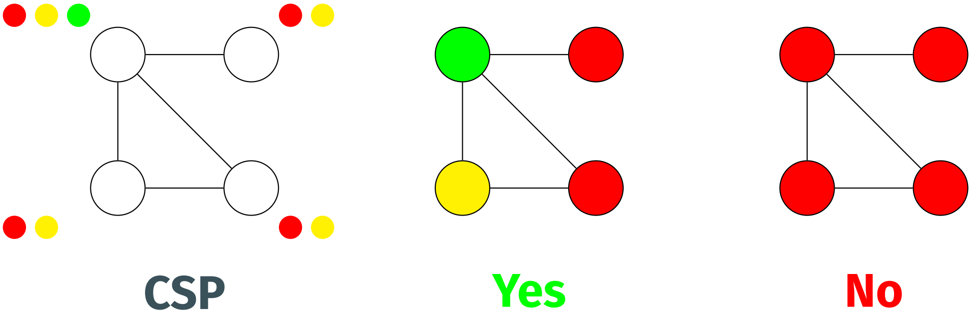
- Network of nodes
- Nodes can take on various colors
- Adjacent nodes should not have the same color
  - If it happens this is a conflict



CSP

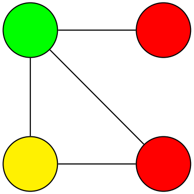
Graph Coloring

- Network of nodes
- Nodes can take on various colors
- Adjacent nodes should not have the same color
  - If it happens this is a conflict

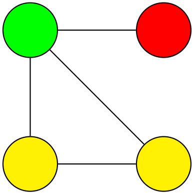


Graph Coloring - MaxCSP

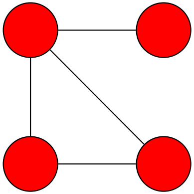
- Optimization Problem
- Natural extension of CSP
- Minimise number of conflicts



0



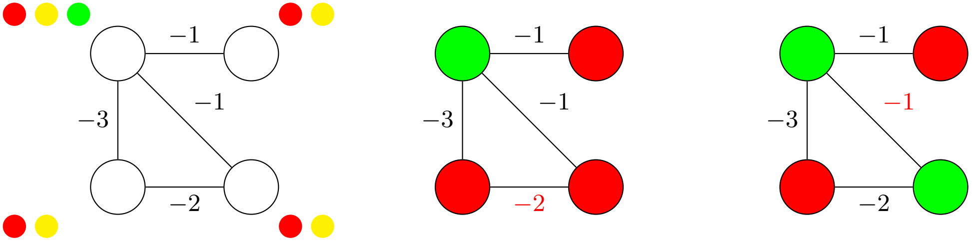
-1



-4

Weighted Graph Coloring - COP

- Optimization Problem
- Conflicts have a weight
- Maximise the sum of weights of violated constraints



COP **-2** **-1**

Constraint Satisfaction Problems [Dechter, 2003]

Definition (CSP)

A CSP is a triplet $\langle X, D, C \rangle$ such as:

- $X = \{x_1, \dots, x_n\}$ is the set of *variables* to instantiate.
- $D = \{D_1, \dots, D_m\}$ is the set of *domains*. Each variable x_i is related to a domain of value.
- $C = \{c_1, \dots, c_k\}$ is the set of *constraints*, which are relations between some variables from X that constrain the values the variables can be simultaneously instantiated to.

Definition (Solution to a CSP)

A solution to a CSP is a complete assignment of values from D to variables from X such that every constraint in C is satisfied.

Issues in CSP

Classical CSPs

- Constraint satisfaction is NP-complete in general
- Constraints are generally expressed as binary constraints
- The topology of a constraint-based problem can be represented by a *constraint network*, in which vertexes represent variables and edges represent binary constraints between variables

Extensions

- Distribution : variables, constraints
  - ex.: constraint c_i belongs to stakeholder j, $\phi(c_i) = j$ (or $belongs(c_i, j)$)
- Dynamics : adding removing variables and/or constraints at runtime

Multi-Agent Approaches to CSP

- **Complete and asynchronous solvers** for combinatorial problems, within the DisCSP framework, such as Asynchronous Backtracking (ABT) or Asynchronous Weak-Commitment Search (AWCS)
- **Distributed local search** methods, such as Distributed Breakout Algorithm (DBA) or Environment, Reactive rules and Agents (ERA) approach

Asynchronous Algorithms for DisCSP

Idea

- Inspired by classical centralised algorithms to solve CSP
- Each agent is responsible for assigning one (or several) variables
- Agents propose values to some other agents (depending on the organisation i.e. constraint network)

Main algorithm: Asynchronous backtracking (ABT) [Yokoo, 2001]

- Agents will perform a distributed version of the backtracking procedure
- ABT is complete
- Extensions exist to handle dynamics

Definition (DisCSP or DCSP)

A *DisCSP* (or DCSP) is a 5-uplet $\langle A, X, D, C, \phi \rangle$ where $\langle X, D, C \rangle$ is a CSP, A is a set of agents and $\phi : X \mapsto A$ is a function assigning variables from X to agents from A.

Centralised Backtracking

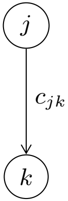
$i \leftarrow 0$
$D'_i \leftarrow D_i$
while $0 \leq i < n$ **do**
    $x_i \leftarrow$ null
    $ok? \leftarrow false$
    while *not ok? and* D'_i *not empty* **do**
        $a \leftarrow$ a value from D'_i
        remove a from D'_i
        if a *is in conflict with* $\{x_0, \ldots, x_{i-1}\}$ **then**
            $x_i \leftarrow a$
            $ok? \leftarrow true$
        end
    end
    if x_i *is null* **then** backtrack
        $i \leftarrow i - 1$
    else
        $i \leftarrow i + 1$
        $D'_i \leftarrow D_i$
    end
end

Asynchronous Backtracking (ABT) [Yokoo, 2001]

- First complete asynchronous algorithm for DisCSP solving
- Asynchronous:
  - All agents active, take a value and inform
  - No agent has to wait for other agents
- Total order among agents: to avoid cycles
  - $i < j < k$ means that: i more priority than j, j more priority than k
- Constraints are directed, following total order
- ABT plays in asynchronous distributed context the same role as backtracking in centralized

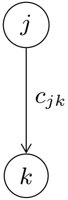
ABT: Directed Constraints

- Directed: from higher to lower priority agents
- Higher priority agent (j) informs the lower one (k) of its assignment
- Lower priority agent (k) evaluates the constraint with its own assignment
  - If permitted, no action
  - else it looks for a value consistent with j
    - If it exists, k takes that value
    - else, the agent view of k is a nogood, backtrack



ABT: Directed Constraints

- Directed: from higher to lower priority agents
- Higher priority agent (j) informs the lower one (k) of its assignment
- Lower priority agent (k) evaluates the constraint with its own assignment
  - If permitted, no action
  - else it looks for a value consistent with j
    - If it exists, k takes that value
    - else, the agent view of k is a nogood, backtrack



generates nogoods: eliminate values of k

ABT: Nogoods

Definition (Nogood)

Conjunction of (variable, value) pairs of higher priority agents, that removes a value of the current one

Example

- $x \neq y$, $d_x = d_y = \{a, b\}$, x higher than y
- When $[x \leftarrow a]$ arrives to y, this agent generates the nogood $[x = a \Rightarrow y \neq a]$ that removes value a of d_y
- If x changes value, when $[x \leftarrow b]$ arrives to y, the nogood $[x = a \Rightarrow y \neq a]$ is eliminated, value a is again available and a new nogood removing b is generated

ABT: Nogood Resolution

- When all values of variable y are removed, the conjunction of the left-hand sides of its nogoods is also a nogood
- **Resolution**: the process of generating the new nogood

Example

- $x \neq y$, $z \neq y$, $d_x = d_y = d_z = \{a, b\}$, x, z higher than y

  $x = a \Rightarrow y \neq a$

  $z = b \Rightarrow y \neq b$

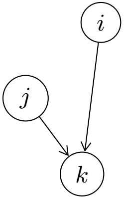
  $x = a \wedge z = b$ is a nogood

  $x = a \Rightarrow z \neq b$ (assuming x higher than z)

How ABT works

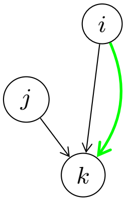
- **ABT agents**: asynchronous action, spontaneous assignment
- **Assignment**: j takes value a, j informs lower priority agents
- **Backtrack**: k has no consistent values with high priority agents, k resolves nogoods and sends a backtrack message
- **New links**: j receives a nogood mentioning i, unconnected with j ; j asks i to set up a link
- **Stop**: “no solution” detected by an agent, stop
- **Solution**: when agents are silent for a while (quiescence), every constraint is satisfied $\rightarrow$ solution; detected by specialized algorithms

ABT: Messages



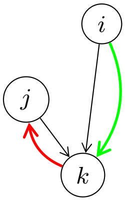
ABT: Messages

- $Ok?(i \rightarrow k, a)$:
  - i informs k that it takes value a



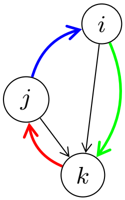
ABT: Messages

- $Ok?(i \to k, a)$:
  - i informs k that it takes value a
- $Ngd(k \to j, i = a \Rightarrow j \neq b)$
  - all k values are forbidden
  - k requests j to backtrack
  - k forgets j value
  - k takes some value
  - j may detect obsolescence



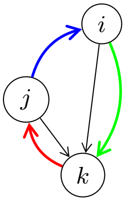
ABT: Messages

- $Ok?(i \to k, a)$:
  - i informs k that it takes value a
- $Ngd(k \to j, i = a \Rightarrow j \neq b)$
  - all k values are forbidden
  - k requests j to backtrack
  - k forgets j value
  - k takes some value
  - j may detect obsolescence
- $Addl(j \to i)$:
  - set a link from i to j, to know i value



ABT: Messages

- $Ok?(i \to k, a)$:
  - i informs k that it takes value a
- $Ngd(k \to j, i = a \Rightarrow j \neq b)$
  - all k values are forbidden
  - k requests j to backtrack
  - k forgets j value
  - k takes some value
  - j may detect obsolescence
- $Addl(j \to i)$:
  - set a link from i to j, to know i value
- Stop:
  - there is no solution



ABT Procedures

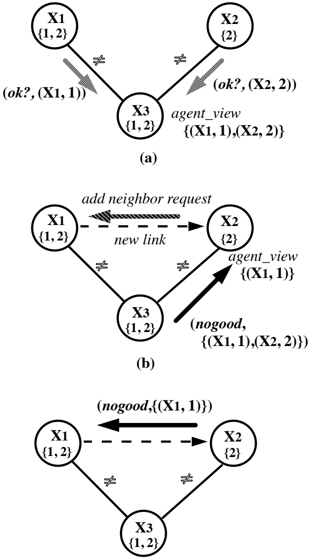
when received (**ok?**, (x_j, d_j)) **do** — (i)
  revise *agent_view*;
  check_agent_view;
end do;

when received (**nogood**, x_j, *nogood*) **do** — (ii)
  record *nogood* as a new constraint;
  when *nogood* contains an agent x_k that is not its neighbor
    do request x_k to add x_i as a neighbor,
      and add x_k to its neighbors; **end do**;
  old_value ← *current_value*; **check_agent_view**;
  when *old_value* = *current_value* **do**
      send (**ok?**, (x_j, *current_value*)) to x_j; **end do**; **end do**;

procedure **check_agent_view**
  when *agent_view* and *current_value* are not consistent **do**
    if no value in D_i is consistent with *agent_view* **then backtrack**;
    else select $d \in D_i$ where *agent_view* and d are consistent;
      current_value ← d;
      send (**ok?**, (x_i, d)) to neighbors; **end if**; **end do**;

procedure **backtrack**
  generate a nogood V — (iii)
  when V is an empty nogood **do**
    broadcast to other agents that there is no solution,
      terminate this algorithm; **end do**;
  select (x_j, d_j) where x_j has the lowest priority in a nogood;
  send (**nogood**, x_i, V) to x_j;
  remove (x_j, d_j) from *agent_view*;
  check_agent_view;

Algorithm 2: ABT Procedures



ABT: Correctness and Completeness

- **Correctness**
  - silent network $\Leftrightarrow$ all constraints are satisfied
- **Completeness**
  - ABT performs an exhaustive traversal of the search space
  - Parts not searched: those eliminated by nogoods
  - Nogoods are legal: logical consequences of constraints
  - Therefore, either there is no solution $\Rightarrow$ ABT generates the empty nogood, or it finds a solution if exists

ABT: Remarks

- Fixed ordered organisation
  - Agents only communicate with agents with lower priority for *ok?*
  - Agents only communicate with the agent with direct higher priority for *nogood*
- No termination procedure is given (but it is easily implemented using Dijkstra's tokens)
- Really distributable
- What if x_0 disappears?...

Extensions and Filiation

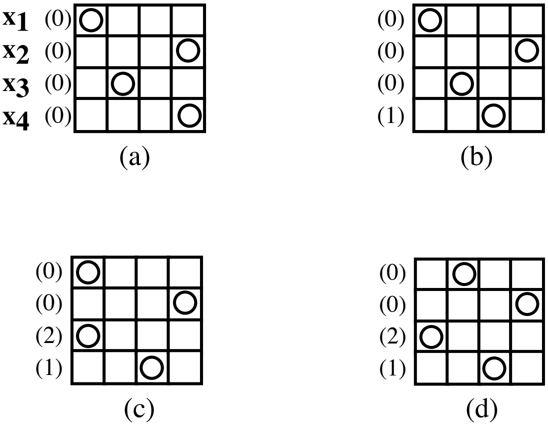
- **Changing ordering** in every conflict with AWCS [Yokoo, 2001]
- Satisfaction $\rightarrow$ **Optimisation** with ADOPT (Asynchronous B&B) [Modi et al., 2005] or APO [Mailler and Lesser, 2006]
- **Adding new agents** at runtime in DynAPO [Mailler, 2005]

Asynchronous Weak-Commitment Search (AWCS) [Yokoo, 2001]

procedure **check_agent_view**
  when *agent_view* and *current_value* are not consistent **do**
    if no value in D_i is consistent with *agent_view* **then backtrack**;
    else select $d \in D_i$ where *agent_view* and d are consistent
      and d minimizes the number of constraint violations
        with lower priority agents; — (i)
     current_value $\leftarrow d$;
     send (**ok?**, (x_i, d, *current_priority*)) to *neighbors*;
    end if; **end do**;

procedure **backtrack**
  generate a nogood V;
  when V is an empty nogood **do**
    broadcast to other agents that there is no solution,
    terminate this algorithm; **end do**;
  when V is a new nogood **do** — (ii)
    send V to the agents in the nogood;
    current_priority $\leftarrow 1 + p_{max}$,
      where p_{max} is the maximal priority value of neighbors;
    select $d \in D_i$ where *agent_view* and d are consistent,
     and d minimizes the number of constraint violations
     with lower priority agents;
    current_value $\leftarrow d$;
    send (**ok?**, (x_i, d, *current_priority*)) to neighbors; **end do**;

Algorithm 3: AWCS Procedures



Distributed Local Search Approaches

Local Search (LS)

- *LS* algorithms explore the search space from state to state
- Always tend to improve the current state of the system
- Can naturally handle dynamics (adding constraints, changing values)
- Time efficient
- Not complete and require some subtle parameter tuning

choose an initial assignment $s(0)$
while $s(t)$ *not terminal* **do**
  select an acceptable move $m(t)$ to another assignment
  apply move $m(t)$ to reach $s(t+1)$
  $t := t+1$
end

Algorithm 4: A generic centralised local search algorithm

Classical Centralised LS Algorithms

Common points

- Initial point (ex: randomly chosen)
- Termination criterion (ex: limit time, δ improvement)
- Acceptable move (ex: $+\epsilon$)

Famous LS Methods

- **Tabu search** [GLOVER and LAGUNA, 1997]
- **Simulated annealing** [KIRKPATRICK et al., 1983]
- **Iterative Breakout method** [MORRIS, 1993]

Distributed Breakout Algorithm (DBA)

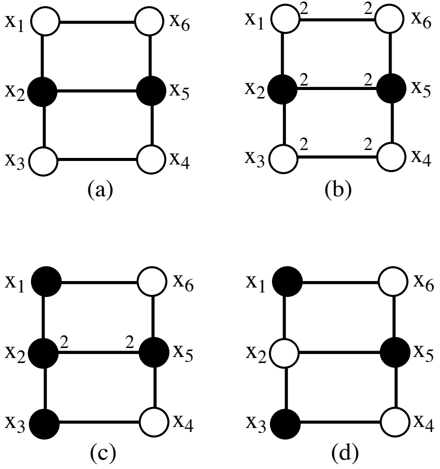
wait_ok? mode — (i)
when received (**ok?**, x_j, d_j) **do**
  add (x_j, d_j) to *agent_view*;
  when received ok? messages from all neighbors **do**
    send_improve;
    goto wait_improve mode; **end do**;
  goto wait_ok mode; **end do**;

procedure **send_improve**
  current_eval ← evaluation value of *current_value*;
  my_improve ← possible maximal improvement;
  new_value ← the value which gives the maximal improvement;
  send (**improve**, x_i, *my_improve*, *current_eval*) to neighbors;

wait_improve? mode — (ii)
when received (**improve**, x_j, *improve*, *eval*) **do**
  record this message;
  when received improve? messages from all neighbors **do**
    send_ok; clear *agent_view*;
    goto wait_ok mode; **end do**;
  goto wait_improve mode; **end do**;

procedure **send_ok**
  when its improvement is largest among neighbors **do**
    current_value ← *new_value*; **end do**;
  when it is in a quasi-local-minimum **do**
    increase the weights of constraint violations; **end do**;
  send (**ok?**, x_i, *current_value*) to neighbors;

Algorithm 5: DBA Message Handler



Distributed Breakout Algorithm (DBA) (cont.)

Principles of DBA [Yokoo, 2001]

- Distribution difficulties:
  - *(i)* if two neighbouring agents concurrently change their value, the *system may oscillate*
  - *(ii)* detecting the fact that the whole system is trapped in local minimum requires the agents to *globally exchange data*
- DBA answers:
  - *(i)* for a given neighbourhood, only the agent that can *maximally improve* the evaluation value is given the right to change its value
  - *(ii)* agents only detects *quasi-local-minimum*, which is a weaker local-minimum that can be detected only by local interactions

Distributed Breakout Algorithm (DBA) (cont.)

Remarks

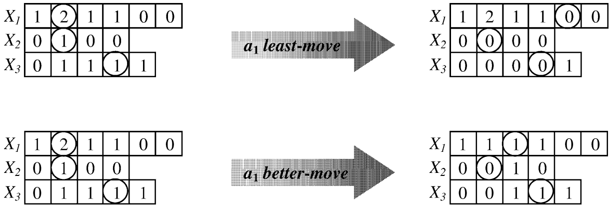
- Distributed version of the iterative breakout algorithm
- Two-mode behaviour alternating between exchange of potential improvement and exchange of assignments
- ✓ There is no order over the agents society $\rightarrow$ neighbourhoods
- The system halts if a solution is found or if the weight of constraints have reached a *predefined upper bound*
  - $\rightarrow$ the **only** difficult parameter to set
- ✗ DBA is not complete
- ✓ DBA is able to detect the termination or a global solution only by reasoning on local data.

Environment, Reactive rules and Agents (ERA) [Liu et al., 2002]

Components

- A discrete grid **environment**, that is used as a communication medium
- **Agents** that evolves in some regions of the grid (their domain)
  - Agents move *synchronously*
  - Agents cannot move in the domain of other agents, but can mark it with the number of potential conflicts
  - These marks represents therefore the number of violated constraints if an agent chooses the marked cell
- **Rules** (*moves*) that agent follow to reach an equilibrium
  - 3 possible actions
    - *least-move*: the next cell is the one with minimum cost
    - *better-move*: the next cell is randomly chosen and if it has less conflicts than the actual one the agent moves else the agent rests
    - *random-move*: the next cell is randomly chosen
  - A decision consists in a random Monte-Carlo choice of the action to perform

$t \leftarrow 0$
initialise the grid to 0 violation in each cell; **foreach** *agent i* **do**
  randomly move to a cell of row *i*
end
while $t < t_{max}$ *and no solution* **do**
  foreach *agent i* **do**
    select a move behaviour
    compute new position
    decrease markers in all cells with past violations



Environment, Reactive rules and Agents (ERA) [Liu et al., 2002] (cont.)

Remarks

- The environment is the communication medium
  - ✓ There is no asynchronous mechanisms and message handling
  - ✗ Synchronisation point: high synchronous solving process with no benefit from distribution, in case of high connected constraint networks
- ✓ ERA quickly finds assignments close to the solution $\rightarrow$ repairing issues
- ✗ Redundant usage of random choices: non-guided method, close to random walk, and non complete
- ✗ Termination: ERA requires a time limit (t_{max}) (problem-dependant)

Panorama

Algorithm	**Type**	**Memory**	**Messages**	**Remarks**
ABT	CSP	Exponential	–	Complete, Static ordering
AWCS	CSP	Exponential	–	Complete (only with exponential space), Reordering, fast
DBA	Max-CSP	Linear	Bounded	Incomplete, Fast
ERA	Max-CSP	Polynomial	n/a	Incomplete, randomness

Table: DCSP and DCOP algorithms

Using Distributed Problem Solving

Problem and Environment Characteristics

- Geographic distribution
  - ex: agents are physically distributed, and solving the whole problem is not possible in a centralised manner
- Constraint network topology
  - ex: bounded vertex degrees or large constraint graph diameter
- Knowledge encapsulation
  - ex: *privacy* preserving, limited knowledge
- Dynamics
  - ex: rather than solving the whole problem again, only repair sub-problems

Some Applications

- Frequency assignment
- Scheduling
- Resource allocation, Manufacturing control
- Supply chain
- ...

References

DECHTER, R. (2003). *Constraint Processing*. Morgan Kaufmann.

GLOVER, F. and M. LAGUNA (1997). *Tabu Search*. Kluwer.

KIRKPATRICK, S., C. GELLAT, and M. VECCHI (1983). "Optimization by Simulated Annealing". In: *Science* 220.4598, pp. 671–680.

LIU, J., H. JING, and Y. Y. TANG (2002). "Multi-agent Oriented Constraint Satisfaction". In: *Artificial Intelligence* 136.1, pp. 101–144.

MAILLER, R. (2005). "Comparing two approaches to dynamic, distributed constraint satisfaction". In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*. ACM Press, pp. 1049–1056.

MAILLER, R. and V. R. LESSER (2006). "Asynchronous Partial Overlay: A New Algorithm for Solving Distributed Constraint Satisfaction Problems". In: *Journal of Artificial Intelligence Research* 25, pp. 529–576.

MODI, P. J., W. SHEN, M. TAMBE, and M. YOKOO (2005). "ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees". In: *Artificial Intelligence* 161.2, pp. 149–180.

MORRIS, P. (1993). "The Breakout Method for Escaping from Local Minima". In: *AAAI*, pp. 40–45.

YOKOO, M. (2001). *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer.