IRAD: Analyzing ULS System Behavior and Evolution through Agent-based Modeling and Simulation

Andres Diaz Pace – adiaz@sei.cmu.edu
Joseph Giampapa – garof@sei.cmu.edu
Mark Klein – mk@sei.cmu.edu
John Goodenough – jbg@sei.cmu.edu

31 March 2010

© 2010 Carnegie Mellon University

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

© 2010 Carnegie Mellon University

Agenda

Purpose:

- Become familiar with distributed problem-solving algorithms
  - D-COP algorithms as “toolbox” for agent-based modeling

From CSP to DisCSP to DCOP

- Basic concepts and techniques of constraint satisfaction problems (CSP)
- Extension of CSP to a distributed setting using agents
- Main algorithms for distributed CSP (DisCSP)
- Extension of DisCSP to distributed constraint optimization (DCOP)
- Main algorithms for DCOP
- Tradeoffs in DCOP

© 2010 Carnegie Mellon University

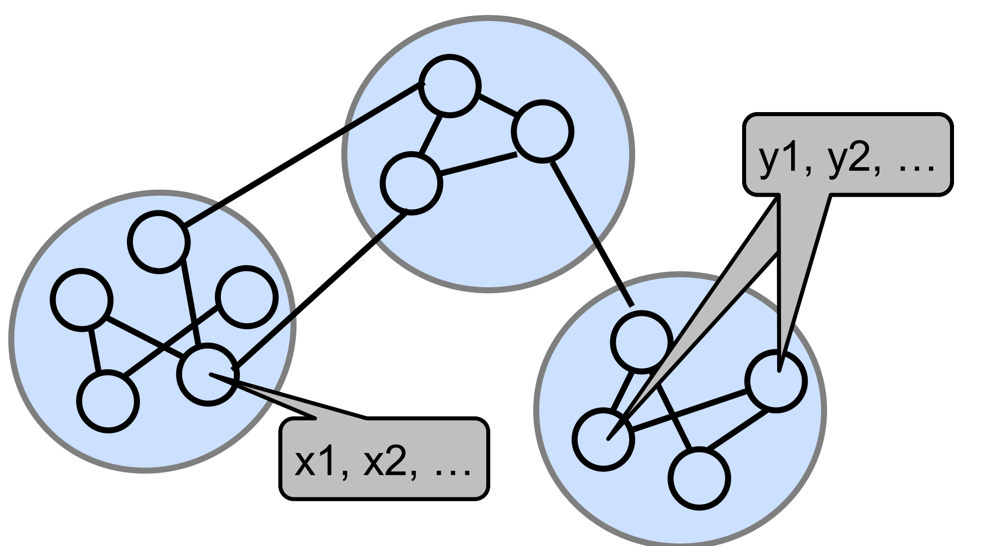
Why CSP + Agents?

An agent is an autonomous entity with a decision-making capability

- An agent manipulates a set of variables to reach some objective
- For now, we assume that agents are cooperative

CSP is a problem-solving framework for determining the values of a set of variables, subject to constraints on the variables

- Satisfaction of constraints involves costs/utilities (optimization)



Inter-related agent communities resonate with SoS/ULS ideas

© 2010 Carnegie Mellon University

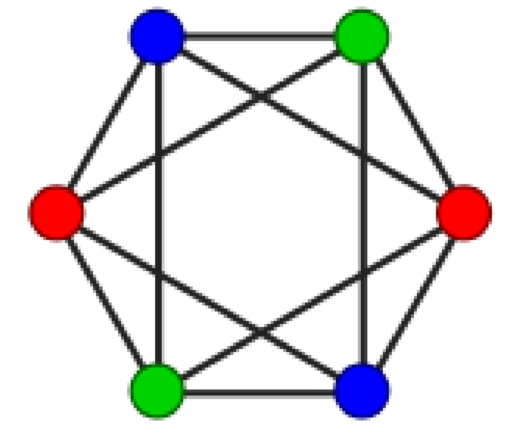
Approaches to Distributed Agent Assignment

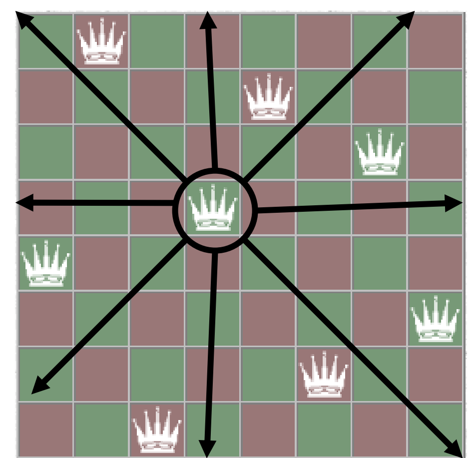
1. Constraint satisfaction
   - Find a value assignment for all the variables that fulfills all constraints

2. Constraint optimization
   - Find a value assignment for all the variables that optimizes an objective function $f \rightarrow R$
     – maximize utility or minimize cost
   - In practice, large agent networks lead to **tradeoffs**
     – find a quasi-optimal solution quickly
     – find an optimal solution sharing as little information as possible
     – find an optimal solution with little communication overhead (among agents)
     – find an optimal solution with fixed memory sizes in each agent
     – …

© 2010 Carnegie Mellon University

Types of Problems





N-queens

Graph coloring

Meeting scheduling

Several real-life problems can be modeled as constraint satisfaction or constraint optimization problems

- Sensor networks
- Robot patrolling
- Distributed resource allocation
- Distributed planning
- …

© 2010 Carnegie Mellon University

The Path to D-COP

Technique	Problem type	Planning Algorithm	Knowledge
CSP	Satisfaction • hard constraints	Centralized	Total knowledge in a single agent
Dis-CSP	Satisfaction • hard constraints	Distributed • agents • synchronous or asynchronous	• Knowledge distributed among agents • Each agent knows its neighborhood •Partial centralization is possible
DCOP	Optimization • soft constraints • objective function to be maximized/minimized	Distributed • agents • asynchronous	• Knowledge distributed among agents • Each agent knows its neighborhood •Partial centralization is possible

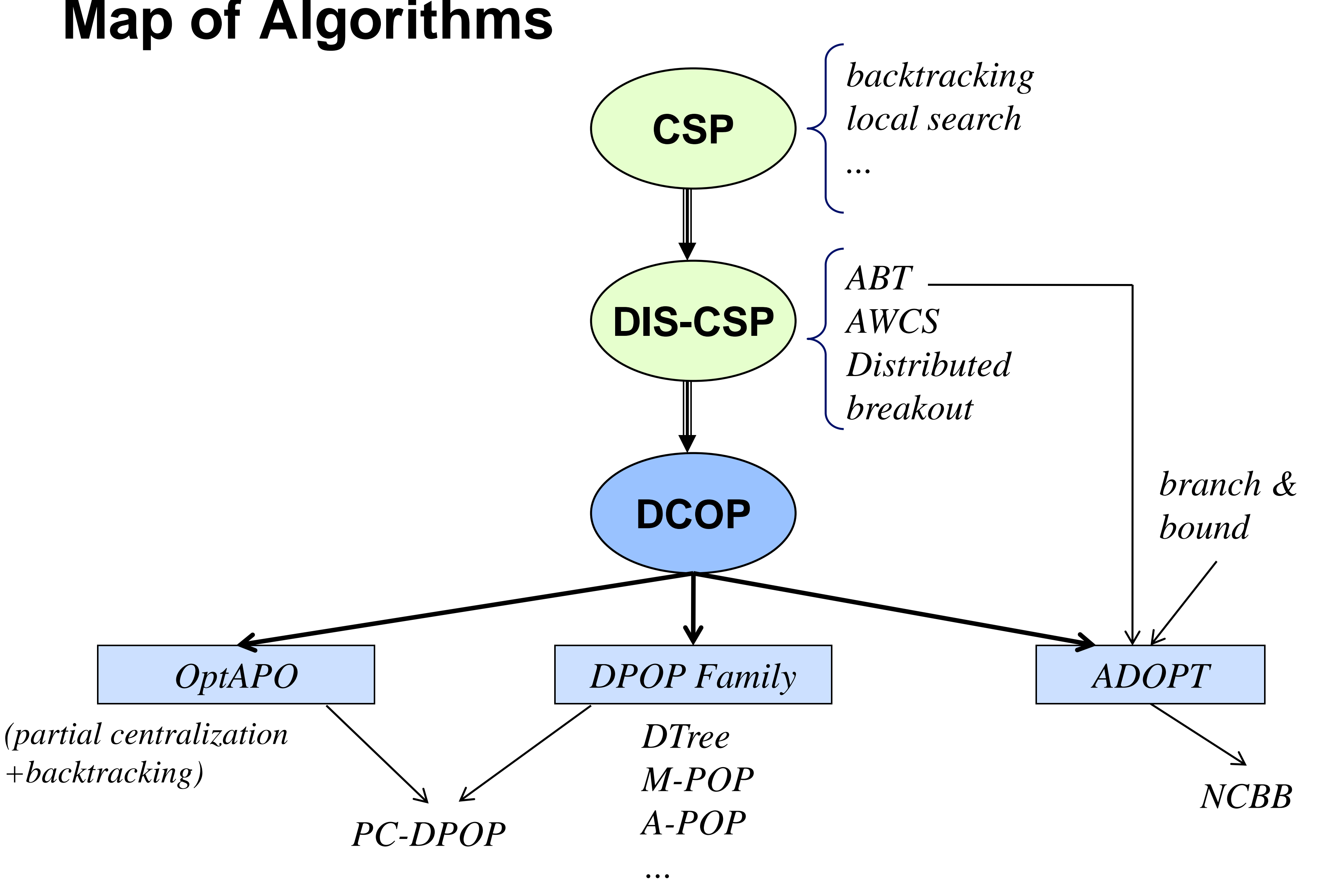
© 2010 Carnegie Mellon University

Features in D-COP Algorithms

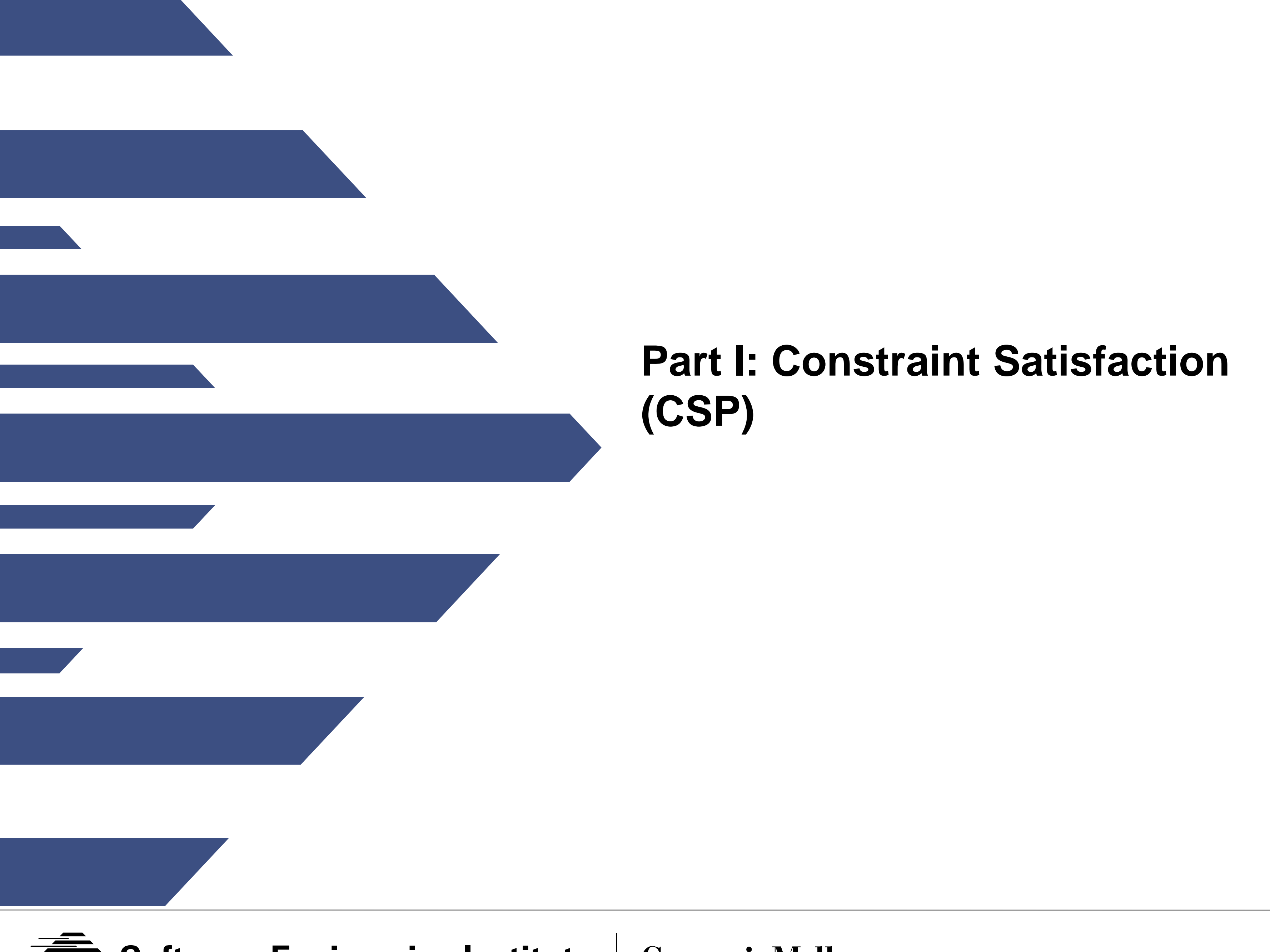
	CSP	DIS-CSP	DCOP
Control & degree of distribution	n/a	X	X
• Partially-centralized control, distributed control	--	x	x
• Priority schema	--	x	x
Constraint graph structure	X	X	X
• Dense versus sparse graphs	--	x	x
• Tree-shaped graphs (e.g., DAGs, or DFS-induced ordering)	x	x	x
– If cycles, translate graph to a cycle-free form			
– No constraints among siblings			
Communications	n/a	X	X
• Number of messages sent, size of messages	--	x	x
• Links to neighboring agents	--	--	x
Quality of solutions	X	X	X
• Complete versus incomplete (heuristic) algorithms	x	x	x
• Bounds on solution cost	--	--	x

© 2010 Carnegie Mellon University

Map of Algorithms



© 2010 Carnegie Mellon University



Part I: Constraint Satisfaction (CSP)

© 2010 Carnegie Mellon University

CSP Formulation

D(X2) = { a, d, e, …}

D(X1) = { a, b, c, …}

C1(X1,X2)

X2 ← ?

C3(X2,X3)

D(X3) = { b, e, f, …}

X1 ← ?

C2(X2,Xn)

X3 ← ?

C(Xi, Xj) = Xi ≠ Xj
(true/false)

Cm(X1,Xn)

Xn ← ?

D(Xn) = { g, … }

set of **variables** X1, X2, …, Xn

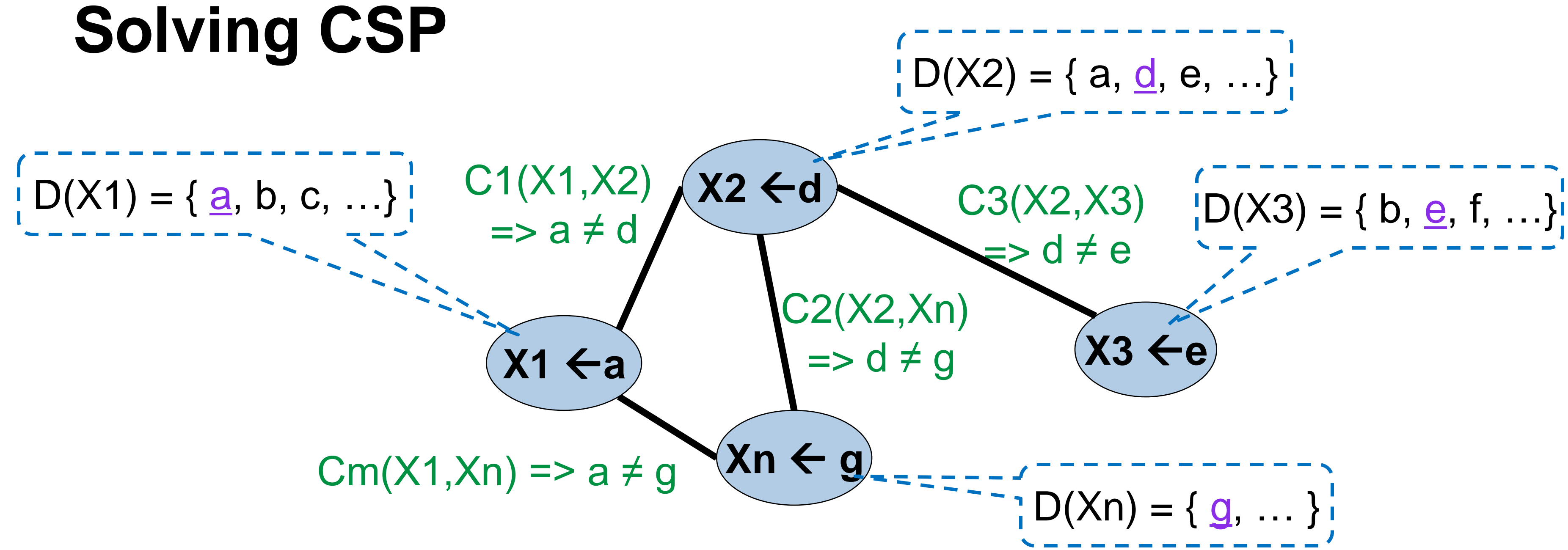
each variable has a non-empty **domain** D(Xi) of possible **values** (usually discrete variables with finite domains)

set of **constraints** C1, C2, …, Cm
each constraint involves some subset of variables and specifies allowable combinations of values for that subset

The constraints define a **constraint network**

© 2010 Carnegie Mellon University

Solving CSP



state of the problem: **assignment** of values to some (or all) variables

- consistent: an assignment that does not violate any of the constraints
- complete: an assignment that mentions all the variables

A **solution** is an assignment that is both consistent and complete

Finding a solution involves **search** (in the domain space)

© 2010 Carnegie Mellon University

Centralized CSP Approach

Traditionally, there is **one central agent** for solving the problem that

- Knows about all the variables, domains, and constraints of the problem
- Executes an algorithm to find a solution

Main technique: Backtracking

- Depth-first search for possible assignments of variables to values
- Start with a valid assignment for one variable, and progressively add assignments to variables without violating any constraint

© 2010 Carnegie Mellon University

Complexity of Solving CSP

Related to the structure of constraint graph

- Tree-structured problems can be solved in linear time (in # variables)
- Strategy: try to reduce problems to trees

1) Remove nodes

- assign values to variables so that the remaining nodes form a tree.
- cycle cutset (when the graph is nearly a tree)
- in general, finding the smallest cycle cutset is NP-hard (heuristics exist)

2) Collapse nodes together

- construct a "tree decomposition" of the constraint graph into a set of connected components (sub-problems), and solve each sub-problem independently
- each sub-problem should be as small as possible (tree width)
- finding a good tree decomposition is NP-hard (heuristic exist)

© 2010 Carnegie Mellon University

Note: “Binarization” of Constraints

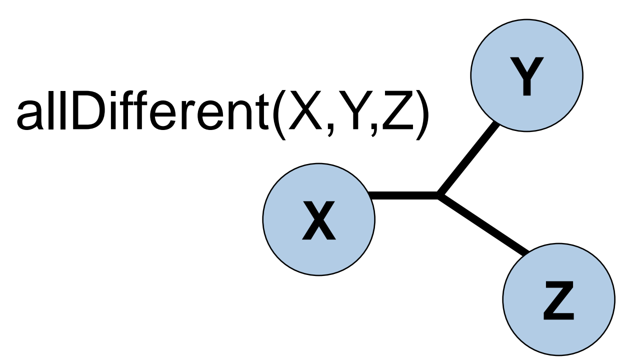
A constraint can affect any number of variables

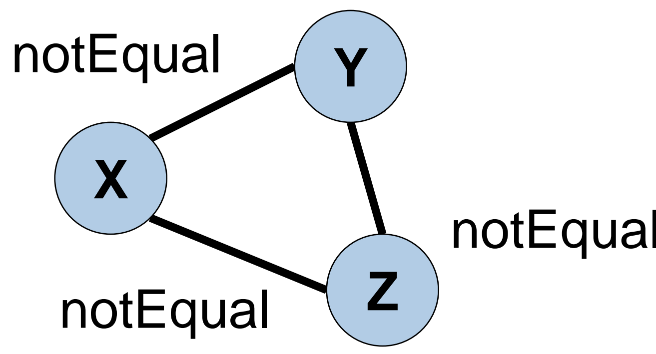
If all the constraints are binary, the variables and constraints can be represented in a **constraint graph**

- CSP algorithms can exploit graph search techniques

A constraint of higher arity can be expressed in terms of binary constraints. Hence, binary CSPs are representative of all CSPs

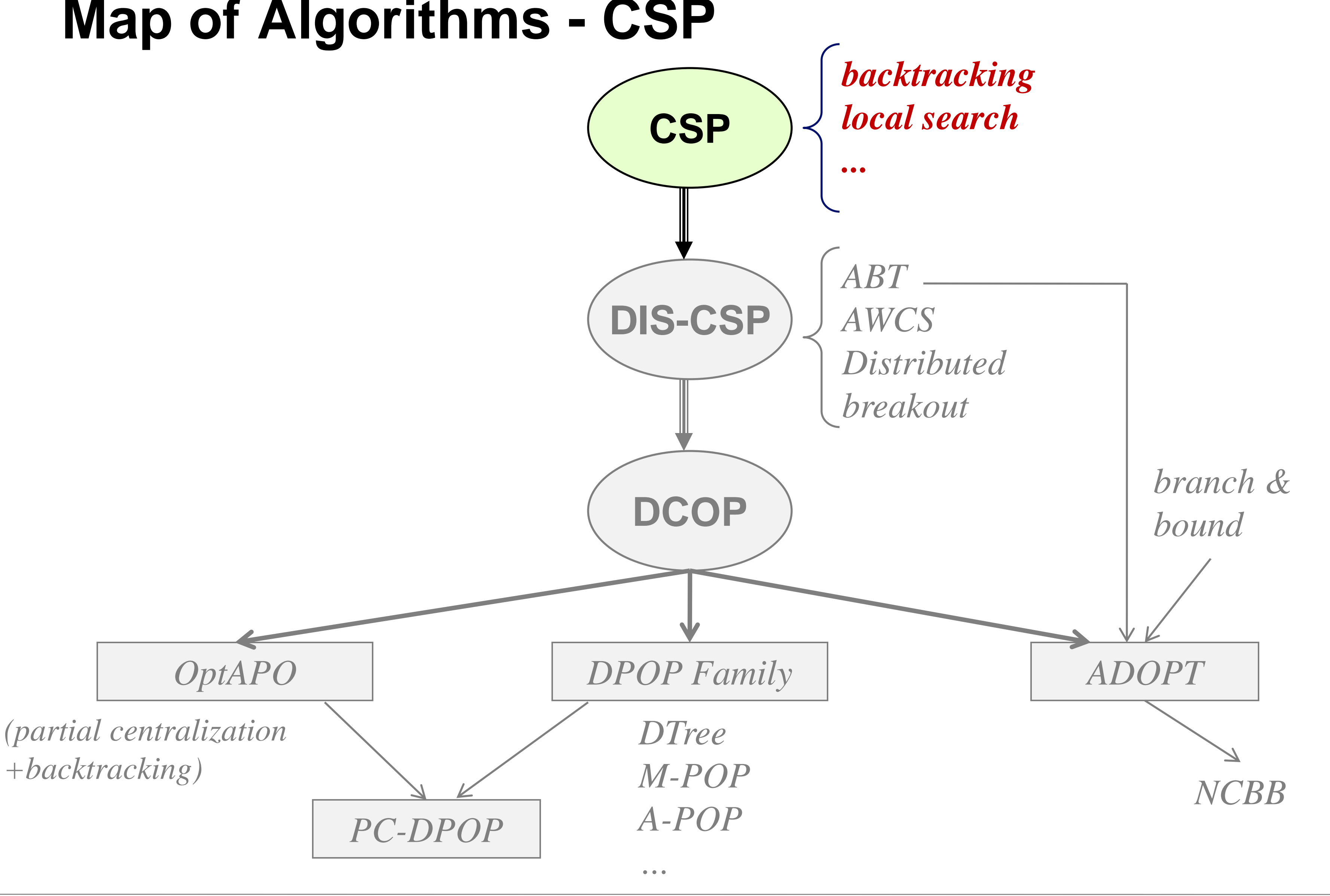
- Choice between binary or non-binary constraints depends on the algorithm and the supporting toolkit





© 2010 Carnegie Mellon University

Map of Algorithms - CSP



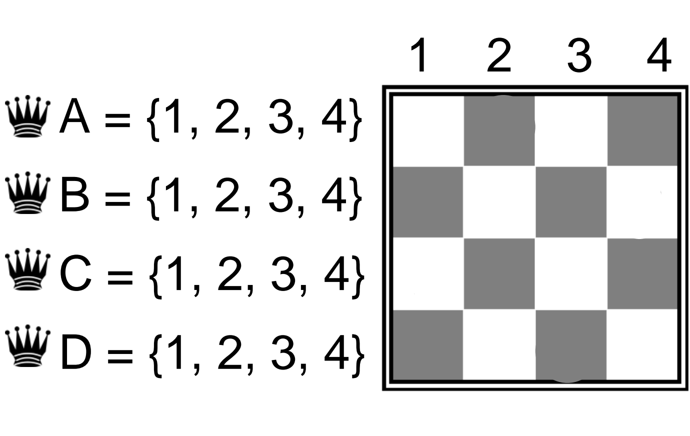
© 2010 Carnegie Mellon University

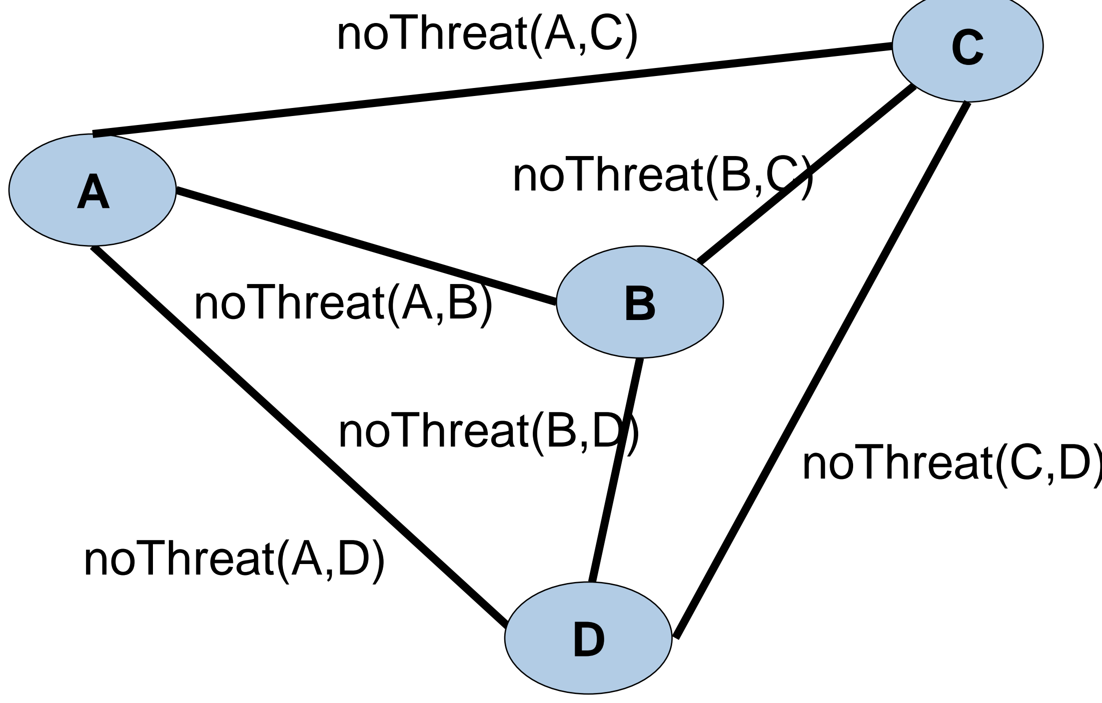
Example: 4-queens

Variables: A, B, C, D

Domain: {1, 2, 3, …, N} N = 4

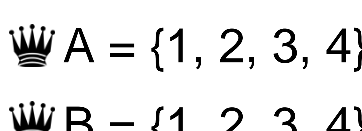
Constraints: for all pairs of queens → noThreat(Qi, Qj)





© 2010 Carnegie Mellon University

4-queens: Solutions

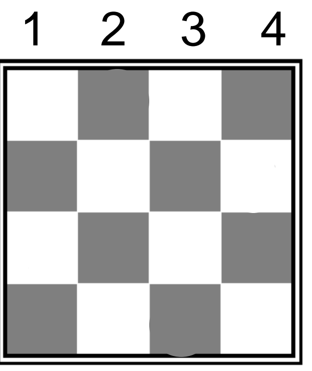


A = {1, 2, 3, 4}

B = {1, 2, 3, 4}

C = {1, 2, 3, 4}

D = {1, 2, 3, 4}

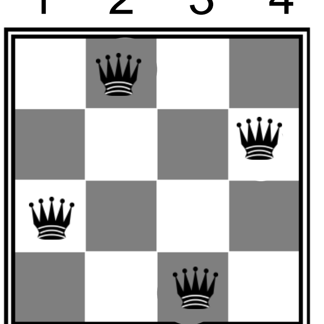


A = {2}

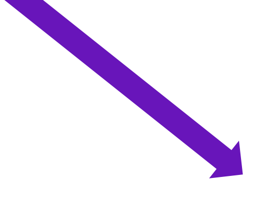
B = {4}

C = {1}

D = {3}



legal solution



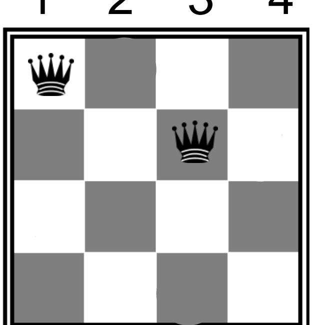
no solution

A = {1}

B = {3}

C = X

D = X

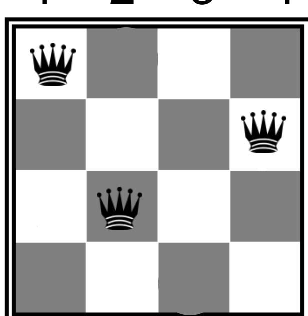


A = {1}

B = {3}

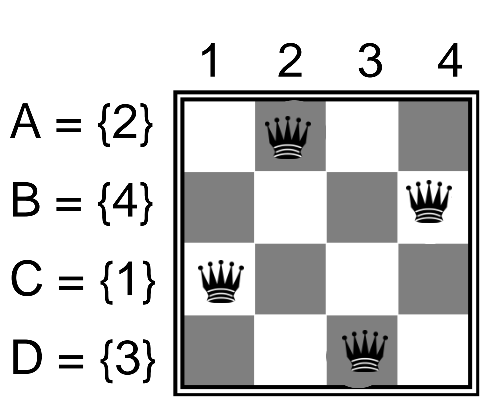
C = {2}

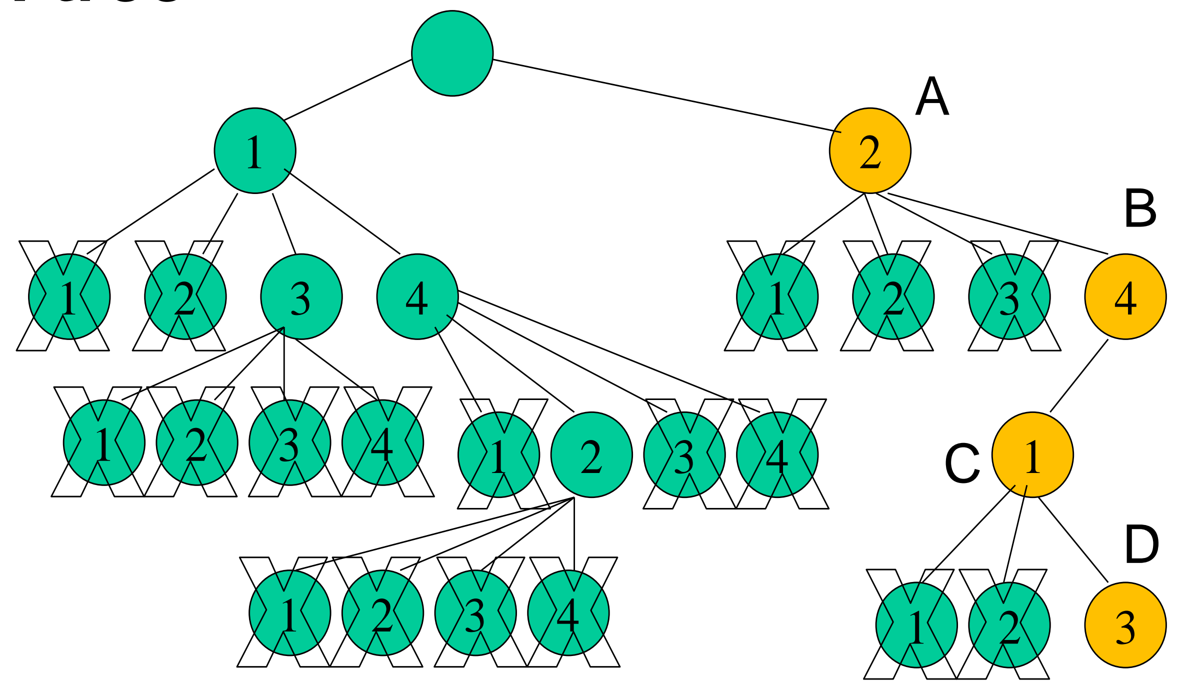
D = X



© 2010 Carnegie Mellon University

4-queens: Search tree





Backtracking algorithm

- Start with a “partial solution” that assigns values to some variables in such a way it satisfies the constraints
- Expand the partial solution by adding variables one by one, until you have a complete solution
- Backtracking operation: when the value of variable doesn’t satisfy a constraint, change the value. If no values are left, then fail.

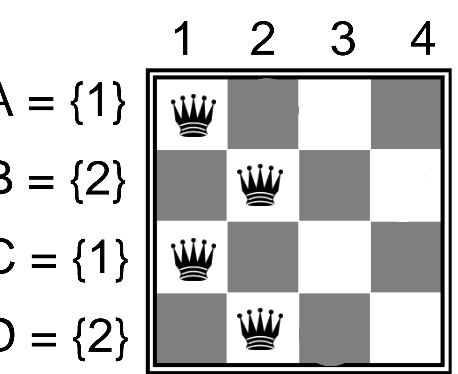
© 2010 Carnegie Mellon University

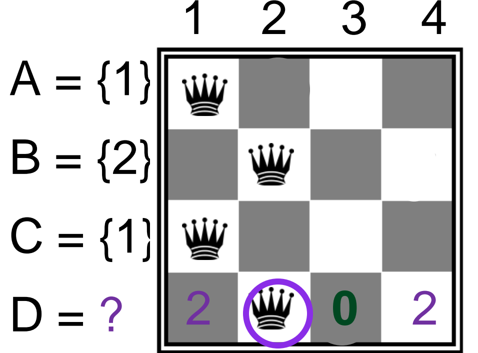
Alternative to Backtracking: Iterative search

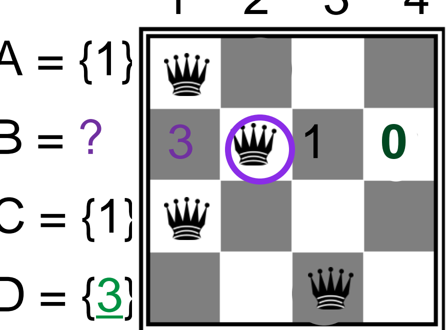
"Improve what you have until you can't make it better"

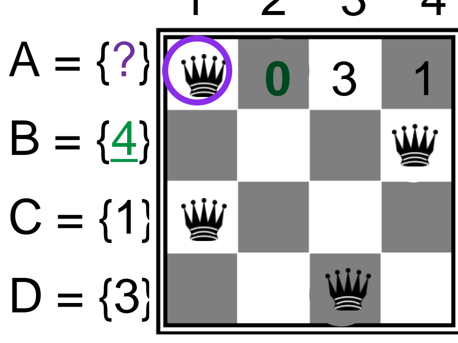
Assign a value to every variable, and try to "repair" a flawed solution and produce a valid one

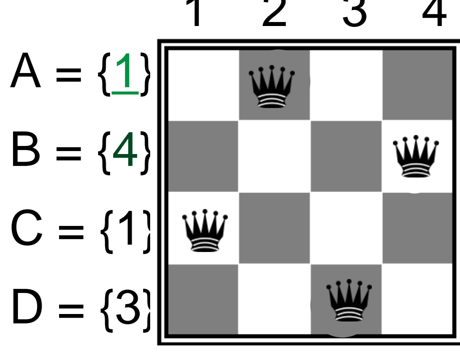
- At each step, apply local search to change the value of one variable
- Select the value that minimizes the number of violated constraints → **min-conflict heuristic**











© 2010 Carnegie Mellon University

Observations on Iterative Search

The algorithm works well, given a “reasonable” initial state

The structure of the constraint network is important

- e.g., the n-queens defines a “dense” space

Tradeoff between efficiency and completeness

- Local search is faster than a systematic search using backtracking
- but finding a valid solution cannot be guaranteed

Local search techniques can be used in a setting where the problem changes (e.g., queens, positions, dimensions of chessboard) over time

- Local search will “repair” the current solution
- not need to start search from scratch for every change

© 2010 Carnegie Mellon University

Complexity of Solving CSP

CSP is a combinatorial problem

- Time complexity is exponential in the number of variables/size of domains in the worst case

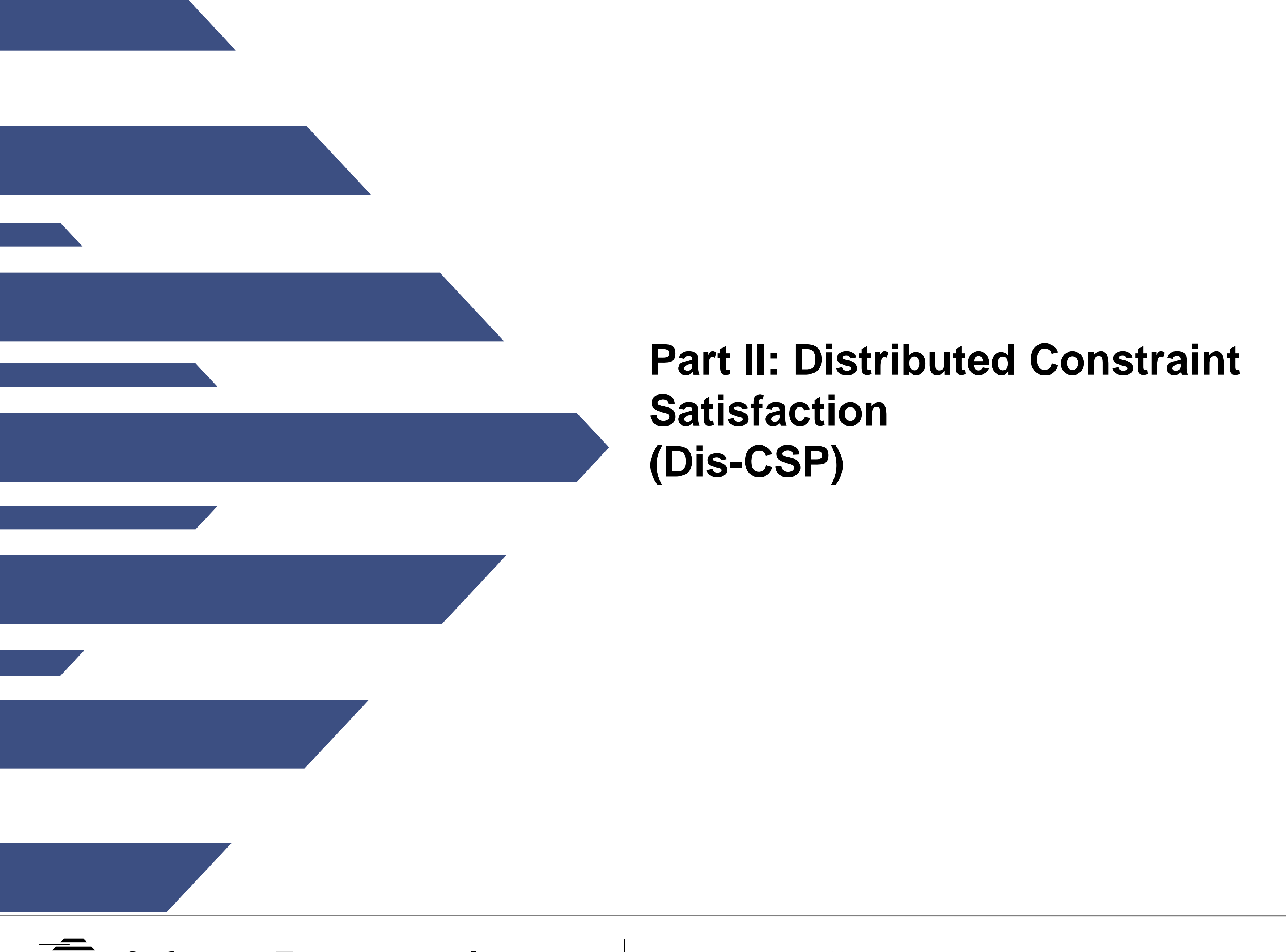
However, there are heuristics to make search more efficient

- Order the variables and their values
  - Select variable with the fewest “legal” values
  - Min-conflict: each variable has a tentative value that should satisfy as many constraints as possible
- Use consistency checks to prune infeasible values of domains

Structure of the constraint graph

- Tree-shaped problems can be solved in time linear in number of variables
- Strategy: try to reduce graphs to trees
  - Remove nodes in cycles to form “trees” (cut set)
  - Collapse nodes together into “strongly connected components”

© 2010 Carnegie Mellon University



Part II: Distributed Constraint Satisfaction (Dis-CSP)

© 2010 Carnegie Mellon University

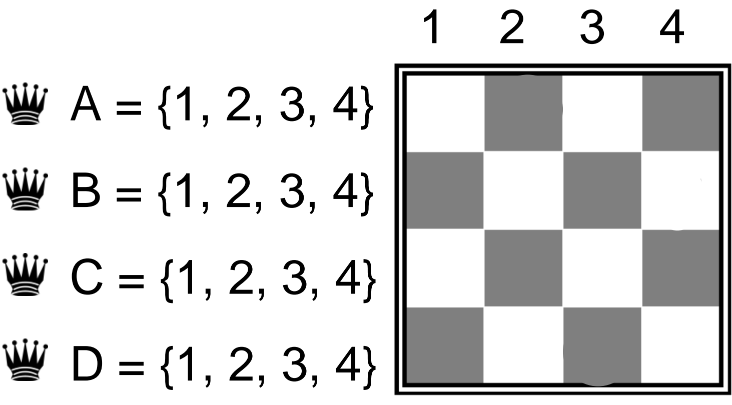
Distributed CSP (Dis-CSP)

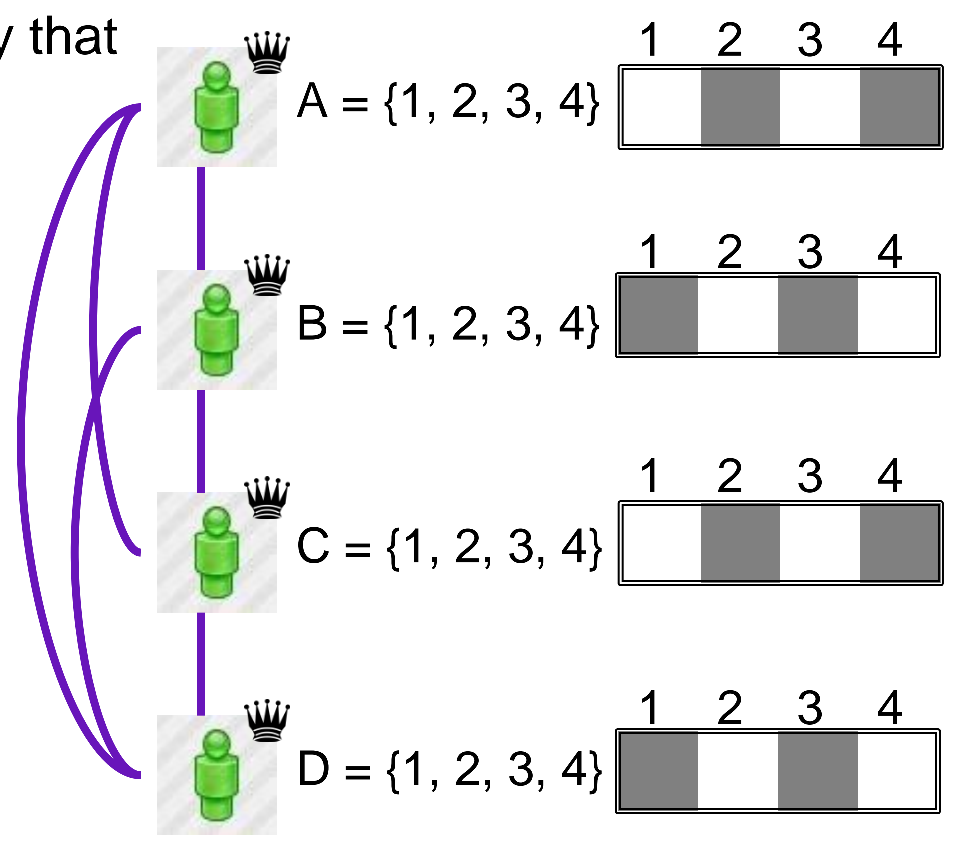
Variables/constraints are distributed among agents

- first studied by Yokoo et al. [Yokoo98]

An agent is an **autonomous** entity that

- makes “local” decisions to assign its variables
- communicates with neighboring agents via messages





© 2010 Carnegie Mellon University

Motivation behind Dis-CSP

Dis-CSP provides a framework for modeling distributed problems with constraint satisfaction characteristics

- distributed resource allocation (e.g., in a communication network)
- distributed scheduling
- sensor networks
- …

Dis-CSP (and DCOP!) can be a suitable problem-solving approach for different reasons:

- Take advantage of knowledge that is inherently distributed among agents
- Exploit potential parallelism in constraint networks that define loosely-connected sub-problems
- A central "authority" is not practical or feasible
- Privacy/security concerns
- Robustness against failures
- …

© 2010 Carnegie Mellon University

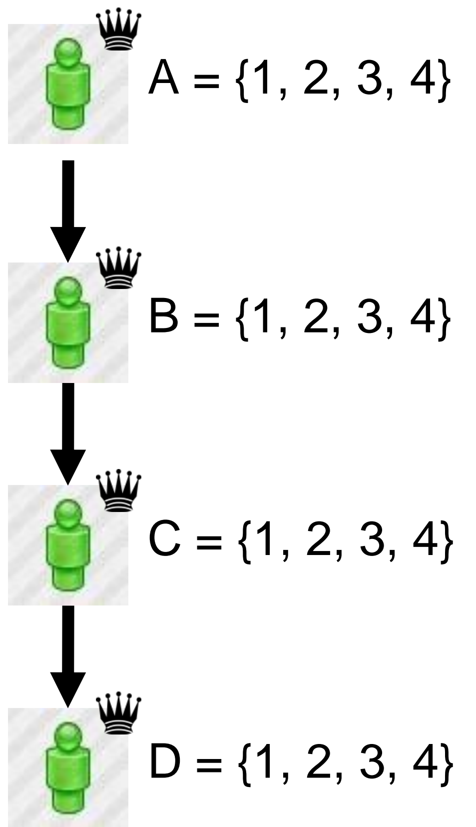
Naïve Approach – Synchronous Backtracking (ST)

Basic algorithm

- Order the agents based on a **priority schema**
  - priorities impose a total order for the agents
- "token passing" protocol
- Each agent receives a partial solution from its previous agent(s), and either
  - pass its assignment to the following agent
  - or send a "nogood" message (back to a previous agent) if no legal value is found
- The agent that receives a "nogood" message performs a backtracking step and changes its value

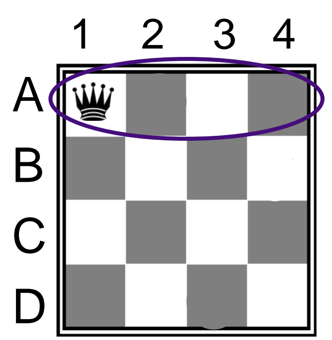
SB distributes the knowledge of the problem, but does not take advantage of parallelism

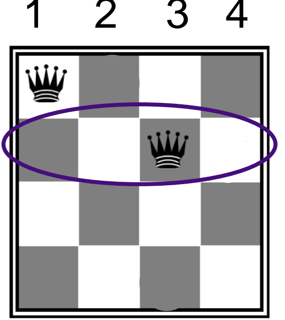
Priorities: A < B < C < D

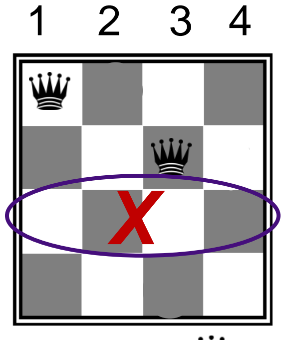


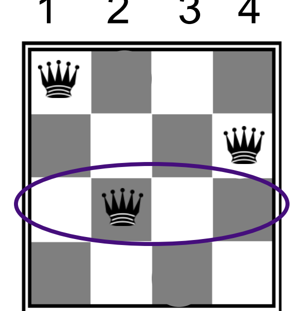
© 2010 Carnegie Mellon University

4-queens: Synchronous Backtracking

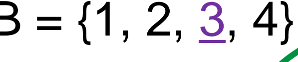


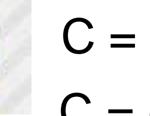






A = {1, 2, 3, 4}



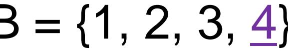


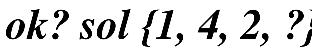
C = {1, 2, 3, 4}

C = {1, 2, 3, 4}

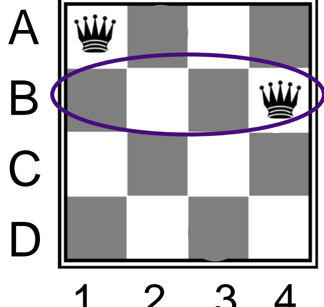
ok? sol {1, ?, ?, ?}

nogood





ok ? sol {1, 4, ?, ?}



D = {1, 2, 3, 4}

© 2010 Carnegie Mellon University

Map of Algorithms - DIS-CSP

CSP

backtracking
local search
…

DIS-CSP

ABT
AWCS
Distributed breakout

DCOP

branch & bound

OptAPO
(partial centralization +backtracking)

DPOP Family
DTree
M-POP
A-POP
…

ADOPT
NCBB

PC-DPOP

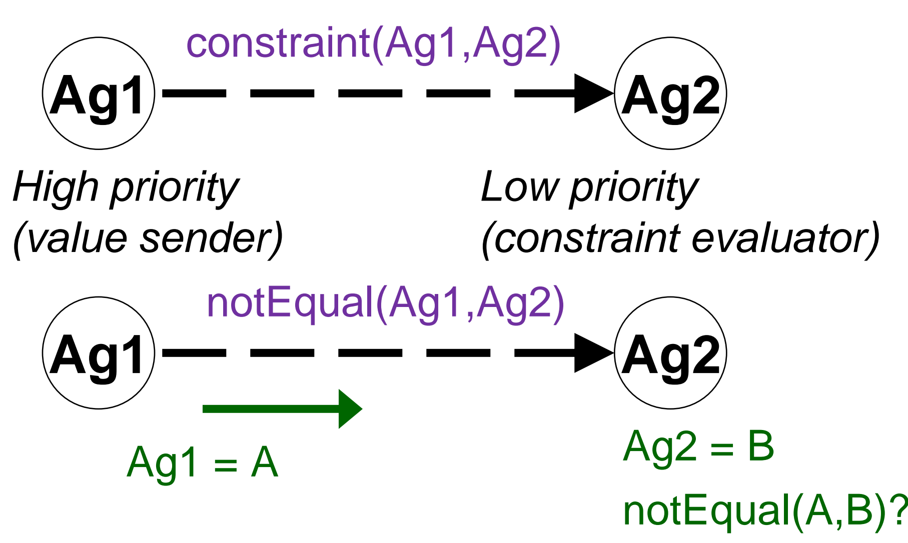
© 2010 Carnegie Mellon University

Asynchronous Backtracking (ABT)

First described in [Yokoo98]

Idea: Allow agents to run concurrently and asynchronously

- Every constraint is a **directed link** between a first agent that is assigned to the constraint and a second agent that receives the first agent's value
- Agents are assigned to priorities
  - define directionality of constraints (links) in the constraint graph
  - create a fixed hierarchy among agents



© 2010 Carnegie Mellon University

4-queens: Constraint Graph with Priorities

Variables: A, B, C, D

Domain: {1, 2, 3, 4 }

Constraints:

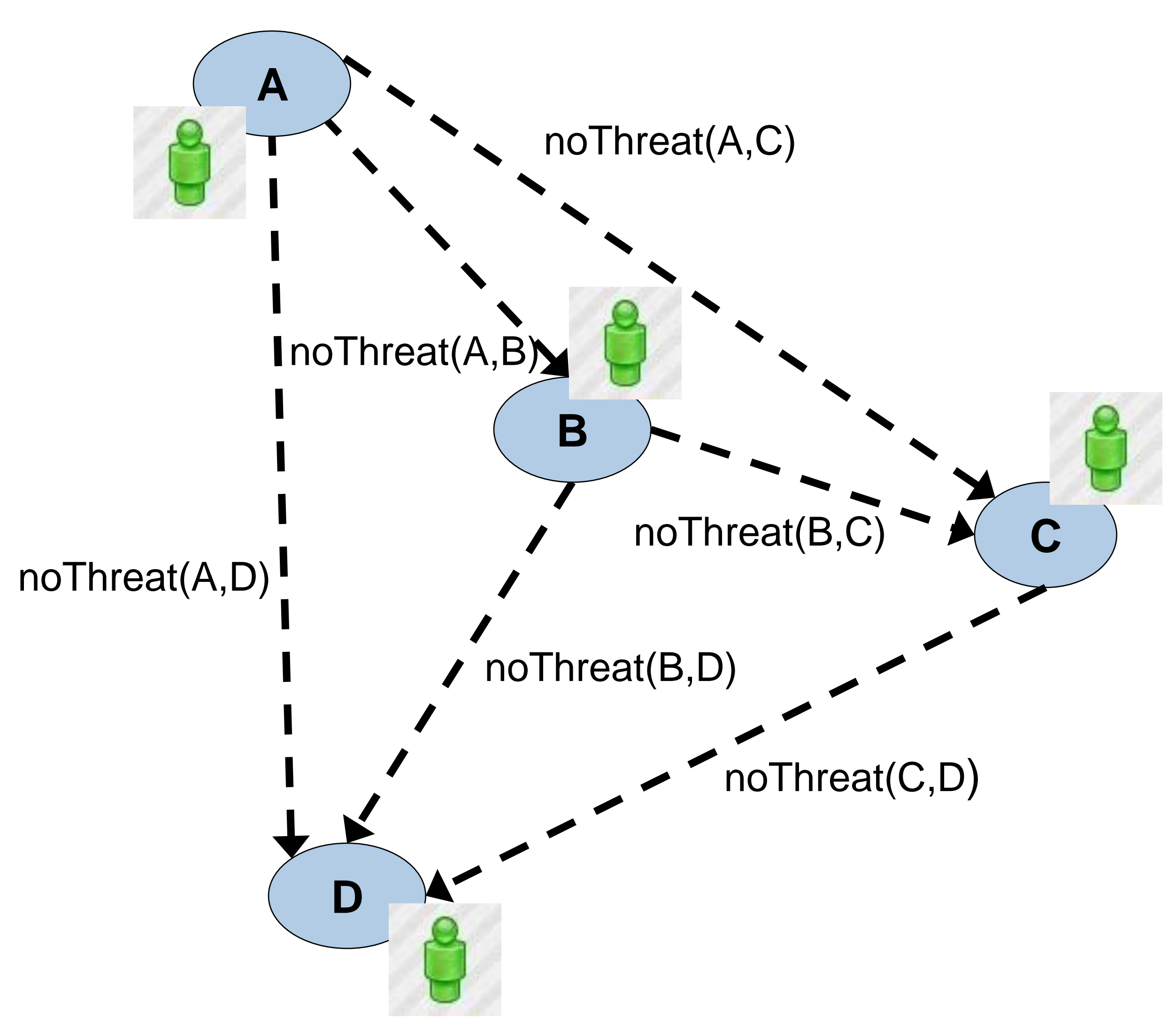
for all pairs of queens

→ noThreat(Qi, Qj)

Priority schema:

Alphabetic order of variable identifiers

A < B < C < D



© 2010 Carnegie Mellon University

Assumptions in Dis-CSP

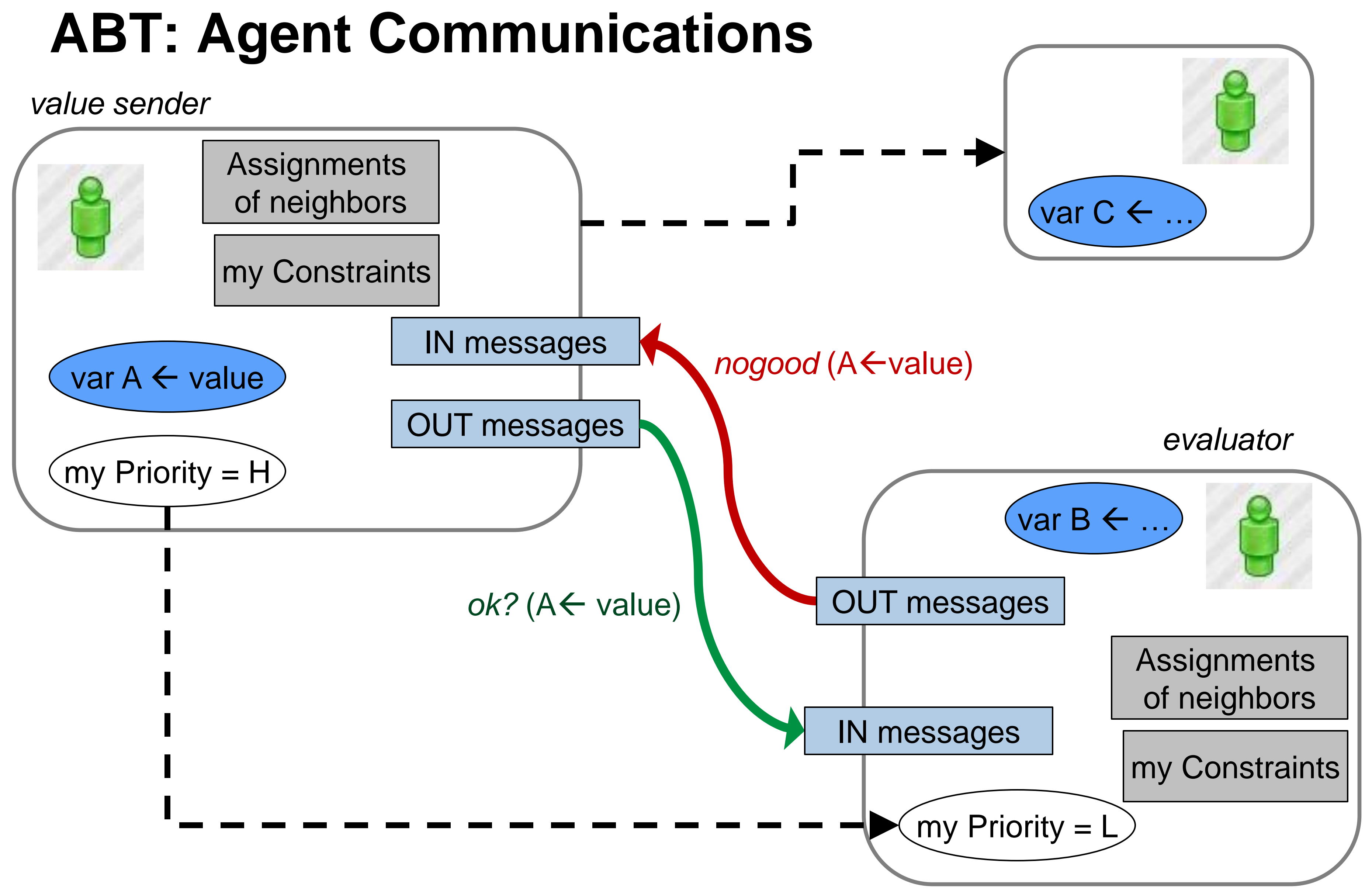
Additional assumptions made by Yokoo et al. [Yokoo98]

- An agent can send messages if it knows the addresses of the recipients
- Possible random delay in message transmission
- Messages are received in the order in which they are sent
- Each agent has exactly one variable (*)
- All constraints are binary (*)
- Each agent knows all constraint predicates relevant to its variable

(*) can be relaxed in some formulations, although at the expenses of efficiency.

© 2010 Carnegie Mellon University

ABT: Agent Communications



© 2010 Carnegie Mellon University

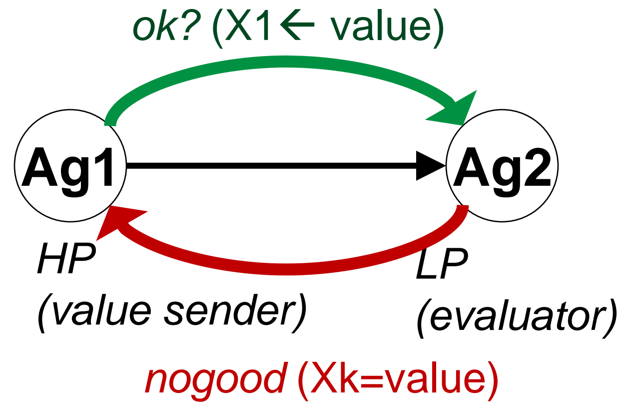
ABT: Messages and Algorithm

Types of messages

- *ok?*: a value-sending agent asks whether its assignment is acceptable
- *nogood*: a constraint-evaluating agent indicates a constraint violation
- *add_link*: request to add a new link (constraint discovered while solving the problem)

Two phases:

- Each agent instantiates its variable concurrently and sends the value to agents connected by outgoing links with an *ok?* message
- Agents wait for and respond to messages



© 2010 Carnegie Mellon University

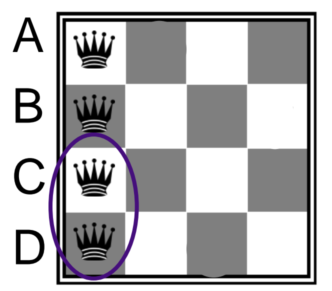
ABT: Rules for Handling Messages

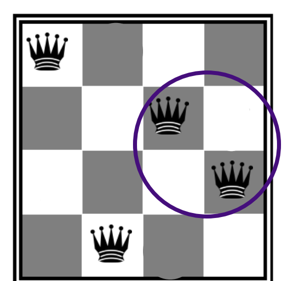
1. When an agent changes its value, it sends an *ok?* message to its (lower-priority) neighboring agents
2. An agent changes its assignment if its current value is not consistent with the assignments of higher priority agents
3. If there´s no possible assignment that is consistent with the higher priority agents, then the agent sends a *nogood* message to the higher priority agents
4. After receiving a *nogood* message, the higher priority agent tries to change its value

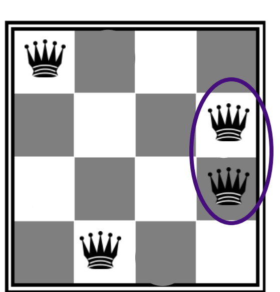
© 2010 Carnegie Mellon University

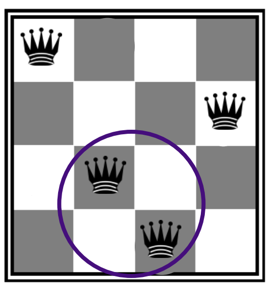
Example of ABT - 1

priorities: A < B < C < D









A = {1}

Indicates reason for move

A

B = {1}

B = {3}

B = {3}

B = {4}

A

C = {1}

C = {4}

C = {4}

B

C = {2}

C = {2}

D = {1}

A,C

D = {2}

D = {2}

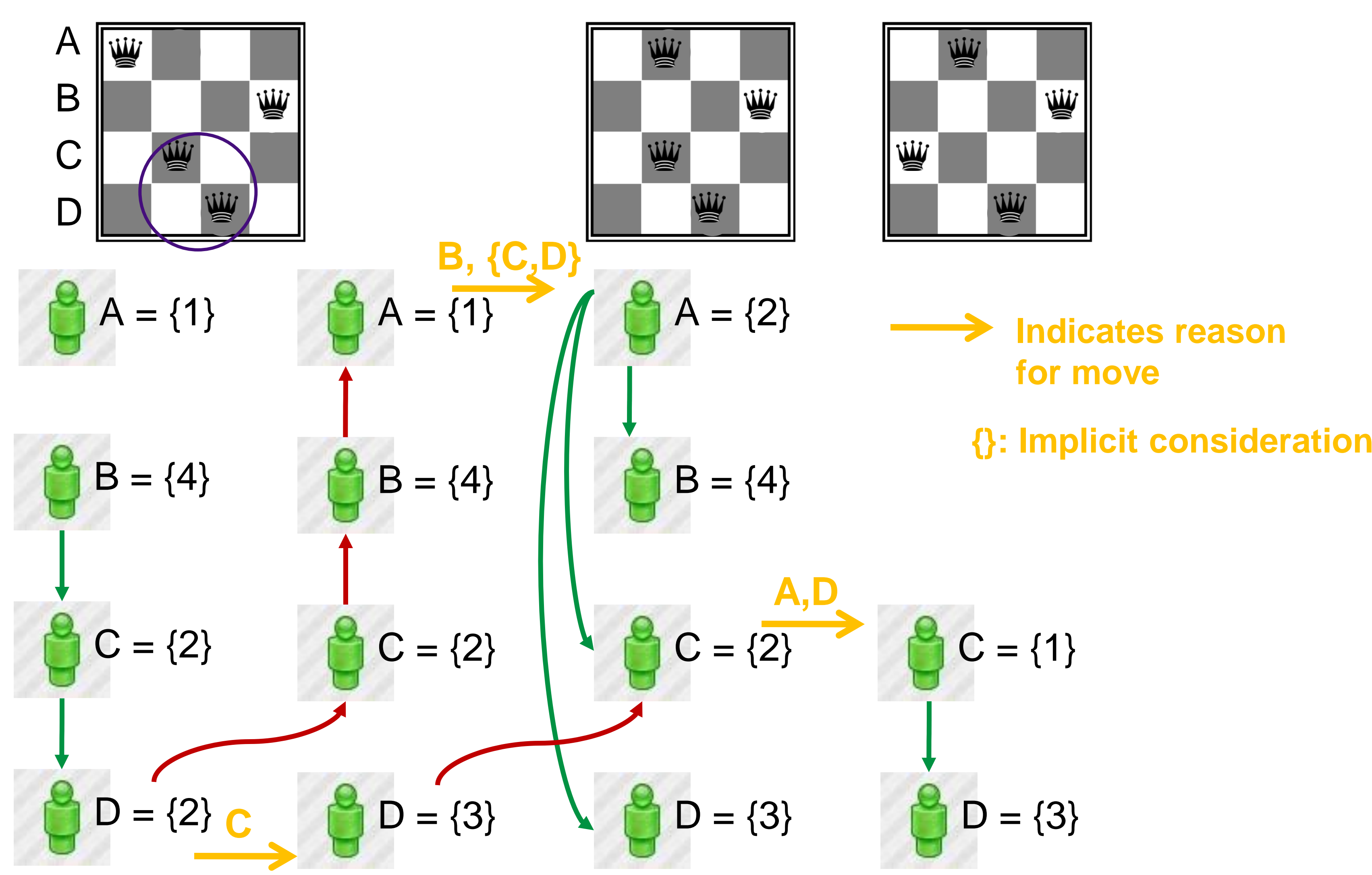
D = {2}

C

D = {3}

© 2010 Carnegie Mellon University

Example of ABT - 2

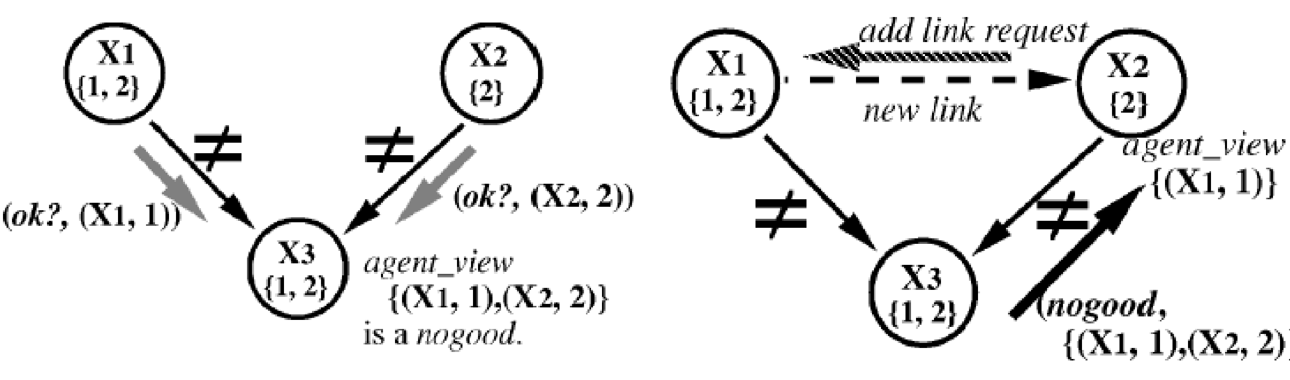


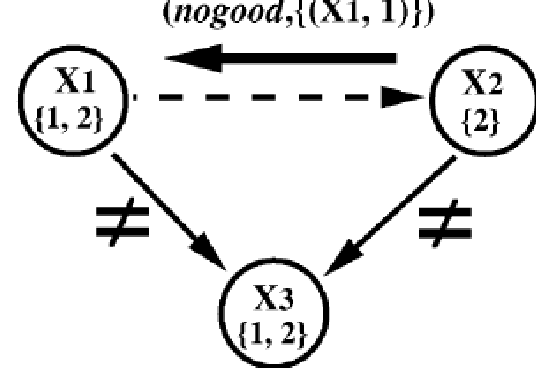
© 2010 Carnegie Mellon University

ABT: Adding New Links

A *nogood* message can be seen as a constraint derived from the original constraints

- by incorporating derived constraints, agents can avoid repeating the same mistake
- The full constraint network doesn't need to be specified at the beginning of the algorithm





High-priority agents X1 and X2 communicate their values to low-priority agent X3

Agent X3 cannot find a feasible value, so it asks their "superiors" to get an agreement

A new constraints between X2 and X1 is established

© 2010 Carnegie Mellon University

ABT: Detecting Termination Conditions

The agents will reach a **stable state** (if such state exists)

- All the assignments of variables to values satisfy all the constraints
- All the agents are waiting for an incoming message
- Determining whether the agents as a whole have reached a stable state is not contained in the ABT algorithm
  - a separate (distributed) termination algorithm is needed for this

© 2010 Carnegie Mellon University

Asynchronous Weak-Commitment Search (AWCS)

In ABT, if a high priority agent makes a bad value choice, the lower priority agents need to perform an exhaustive search to “revise” the bad decision

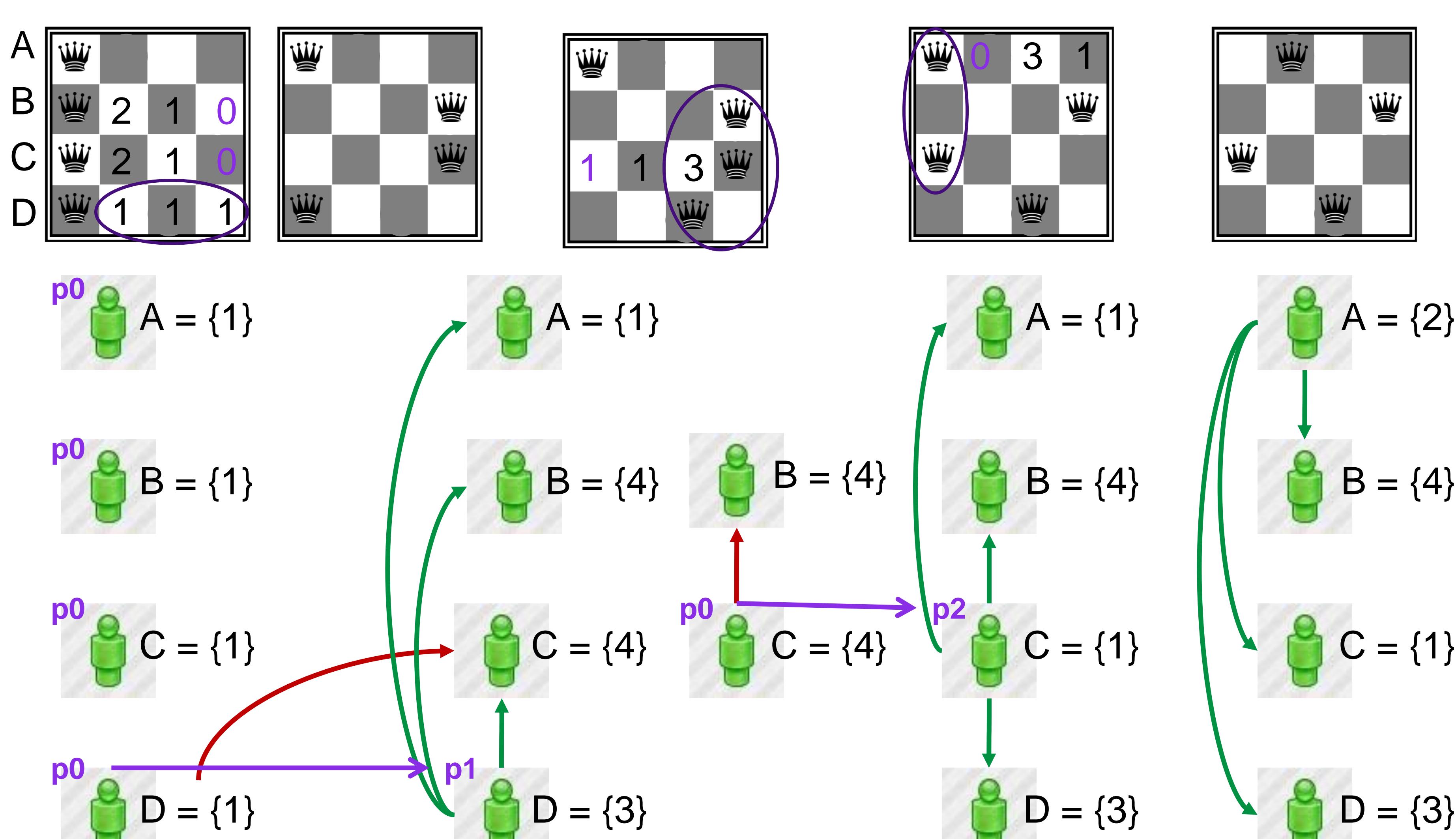
Intuition: teams with **flexible** hierarchies (i.e., a priority schema) perform better (converge quicker to a solution) than teams with **rigid** hierarchies

Idea: Use **min-conflict heuristic** (local search) in ABT for ordering values

- An agent prefers the value that minimizes the number of constraint violations with variables of lower priority agents
- Rules for allowing an agent to increment its priority at runtime
  - priorities are initially set to 0
  - a priority value is changed if and only if there’s no consistent value for the agent’s value (i.e., a new *nogood* is found)
  - the new priority value is communicated to the rest of the agents

© 2010 Carnegie Mellon University

Example of AWCS



© 2010 Carnegie Mellon University

Observations on ABT and AWCS

Empirical results show that AWCS performs better than ABT in practice

Both algorithms are complete

- Completeness: The algorithm finds a solution if one exists, and terminates with failure when there is no solution
- ABT is worst-case time exponential in the number of variables, and its space complexity depends on the number of values to be recorded for a variable (i.e., polynomial)
- AWCS is also worst-case time exponential in the number of variables, but its space complexity is exponential in the number of variables
  - In practice, the number of *nogoods* recorded by each agent can be limited to a fixed number (~ 10). If so, space complexity is improved but the theoretical completeness cannot be guaranteed

© 2010 Carnegie Mellon University

Extension: Handling Multiple Local Variables

Method 1: Each agent finds all solutions to its local problem

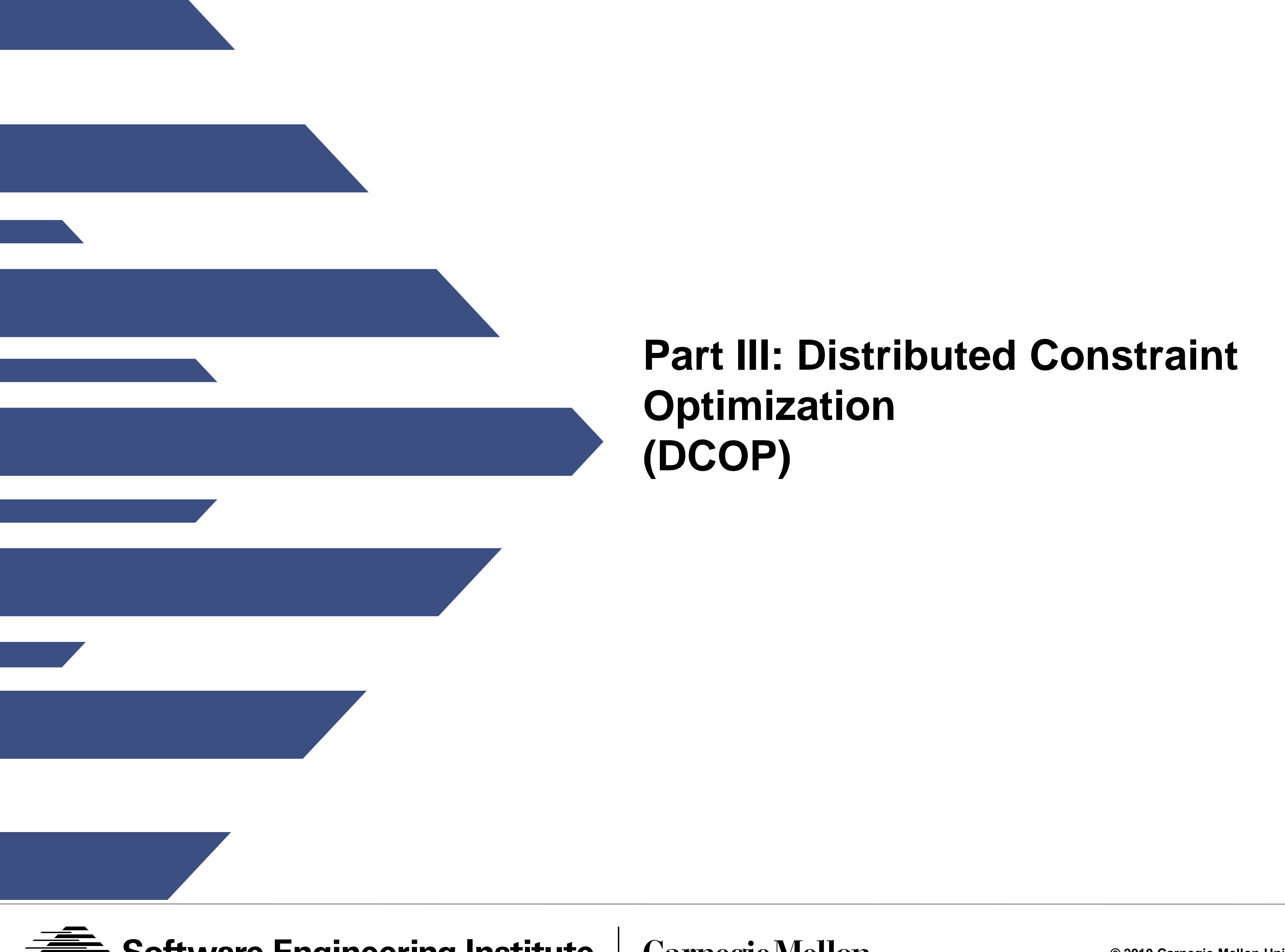
- The problem can be re-formalized as a Dis-CSP in which each agent has one local variable whose domain is the cross product of the domains of the local solutions
  - when a local problem is large/complex, finding all solutions is difficult

Method 2: An agent creates multiple virtual agents

- Each virtual agent corresponds to one local variable
- The concurrent activities of these virtual agents are simulated by the parent agent
  - simulating concurrent activities at the local level can be expensive

Both methods are neither efficient nor scalable to larger problems

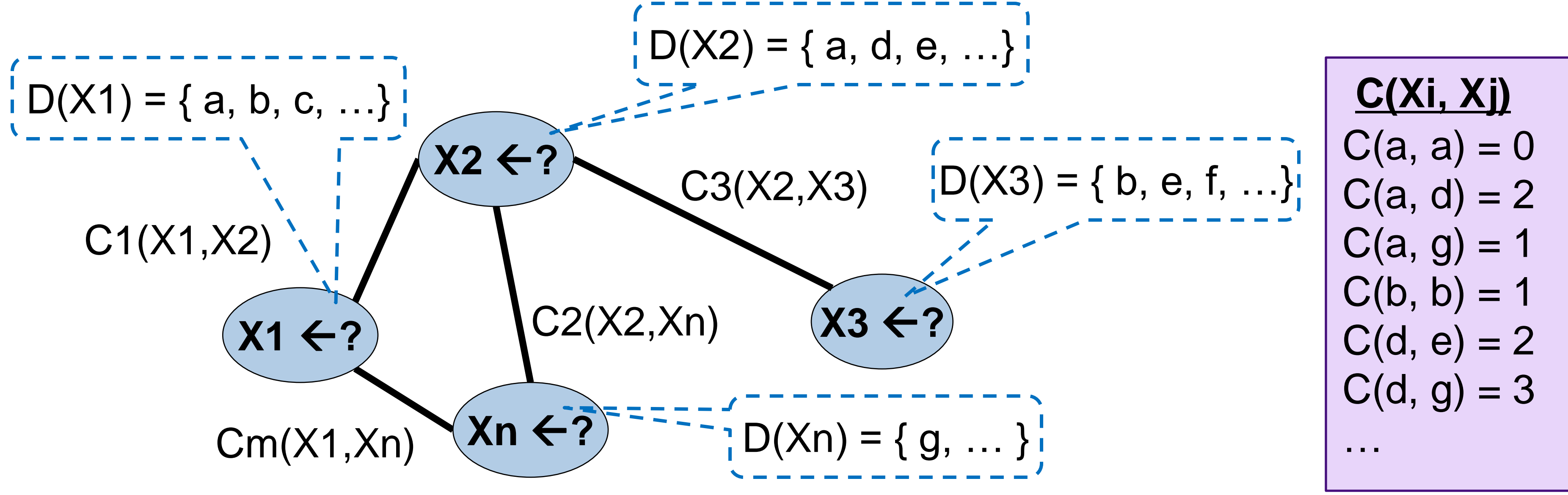
© 2010 Carnegie Mellon University



Part III: Distributed Constraint Optimization (DCOP)

© 2010 Carnegie Mellon University

Distributed Constraint Optimization (DCOP)



set of **variables** X1, X2, …, Xn

each variable has a non-empty **domain** D(Xi) of possible **values**

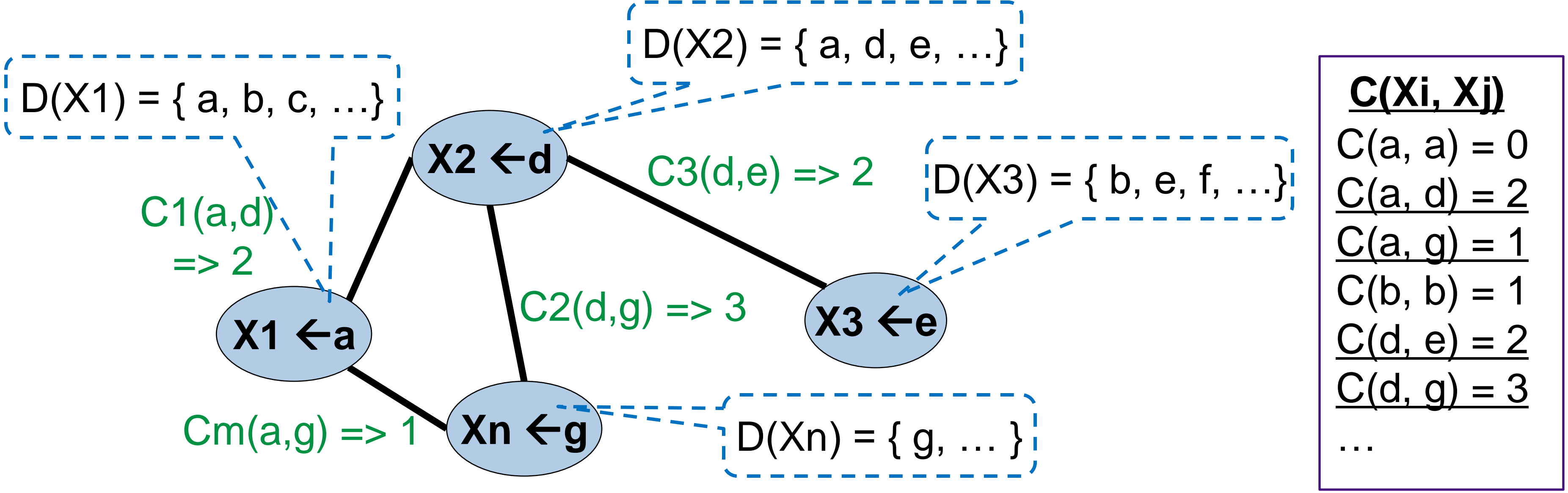
set of **valued constraints** C1, C2, …, Cm

Constraints quantify a "degree of satisfaction" (soft constraints)

each constraint maps any instantiation of its variables to a real number (cost or utility)

© 2010 Carnegie Mellon University

Solving DCOP is more than "Satisfaction"



F = C1 + C2 + … + Cm = 2 + 2 + 1 + 3 = 8

minimize F = ∑ Cij

A **solution** is an assignment of all the variables to values

Finding a solution still involves **search**, but the goal is not just to find any solution

Goal: find the **best solution** such that the sum of the constraint costs is minimized (or the sum of the utilities is maximized) → **objective function**

© 2010 Carnegie Mellon University

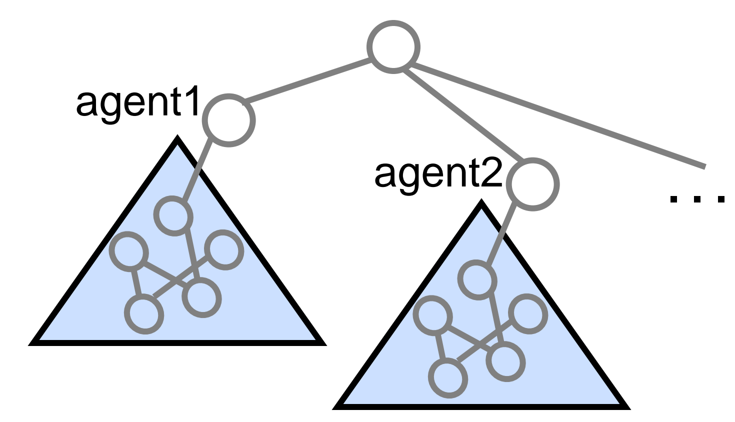
DFS Tree Ordering of the Constraint Graph

In D-COP algorithms, the agents need to be prioritized in a depth-first search (DFS) tree, such that

- Each agent has a single parent and multiple children
- Constraints are only allowed between agents that are in an ancestor-descendant relationship in the tree
- An agent has only information about ancestors it shares a constraint with

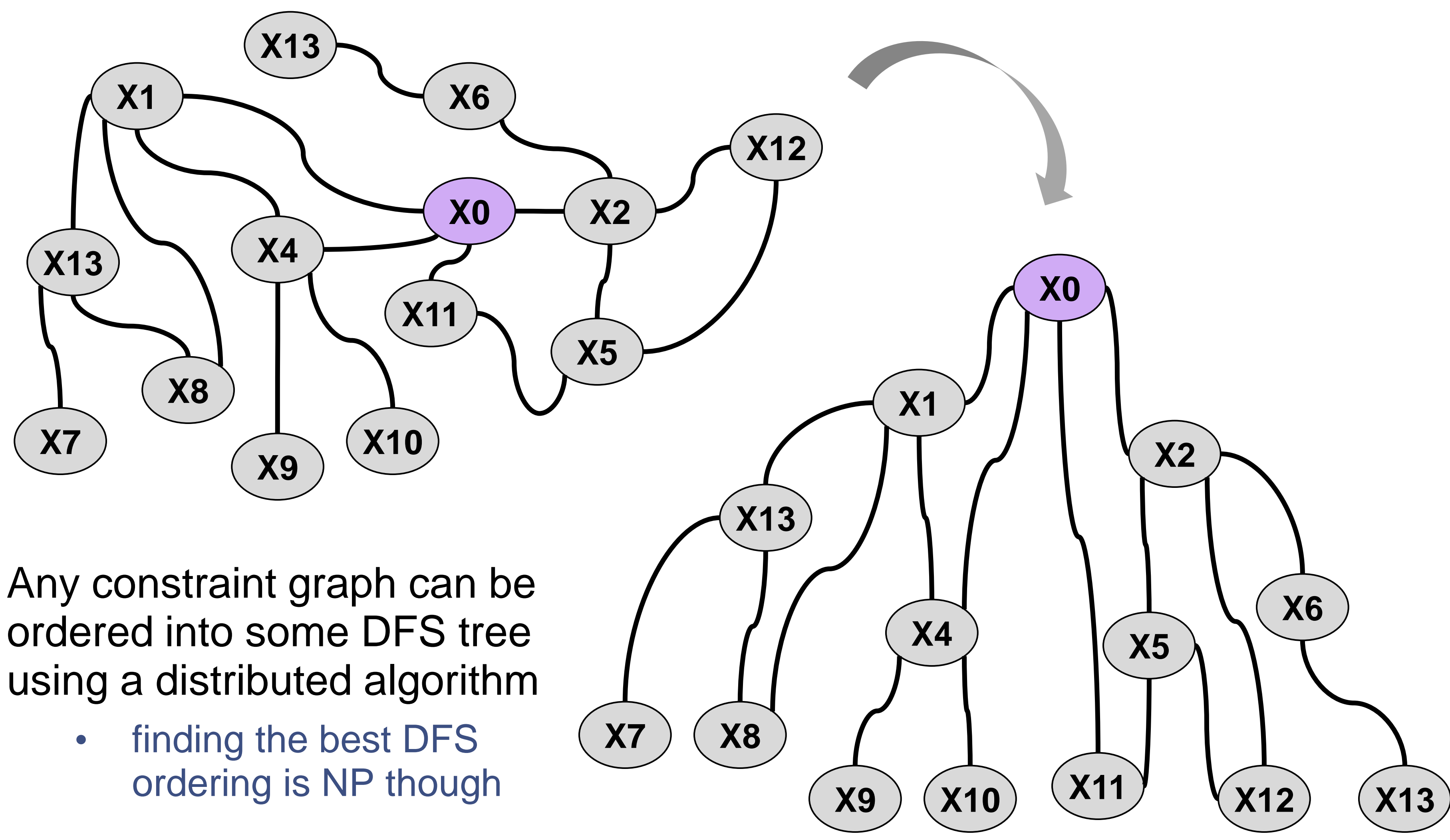
The purpose of the DFS tree is to **decompose the global cost function**

- Given the assignments to all ancestors,
- an agent in a given sub-tree can work on minimizing its part of the solution
- this work can be done independently of agents in other sub-trees



© 2010 Carnegie Mellon University

Example of DFS Ordering



Any constraint graph can be ordered into some DFS tree using a distributed algorithm

- finding the best DFS ordering is NP though

© 2010 Carnegie Mellon University

DCOP: Requirements and Issues

1. Agents need to optimize a global objective function in a distributed fashion using local communication (neighbors)
2. Allow agents to operate asynchronously (as in Dis-CSP)
3. **Quality guarantees**
   - A method to find provably optimal solutions (cost/utility) when possible

Main challenge: combining items 2) & 3)

Building consensus among agents

- Top-down: higher-rank agents decide first, lower-rank agents have to find ways to comply with higher-rank agents
- Bottom-up: agents build solutions starting with small pieces (produced by lower-rank agents) that get bigger and bigger as the reach higher-rank agents

(A tree structure that organizes the agents is assumed)

© 2010 Carnegie Mellon University

Two Search Strategies for a DFS Tree

Backtracking (top-down)

- Assumes an ordering of the variables/agents
- Control shifts between different agents during selection of values
- Requires little memory
- Exponential number of messages - with linear message size
- e.g., ADOPT [Modi05], OptAPO [Mailler04]

Dynamic programming (bottom-up)

- Assumes an ordering of the variables/agents
- Agents incrementally compute all partial solutions (i.e., all possible values), when the solutions are complete, agents pick the best solution
- Requires more memory
- Linear number of messages - but with exponential message size (vectors)
- e.g., RTree, DPOP [Petcu04] and variants

© 2010 Carnegie Mellon University

Backtracking Skeleton

1. Receive value message from ancestor agents (if not root)
2. Choose a feasible value for my variable
3. Inform value choice to children agents (if not leaf)
4. Wait for cost messages from children (if not leaf)
   - If costs are not good enough, ask children to change their values
   - If children cannot find good values, **change current value** and go to 3)
5. Combine costs and report my aggregated cost to parent
   - cost received from children (if not leaf)
   - plus cost of my constraints with the ancestor agents
6. Wait for further messages from ancestors
7. Go to 1)

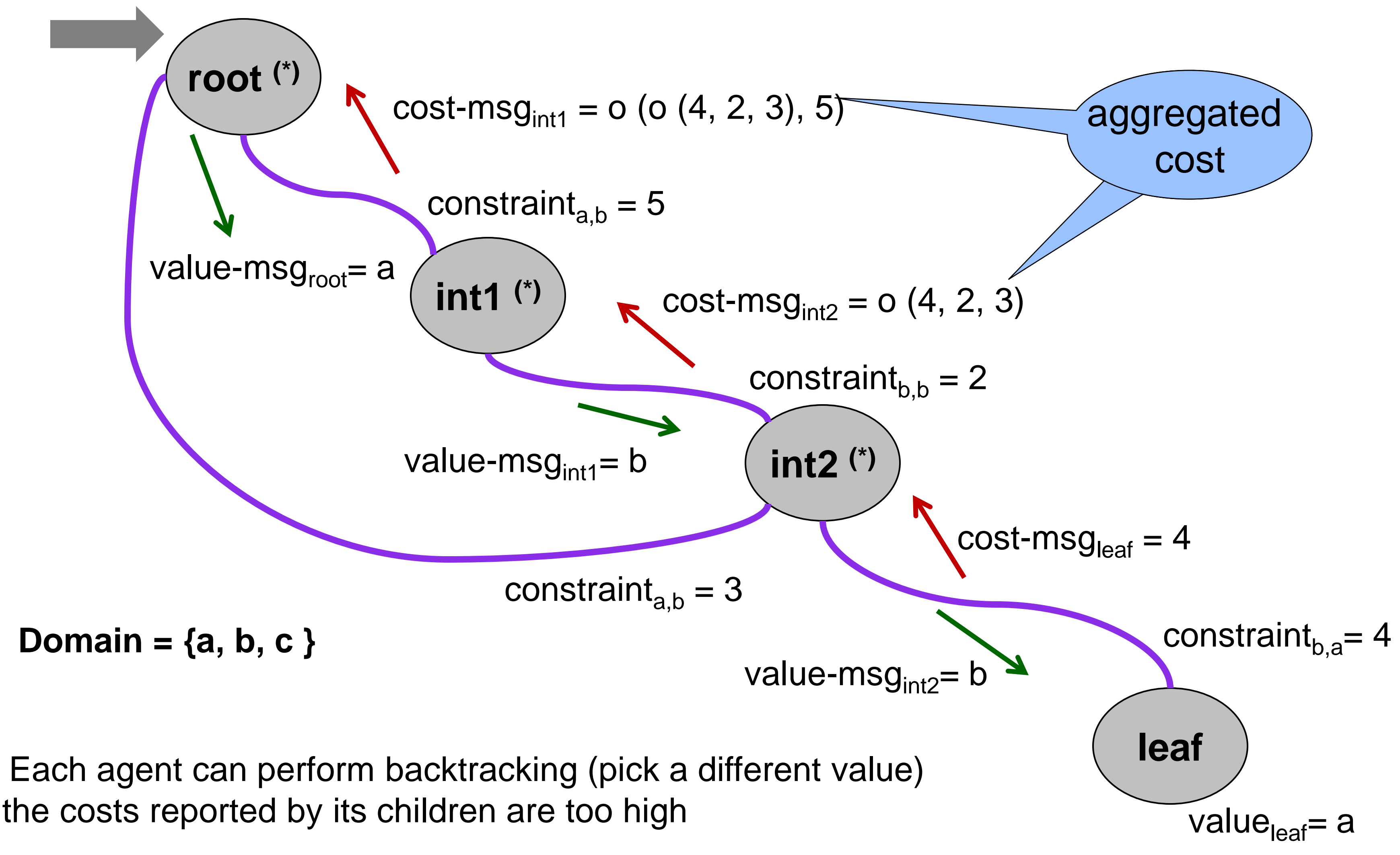
...

8. At some point, root agent decides that the optimum has been found or that there's no solution, and stops the algorithm

Agent doesn't know (a-priori) the value its children will pick

© 2010 Carnegie Mellon University

Example of Values/Costs in Top-down Search



© 2010 Carnegie Mellon University

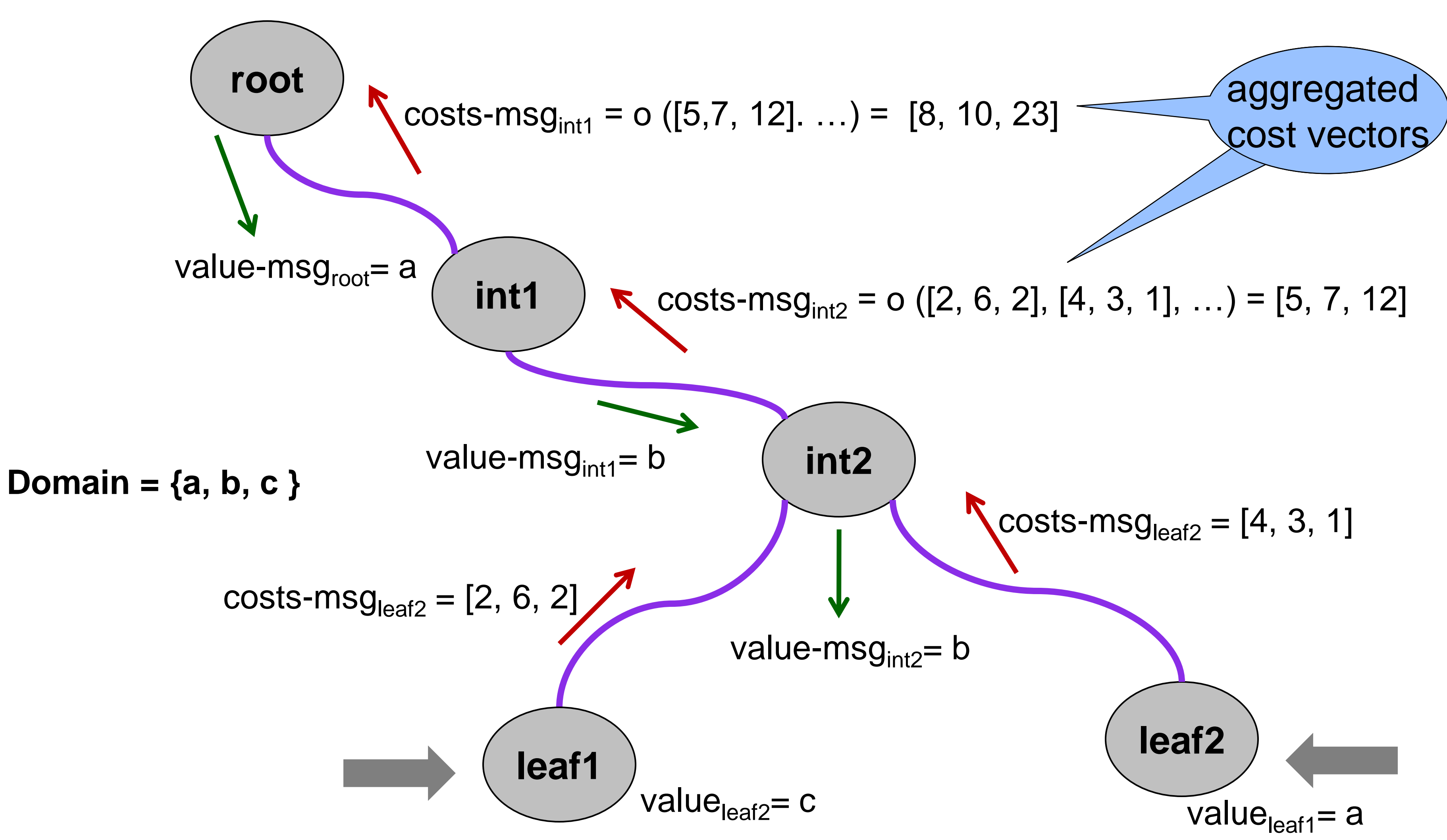
Dynamic Programming Skeleton

1. Leaf agent: Send utility vector for its possible values to parent agent
2. Intermediate agent: Receive utility vectors from children
   – Combine these utilities with my own utility table (based on constraints with my parent) → construct intermediate utility table
   – Send aggregated utility vector to parent agent
3. Root agent: Receive all the utility vectors and construct a global utility table
   – Pick my optimal value from the global utility table
   – Inform my value choice to children agents
4. Intermediate agent:
   – Pick my (local) optimal value, based on the value message from my parent and my intermediate utility table
   – Inform my value choice to children agents
5. Leaf agent: Pick my (local) optimal value and stop

Agent doesn't know the value its parent will pick

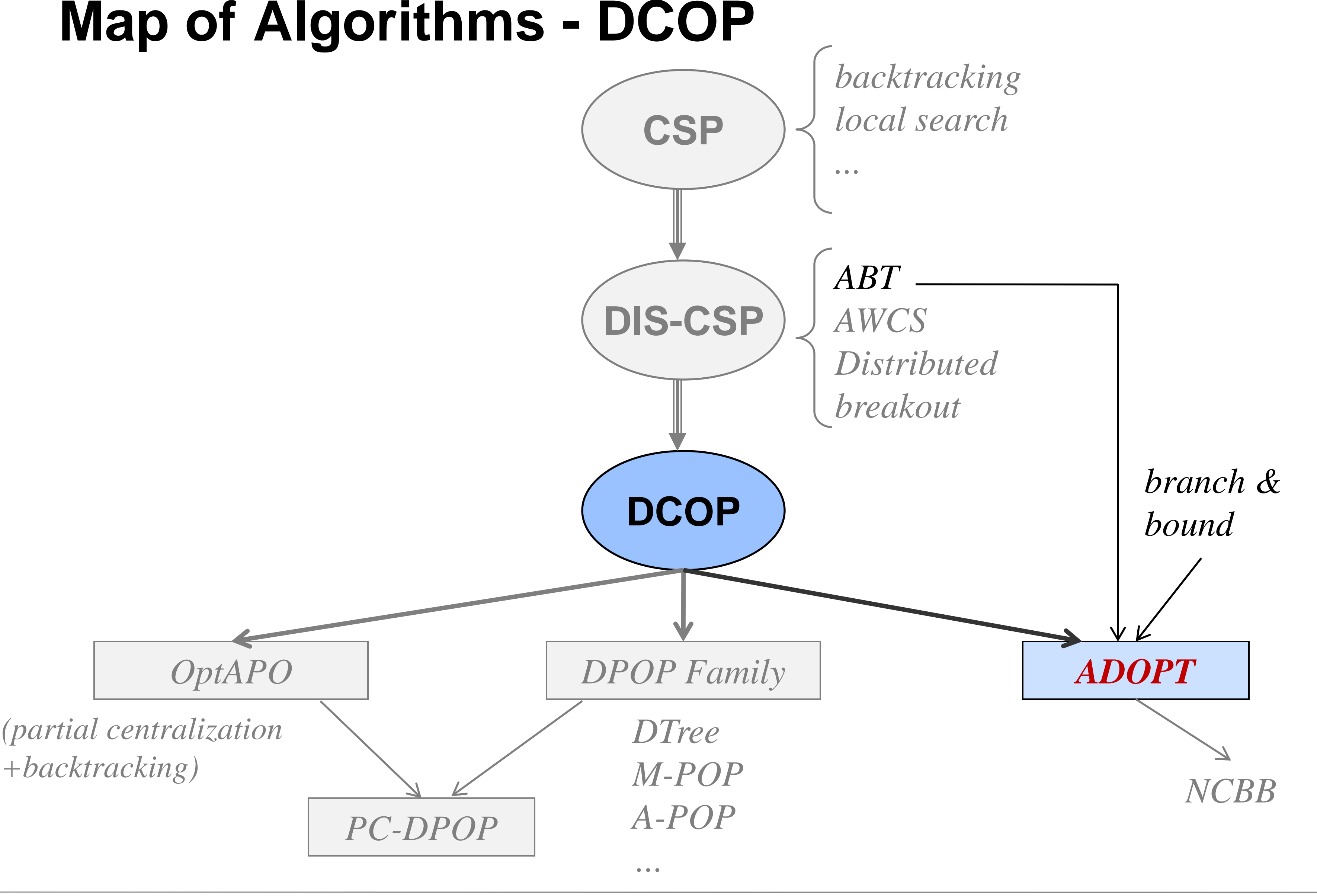
© 2010 Carnegie Mellon University

Example of Costs/Values in Bottom-up Search



© 2010 Carnegie Mellon University

Map of Algorithms - DCOP



© 2010 Carnegie Mellon University

Asynchronous Distributed Optimization (ADOPT)

Developed by Modi et al. [Modi05],

- Combination of ideas from ABT (satisfaction) and Branch & Bound (optimization)

Previous approaches backtrack only when sub-optimality is proven

- Branch and bound: It backtracks when cost exceeds upper bound
  - Limitations
    - sequential,
    - synchronous,
    - computing cost upper bounds requires global information
- Asynchronous backtracking: It backtracks when a constraint is unsatisfiable
  - Limitations
    - only "hard constraints" are allowed

© 2010 Carnegie Mellon University

Relaxing Backtracking

A root agent aggregates global costs

Weak backtracking: Opportunistic best-first search

- Agents can go ahead and make decisions based on local information
- **Cost lower bounds** of solutions are suitable for asynchronous search
  - an initial lower bound is computable based on local information
  - each agent chooses an assignment with smallest lower bound

- ADOPT backtracks when a lower bound gets too high
  - instead of when quality of best solution of sub-problem is determined

© 2010 Carnegie Mellon University

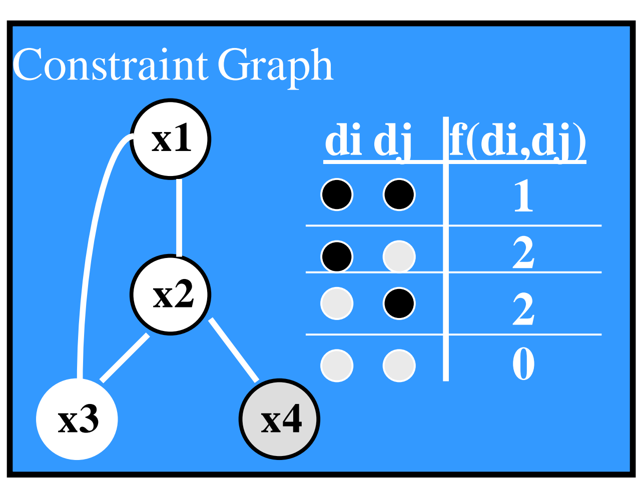
ADOPT: Problem Formulation

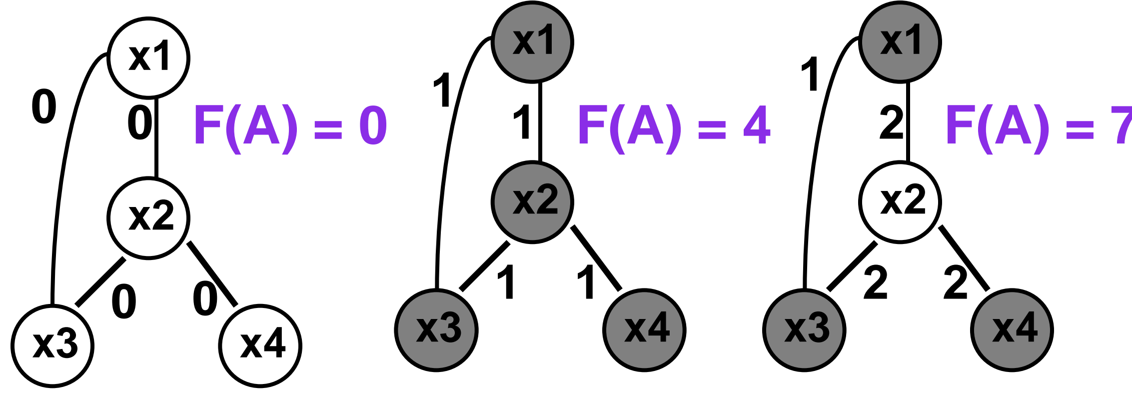
Given

- Variables {x1, x2, …, xn}, each assigned to an agent
- Finite, discrete domains D1, D2, … , Dn,
- For each xi, xj, valued constraint fij: Di x Dj → N (binary constraint)

Goal: Find complete assignment A that minimizes F(A) where

$$F(A) = \Sigma f_{ij}(d_i,d_j), \ \ x_i \leftarrow d_i, x_j \leftarrow d_j \text{ in } A$$





Adapted from slides [Modi05], Agents@USC

© 2010 Carnegie Mellon University

ADOPT: Assumptions

Aggregation operator

- The sum of the constraints is associative, commutative and monotonic
- Monotonicity requires that the cost of a solution can only increase as more costs are aggregated

Constraints are (at most) binary

- This can be extended to non-binary constraints

Each agent is assigned to a single variable

- This can be extended to several variables

Preprocessing

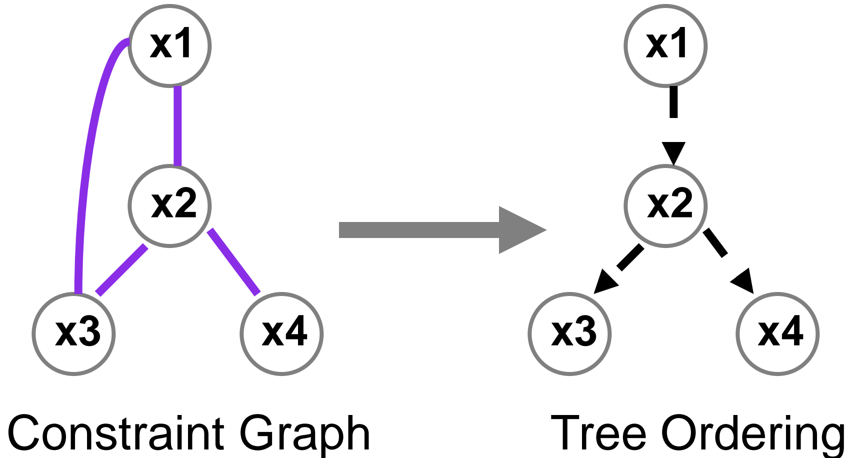
- At the beginning, all the agents must be arranged in depth-first-search (DFS) **tree structure**

© 2010 Carnegie Mellon University

ADOPT: Algorithm

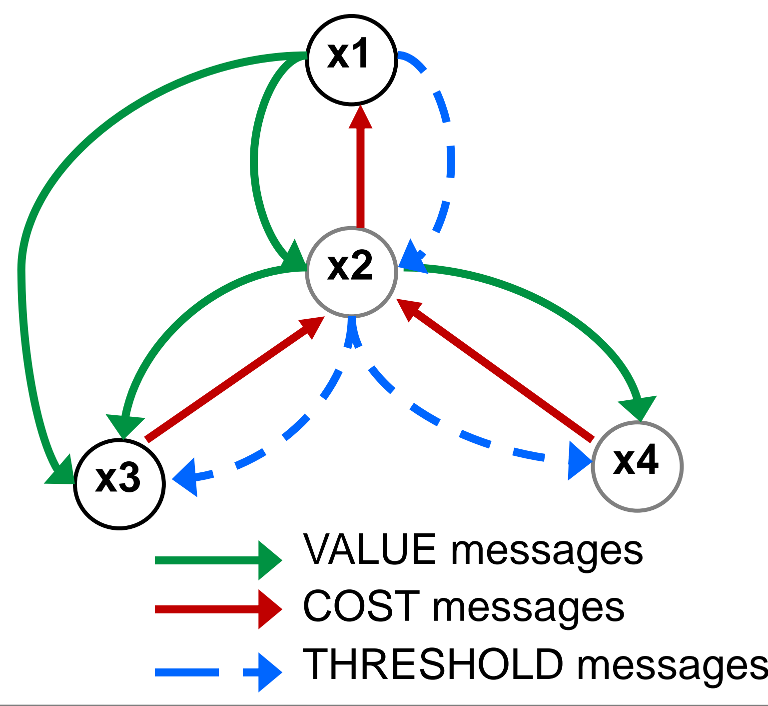
Agents are ordered in a tree

- constraints happen between ancestors/descendents
- no constraints between siblings



Basic algorithm:

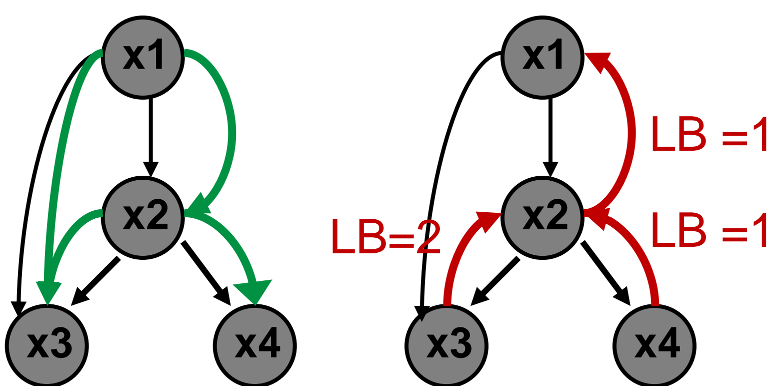
- choose value with minimum cost
- loop until termination-condition true:
  - when receive message:
    - choose value with min cost
    - send **VALUE** message to descendants (like *ok?* in ABT)
    - send **COST** message to parent (like *nogood* in ABT)
    - send **THRESHOLD** message to child



Adapted from slides [Modi05], Agents@USC

© 2010 Carnegie Mellon University

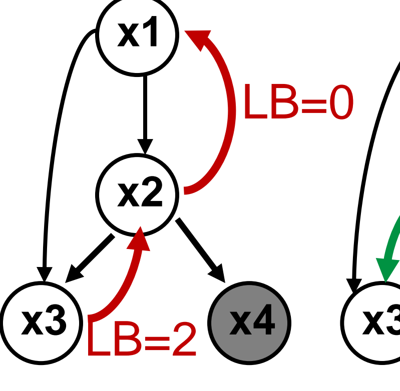
Example of ADOPT



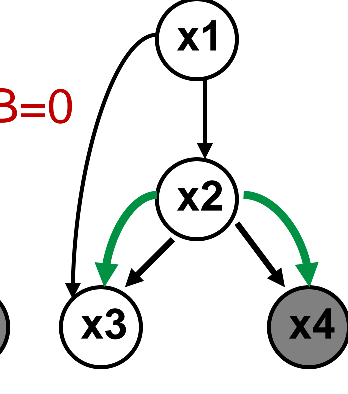
concurrently choose value, send value to descendents

report lower bounds (costs)

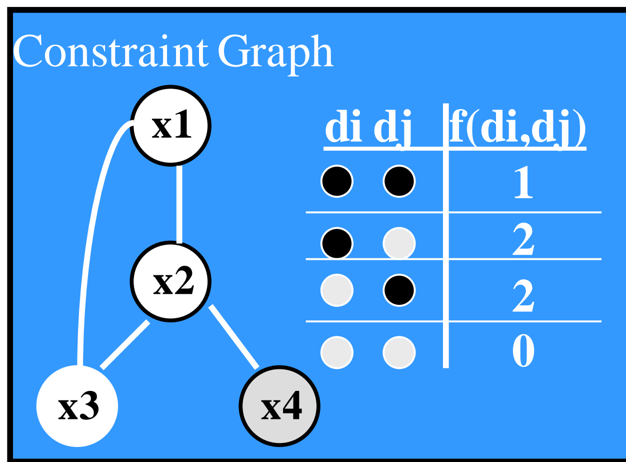
x1 switches value to "white" and propagates it



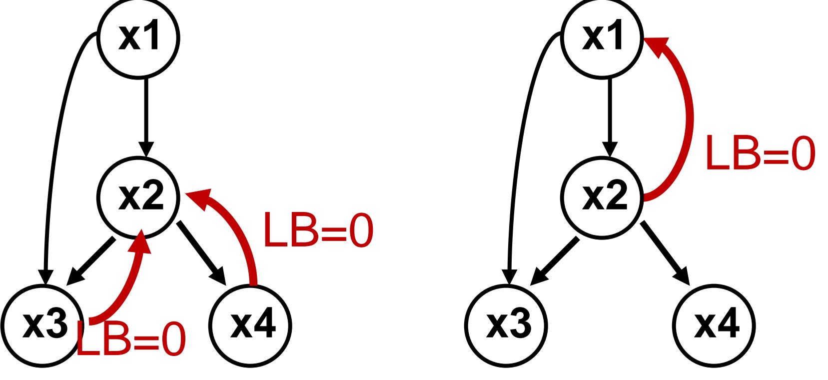
x2, x3 switch value and report new lower bounds



X2 propagates its new value



di	dj	f(di,dj)
●	●	1
●	○	2
○	●	2
○	○	0



x2, x4 report new lower bounds

optimal solution

Adapted from slides [Modi05], Agents@USC

© 2010 Carnegie Mellon University

Key Ideas in ADOPT

Opportunistic best-first search

- No global information is required, only local interactions
- Allow each agent to change a variable value whenever it detects there is a possibility that some other solution may be better than the current under investigation
  - this doesn't guarantee the new value to be better though
- Each agent picks the value with smallest lower bound

Backtrack threshold

- When an agent has to revisit a previous solution, it uses a stored lower bound to increase efficiency
- A variable/agent doesn't change its assignment as long as its total cost (i.e., lower bound) is less that the backtrack allowance

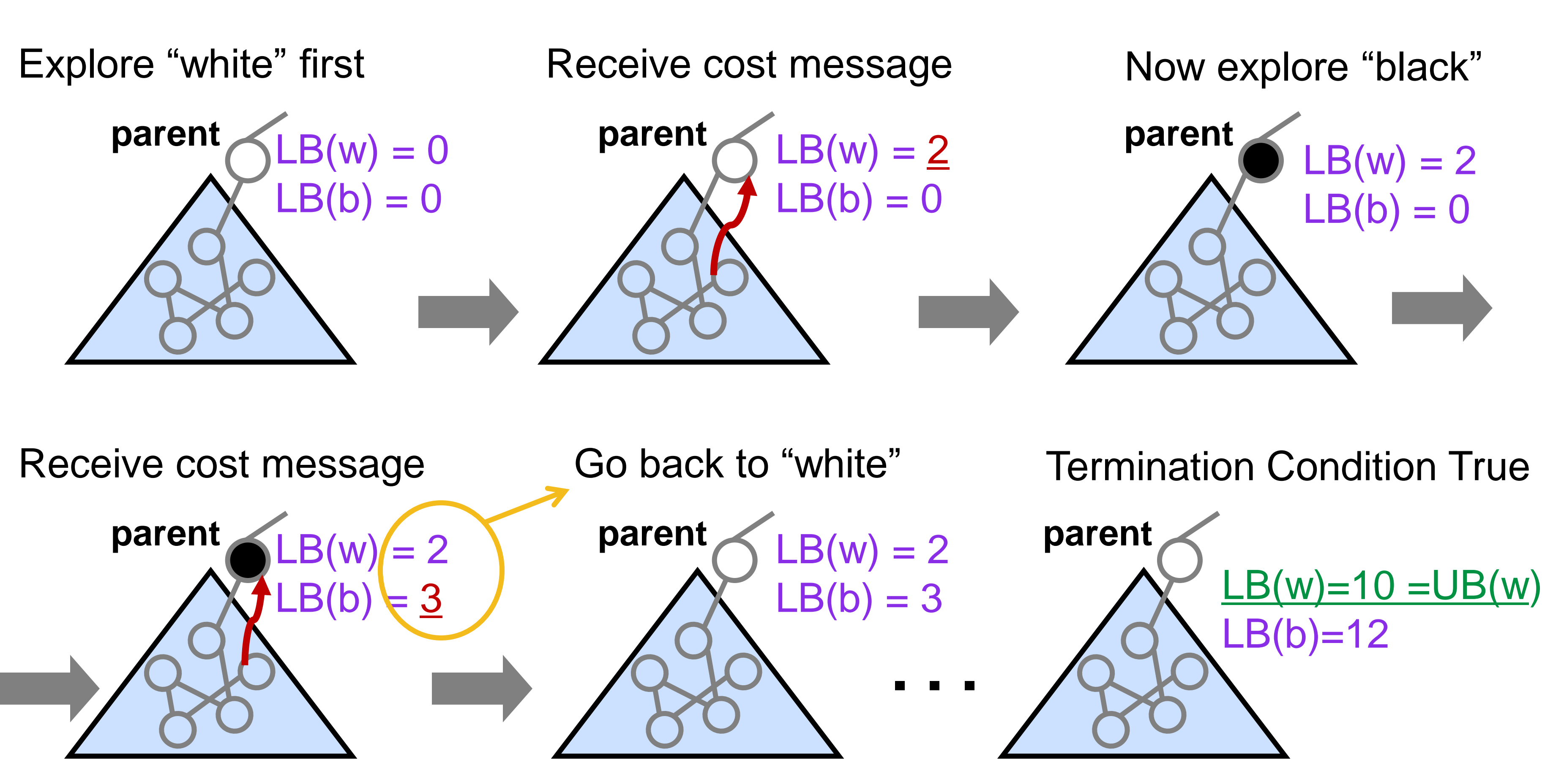
Bounded error approximation

- Sort of "quality control" for approximate solutions

© 2010 Carnegie Mellon University

Weak Backtracking

Suppose a domain with 2 possible values: "white" and "black"



Adapted from slides [Modi05], Agents@USC

© 2010 Carnegie Mellon University

Computing Lower/Upper Bounds for Costs

For a given agent/variable Xi

- Xi calculates cost as local cost plus any cost feedback received from its children
- δ(value) = sum of the costs from constraints between Xi and higher neighbors
- LB(value) = lower bound for the subtree rooted at Xi, when Xi chooses the value
- UB(value) = upper bound for the subtree rooted at Xi, when Xi chooses the value

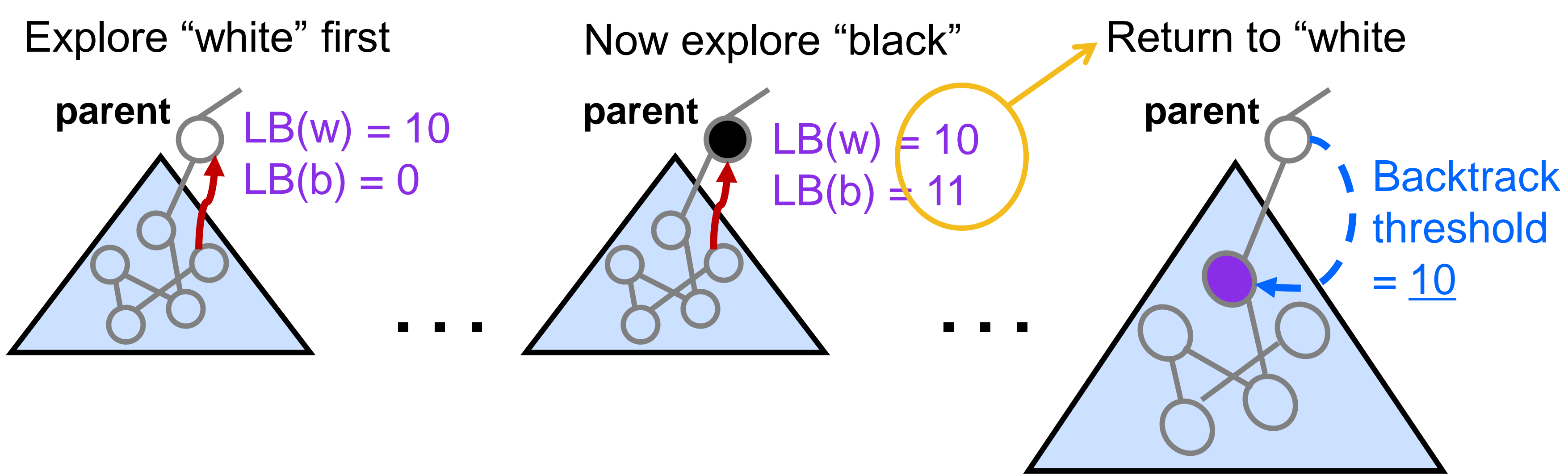
The **lower bound (LB)** for Xi is the minimum LB(value) over all value choices for Xi

The **upper bound (UB)** for Xi is the minimum UB(value) over all value choices for Xi

© 2010 Carnegie Mellon University

Backtracking Threshold – Parent

Parent informs children nodes about lower bound (for search)



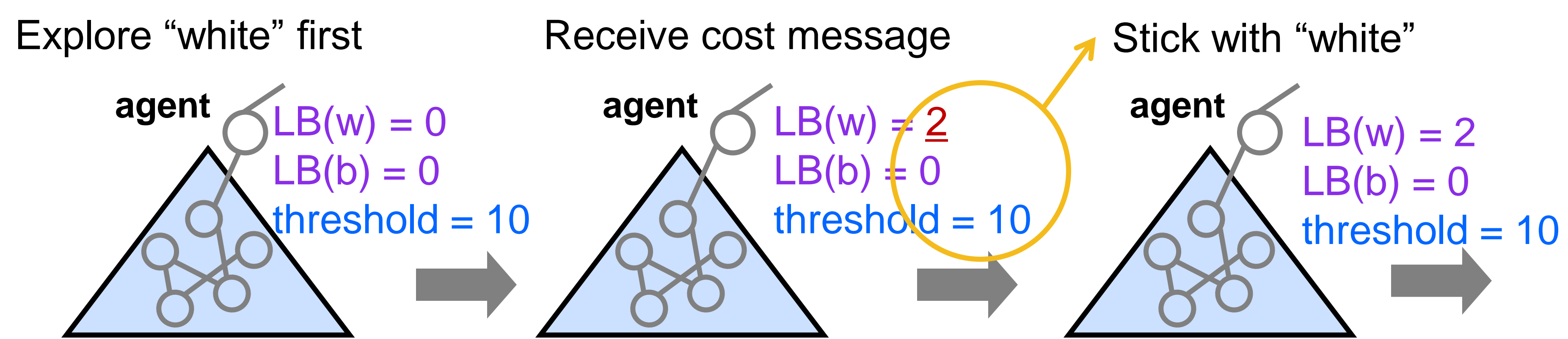
The parent knows (from previous experience) that cost >= 10
So, it informs its children not to bother searching for solutions whose cost is less than a threshold of 10

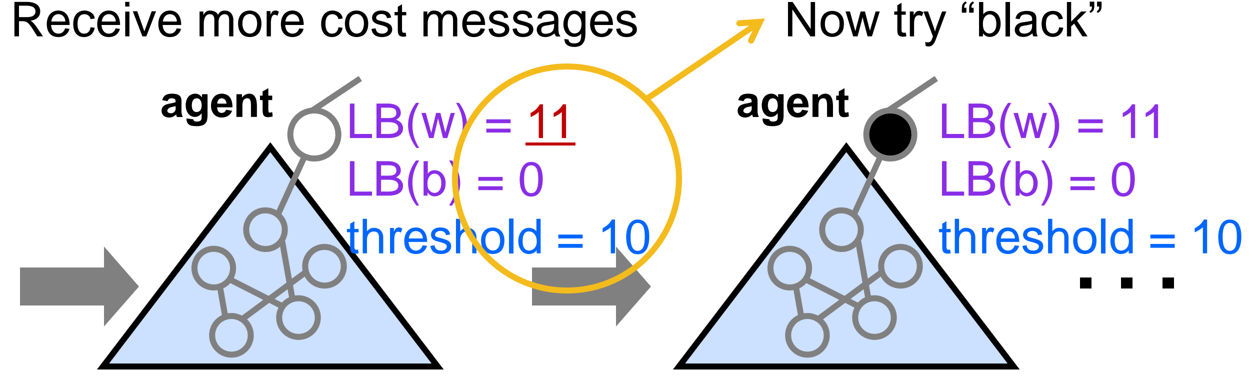
Adapted from slides [Modi05], Agents@USC

© 2010 Carnegie Mellon University

Backtracking Threshold – Child

Suppose Agent x received threshold = 10 from its parent





Key point:
Agent doesn't change value until LB(current value) > threshold

Adapted from slides [Modi05], Agents@USC

© 2010 Carnegie Mellon University

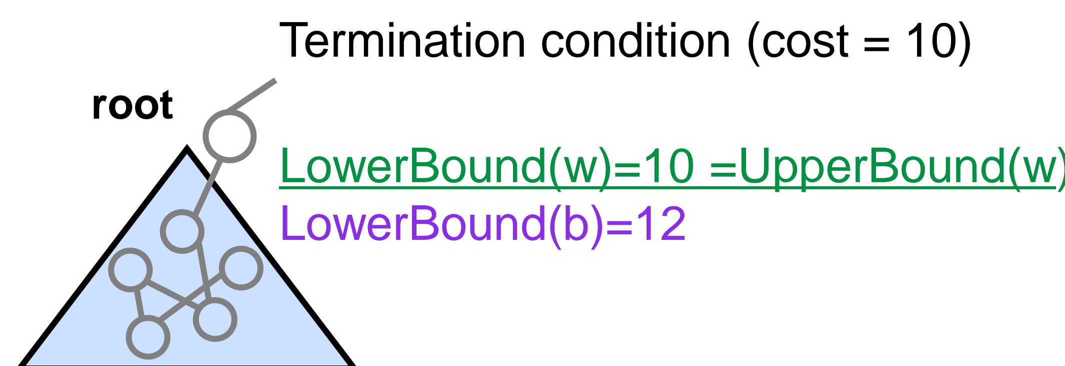
ADOPT: Termination Condition

Termination is a built-in mechanism in the algorithm

- Bound intervals for tracking progress towards the optimal solution
- When the size of the bound interval shrinks to zero, the cost of the optimal solution has been determined and agents can safely terminate
- Intervals are also used for bound-error approximation

Caveat

- Some centralization is needed: a root agent aggregates global costs and detects termination



© 2010 Carnegie Mellon University

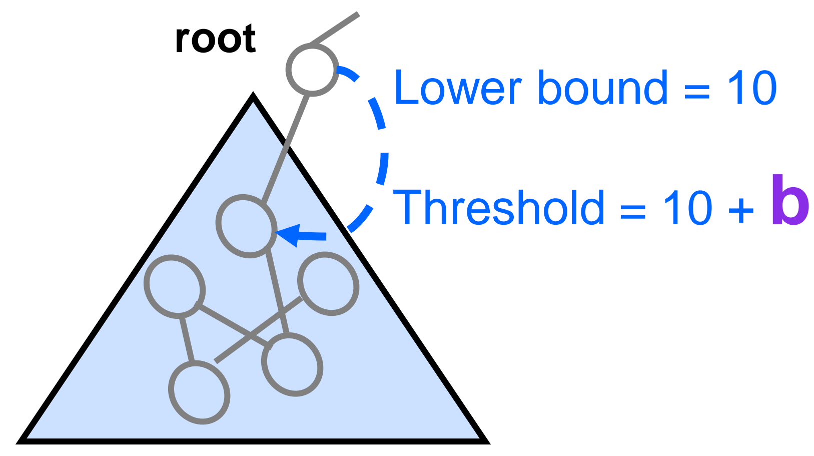
ADOPT: Bounded Error Approximation

Generate solutions whose quality is within a user-specified distance from the optimal solution

- It usually takes less time than required to find the optimum
- Based on lower bound kept during search

Example: *If an optimal solution to an over-constrained graph coloring requires violating 3 constraints, b = 5 indicates that 8 constraints is an acceptable solution for the user*

If user provides error bound **b**
Find any solution **S** where
cost(**S**) ≤ cost(optimal solution) + **b**



Adapted from slides [Modi05], Agents@USC

© 2010 Carnegie Mellon University

Observations on ADOPT

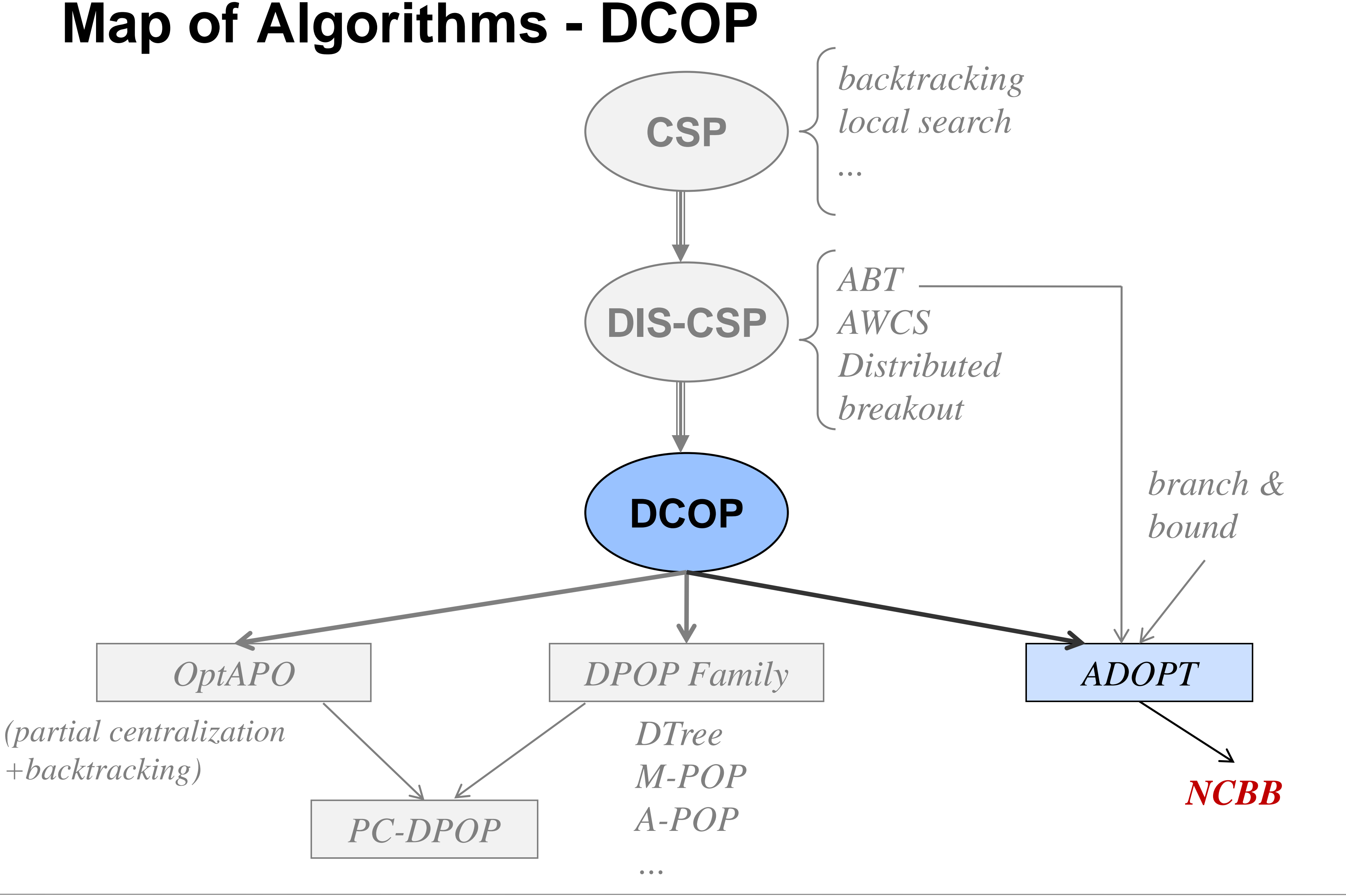
Algorithm is proven to be sound and complete

Worst-case time complexity is exponential in the number of variables

- but it only requires polynomial space at each agent
- Experiments show that sparse graphs can be solved optimally and efficiently (communication messages grow linearly in low density graphs)

© 2010 Carnegie Mellon University

Map of Algorithms - DCOP



© 2010 Carnegie Mellon University

NCBB: No Commitment Branch and Bound

Improvement over ADOPT, developed by [Chechetka06]

Speed up the search by exploiting the distributed aspect of the problem

- have different agents exploring non-intersecting regions of the search space simultaneously

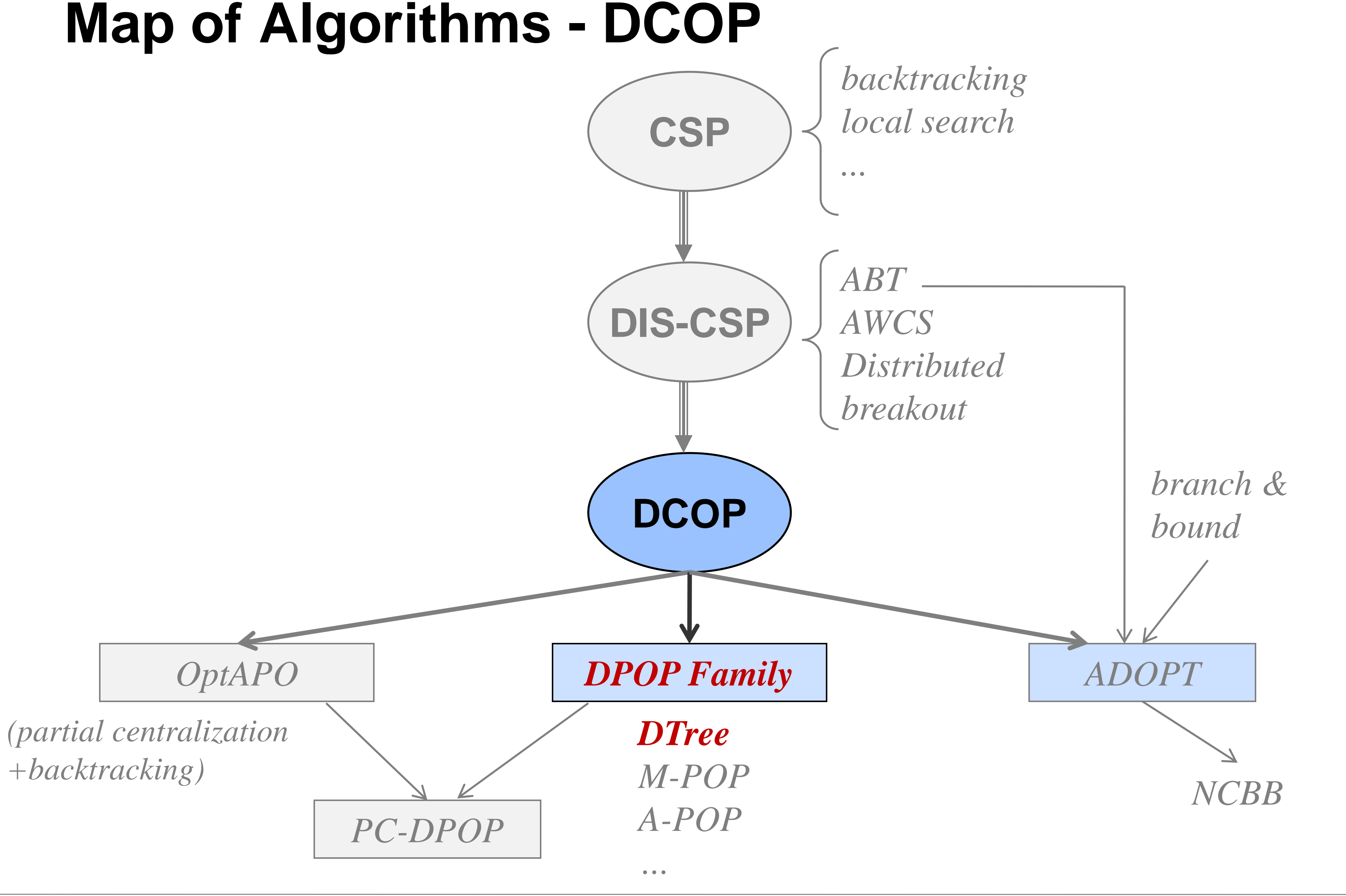
Ideas: Reduce synchronization overhead and prune the search space faster

- *A* (new) *Search* message allows each agent to search different sub-trees than its siblings
- Computation of tighter upper bounds on solution
- Eager propagation of changes in cost lower bounds

NCBB performance is better than ADOPT, and memory is still polynomial

© 2010 Carnegie Mellon University

Map of Algorithms - DCOP



© 2010 Carnegie Mellon University

Distributed Tree-shaped Networks (DTree)

DTree algorithm described in [Petcu04]

Ideas:

- Use a **dynamic-programming style** of exploration (utility vectors)
- Tree-shaped constraint networks (no cycles, yet)

Algorithm has two phases (assuming a tree structure)

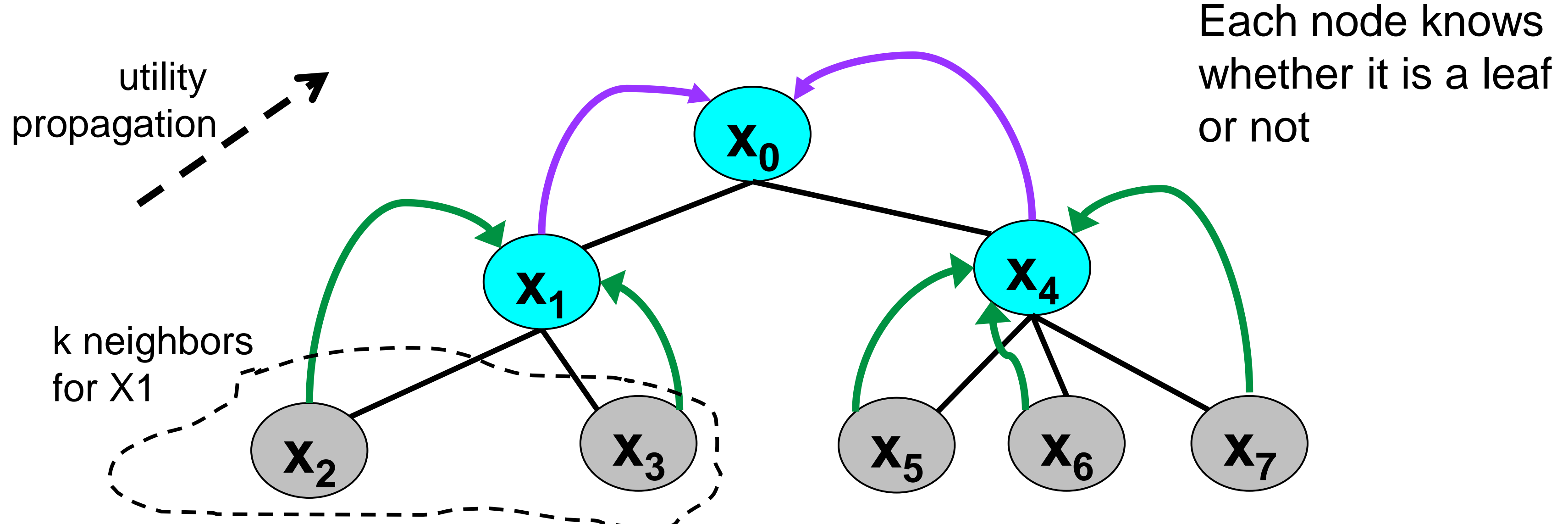
1. **UTIL propagation** from the leaves of the tree all the way up to the root
   - Each utility message **summarizes** the optimal values that can be achieved by the sub-tree rooted at the agent's node for each domain value of the parent agent
2. **VALUE propagation** from the root of the tree downwards to the leaves
   - Once all the utility messages are received from the neighboring agents, the parent agent can choose the optimal value

© 2010 Carnegie Mellon University

DTree: K-1 Propagation Rule

If node (agent) Xi has k neighbors

- Xi will send out a message to its kth neighbor only after having received the other k-1 messages
- Xi will send out the rest of the m-1 messages after having received the messages from the kth neighbor



© 2010 Carnegie Mellon University

DTree: Example of Flow of Messages

DCOP

X_0 [a,b,c]

r_1^0

X_1 [a,b,c]

r_2^1 r_3^1

X_2 [a,b,c]

X_3 [a,b,c]

Message Flow

X_2

r_2^1		X_1 a	X_1 b	X_1 c
X_2	a	0	2	1
	b	3	4	6
	c	5	2	5

$UTIL_2^1$ →

← $X_1 = c$

X_3

r_3^1		X_1 a	X_1 b	X_1 c
X_3	a	6	2	3
	b	3	3	2
	c	4	4	1

$UTIL_3^1$ →

← $X_1 = c$

X_1

X2→X1	X_1 a	X_1 b	X_1 c
	5	4	6

+

X3→X1	X_1 a	X_1 b	X_1 c
	6	4	3

+

r_1^0		X_0 a	X_0 b	X_0 c
X_1	a	2	2	3
	b	5	3	7
	c	6	3	1

$UTIL_1^0$ →

← $X_0 = a$

X_0

X_0 a	X_0 b	X_0 c
15	13	15

(a) Simple DCOP with 4 agents, 3 relations

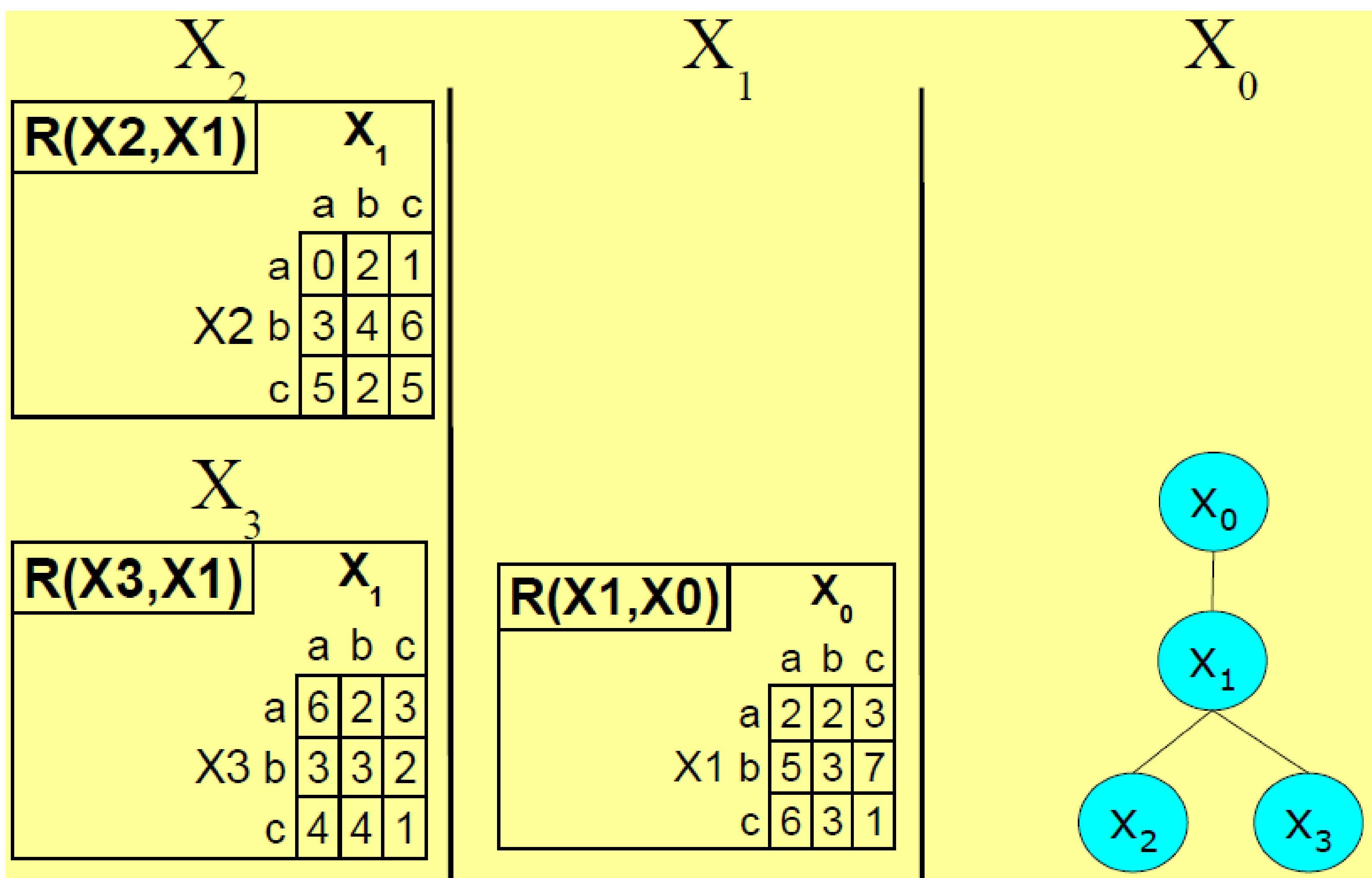
(b) Relations of X_2 and X_3 with X_1

(c) Messages to X_1 and join with relation with X_0

(d) final result

© 2010 Carnegie Mellon University

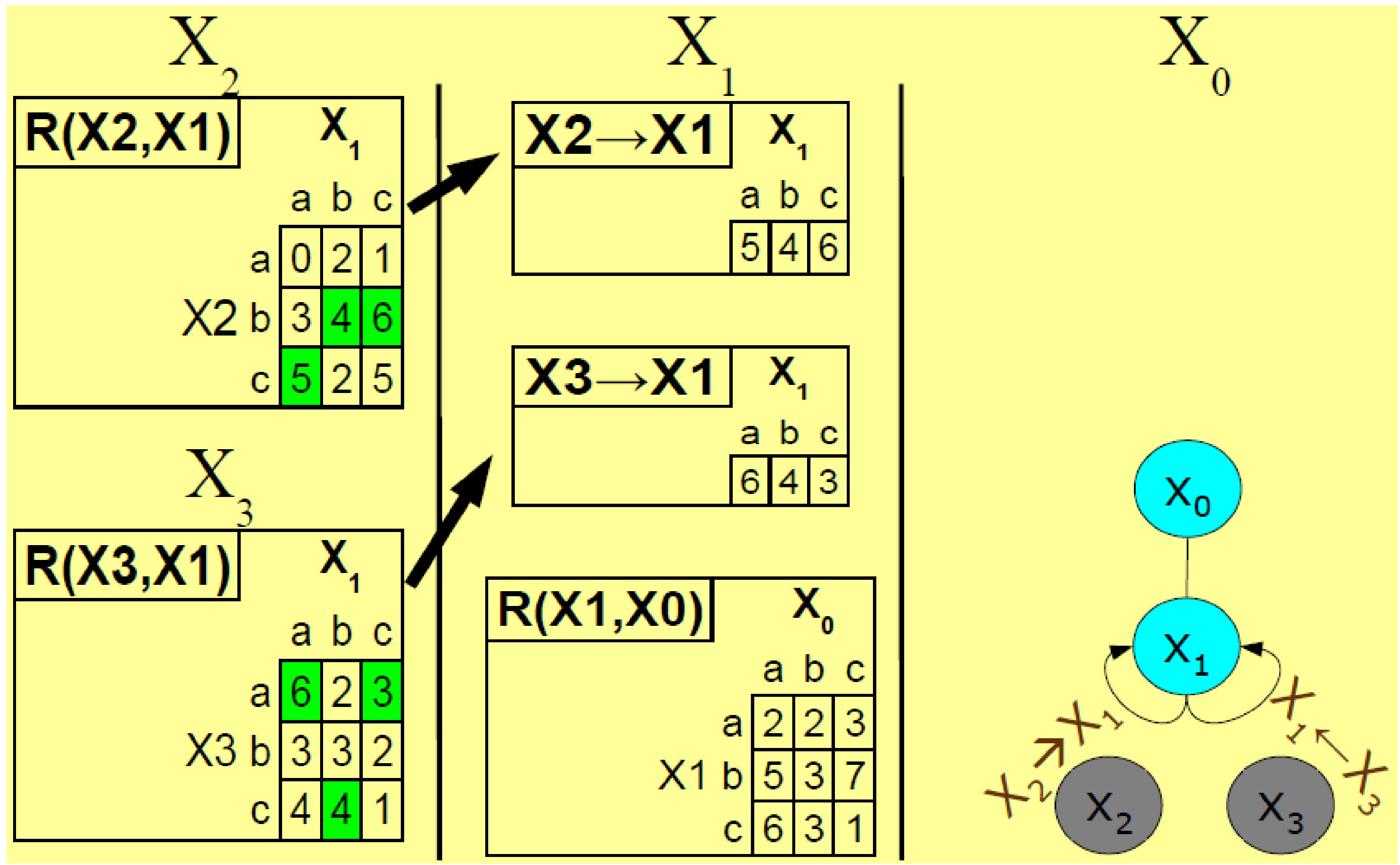
DTree: Utility Computation Example - 1



From slides [Petcu04], IJCAI'05

© 2010 Carnegie Mellon University

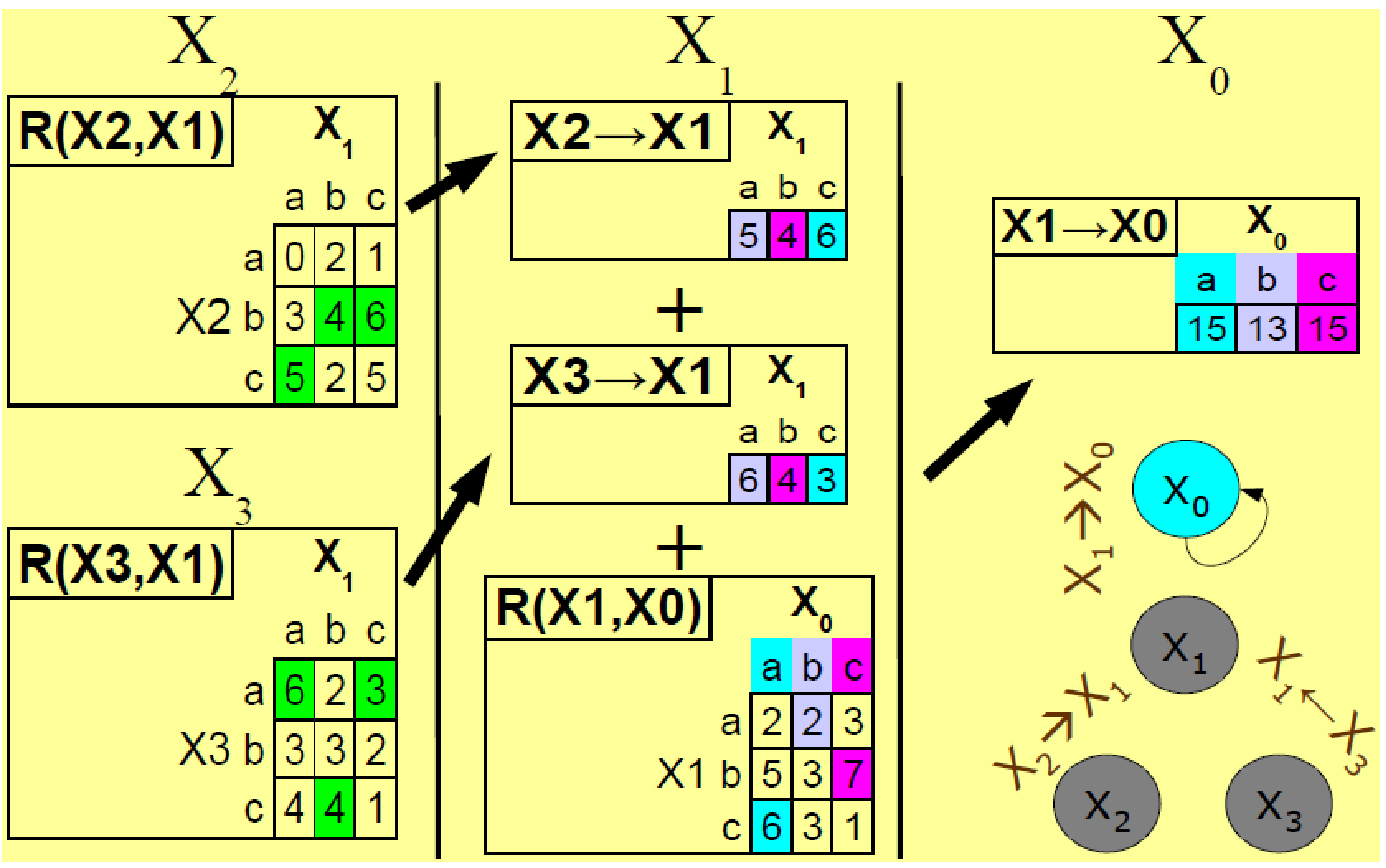
DTree: Utility Computation Example - 2



From slides [Petcu04], IJCAI'05

© 2010 Carnegie Mellon University

DTree: Utility Computation Example - 3



From slides [Petcu04], IJCAI'05

© 2010 Carnegie Mellon University

Distributed Pseudo-tree Optimization (DPOP)

Application of DTree phases to arbitrary constraint topologies

- Constraint graphs can have cycles (agents other than parent/children)
- Dynamic programming technique is still useful with some modifications

Ideas: Transform graph into a pseudo-tree by traversing it in DFS order

- **Phase 0: Topology probing** to fix the pseudo-tree arrangement
- Break the problem into cycle-free parts → **choice of cycle cutset**
- Utility messages are now "multi-dimensional"

Message size is still exponential, but reduced from dom^n to dom^k

- where n = #nodes and k = #induced width of the DFS (tree) arrangement
- the more a problem has a tree-like structure, the lower its induced width
- n >> k for large and loosely coupled problems

© 2010 Carnegie Mellon University

DPOP Variants – Efficiency

Approximations on solution quality

- Tradeoff between solution quality and computational effort
- The user can specify a maximal error bound on the solution
- A-DPOP: Approximation of the solution by adapting the message size

Local search

- Start with some (random) solution and gradually improve it
- LS-DPOP: Nodes make decisions based only on local information
- Applicable in large neighborhoods

Partial centralization

- Use mediation (like in OptAPO) for tightly connected clusters
- Each mediator can take advantage of an efficient CSP solver
- PC-DPOP: Identify difficult sub-problems and centralize them in relevant nodes of the problem

Time-space tradeoff

- MB-DPOP: memory bound extension

© 2010 Carnegie Mellon University

DPOP Variants – Dynamic Problems

Self-stabilization property: Given enough time between changes/failures, the algorithm will converge to the optimal D-COP solution, and then will maintain that state

- Changes in topology
- Changes in the valuations of the agents
- Temporary communication problems

DPOP extensions: Apply self-stabilization protocols to DPOP phases

- SS-DPOP: based on self-stabilizing algorithm introduced by Kollin et al.
- RS-DPOP: continuous-time problem by reacting to problem changes and generating new solutions with low costs
  - DynDCOP: DCOP formulation extended with "stability constraints" and "commitment deadlines"

© 2010 Carnegie Mellon University

Thoughts: Hybrid Algorithms

In practice, D-COP algorithms have to make different tradeoffs among

- Message size, number of messages, memory usage
- Degree of distribution (mediation vs. full decentralization)
- Solution quality (complete vs. incomplete algorithms)
- Adaptation to changes in (dynamic) environments
- Agents that might be not cooperative
- Privacy concerns when the agents reveal information

How to deal with non-binary constraints and local variables efficiently?

How to model “social choice” or “agent preferences” using D-COP?

Incomplete algorithms can be a scalable option for agents that form small groups and optimize within these groups

© 2010 Carnegie Mellon University

References - 1

[Chechetka06] "No-commitment Branch and Bound Search for Distributed Constraint Optimization". A. Chechetka and K. Sycara. In Proceedings of the Fifth international Joint Conference on Autonomous Agents and Multiagent Systems . AAMAS '06, Japan, 2006. ACM, New York, NY, 1427-1429.

[Mailler04] "A Mediation-Based Approach to Cooperative, Distributed Problem Solving". R. Mailler. Doctoral Thesis. University of Massachusetts Amherst. 2004

[Modi05] "Adopt: Asynchronous Distributed Constraint Optimization with Quality Guarantees" – P. Modi, W. Shen, M. Tambe, M. Yokoo Artificial Intelligence, Vol. 161, Issues 1-2,, January 2005, Pages 149-180.

[Petcu04] "A Distributed, Complete Method for Multiagent Constraint Optimization" A. Petcu, B. Faltings - CP Fifth Int. Workshop on Distributed Constraint Reasoning (DCR2004), Toronto, Canada, September, 2004.

[Petcu06] "Recent Advances in Dynamic, Distributed Constraint Optimization", A. Petcu. Technical Report No. 2006/006, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), May, 2006.

© 2010 Carnegie Mellon University

References - 2

[Russell09] "Artificial Intelligence: Modern Approach", Chapter 5 - S. Russell and P. Norvig. Prentice-Hall, 3rd Edition. 2009

[Yokoo98] "The Distributed Constraint Satisfaction Problem: Formalization and Algorithm" – M. Yokoo, H. Durfee, T. Ishida and K. Kuwabara. IEEE Transactions on Knowledge and Data Engineering, Vol. 10, No. 5, September 1998.

© 2010 Carnegie Mellon University

Some Implementations

Choco (CSP in Java)

ADOPT (Java)

Frodo (DPOP platform in Java)

© 2010 Carnegie Mellon University

Software Engineering Institute | Carnegie Mellon

© 2010 Carnegie Mellon University