



---

# ***Distributed Software Development More Problem Solving***

Chris Brooks

Department of Computer Science  
University of San Francisco

# ***Distributed Problem Solving***

- Last week, we talked about *loosely coupled* distributed problems
  - Primarily distributed search
- A large data set is divided across clients.
- Clients interact only with a central server; no client-client communication

## ***“Medium-coupled” problems***

- In medium-coupled problems, each node can do a substantial amount of computing on its own
- A final solution will require communication between nodes.
- Actions taken by one node can affect other nodes.

## ***Example: scheduling classes***

- Before the semester starts, Benson sends every teacher an email: When and where do you want your classes?
- Each professor has their own constraints, and can come up with choices locally:
  - Brooks: no classes before 10 am, in Kudlick
  - Wolber: No classes on Friday
  - Galles: Mornings, nothing on Lone Mountain.
- Brooks is able to eliminate some possible schedules without consulting with anyone else.

## *Example: scheduling classes*

- Some choices require communication with a center (Benson)
  - Only one class in Kudlick at a time.
  - Some classes may be constrained by other departments' choices
    - CS 110 and Calculus I should be at different times.
- Nodes start with a large set of constraints; some are eliminated by the center.
- Hopefully at least one viable schedule remains.
  - If not, someone is “encouraged” to relax their constraints.

# ***Constraint Satisfaction***

- A constraint satisfaction problem is one of assigning values to variables so as to satisfy a set of constraints.
- Typically, any solution that satisfies the constraints is equally acceptable.
- Toy problems:
  - N-queens
  - map coloring
- Real problems:
  - Scheduling CS classes
  - Building a car
  - Register allocation

# Constraint Satisfaction

- More formally, a CSP consists of:
  - a set of variables  $\{x_1, x_2, \dots, x_n\}$
  - Each variable as a domain of possible values  $D_1, D_2, \dots, D_n$
  - and a set of constraints  $C_1, C_2, \dots$ 
    - Unary constraints:  $x < 10, y \bmod 2 == 0$ , etc
    - Binary constraints:  $x < y, x + y < 50$ , etc
    - N-ary constraints:  $x_1 + x_2 + \dots + x_n = 75$
    - (a problem with N-ary constraints can be transformed into a binary constraint problem, with an increase in the number of variables)

## ***Formalizing a CSP***

- An assignment of values to variables that satisfies all constraints is called a *consistent* solution.
- We might also have an objective function  $y = f(x_1, \dots, x_n)$  that lets us compare solutions.

# Approaches

- If the domain of all variables is continuous (i.e. real numbers) and constraints are all linear functions, we can use *linear programming* to solve the problem.
- This is actually the easy version (in P).
  - Express the problem as a system of equations
- If variables are discrete, the problem is harder.
- We can use *dynamic programming*.
- Often, variables have complex or symbolic values.
- In the most general case, we can express a CSP as a search problem.

# *Solving CSPs with search*

- We can use depth-first search to solve a CSP
  - Begin with an initial state: no values assigned to  $x_1, \dots, x_n$
  - Sequentially assign values to variables.
  - We are done if we find an assignment to each variable such that all constraints are met.
- Since CSPs are commutative (for a solution, it doesn't matter which order values are assigned) we can consider one variable at a time.

# *Backtracking*

- With CSPs, the challenge is deciding how to proceed when a constraint is violated.
  - Some assignment of values to variables must be undone, but which?
  - This decision is called *backtracking*
  - Standard DFS undoes the most recently assigned value.
  - This is called *chronological backtracking*
  - Easy to implement
  - Problem: an early assignment may have doomed us to an inconsistent solution.

# Example



- Three-coloring the map of Australia
- Assigning  $Q = \text{red}$ ,  $\text{NSW} = \text{green}$ ,  $V = \text{blue}$ ,  $T = \text{red}$  cannot lead to a solution.
- Different values for  $T$  will not change this.
- $V$  needs a different value.

# *Distributed CSP*

- In some cases, the variables in a CSP may be distributed over multiple nodes or processes or agents.
  - Natural division of problem (e.g. scheduling a meeting, coordinating research teams)
  - Information may be private, or difficult to quantify and share.
- Constraints may be intra-agent or inter-agent.
- Agents communicate by message passing with asynchronous communication.

# *Algorithms for Distributed CSP*

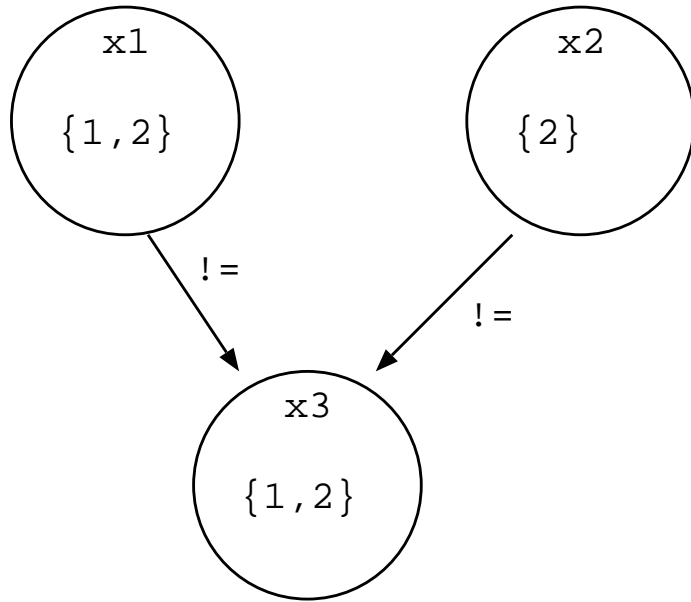
- For purposes of demonstration, we'll make the following assumptions:
  - Each process has a single variable
  - All variables have binary values
  - All constraints are binary.
- Note: this is just to keep the examples simple; the algorithms work fine without these assumptions.

# *Asynchronous Backtracking*

- *Asynchronous backtracking* extends normal backtracking search to a distributed environment.
- Each process has a priority  $p_i$ .
- Each agent chooses a value for its variable and send this value to all processes that it shares a constraint with.
  - No defined order in which messages are sent (asynchronous communication)
  - All agents are selecting values simultaneously.
- If a node cannot find a consistent value, it generates a nogood.

# Asynchronous Backtracking

## Example



- $x_1$  and  $x_3$  can be either 1 or 2.
- $x_2$  can only be 2.
- $x_1 \neq x_3$ ,  $x_2 \neq x_3$
- (we can see that the only solution is  $x_1 = x_2 = 2$ ,  $x_3 = 1$ )

# Asynchronous Backtracking

## Example

- Each node sends messages to other nodes that it shares a constraint with.
  - Includes all known assignments
- For example,  $x_1$  sends an  $(ok?, (x_1, 1))$  to  $x_3$ .
- $x_2$  sends  $(ok?, (x_2, 2))$  to  $x_3$ .
- $x_3$  constructs a *local view* from this:  $((x_1, 1), (x_2, 2))$
- $x_3$  cannot choose a value consistent with this local view.

# Asynchronous Backtracking

## Example

- x3 sends a nogood to the lowest-priority process in its local view. (x2).
  - $(\text{nogood}(x1, 1), (x2, 2))$
  - Because of asynchronous communication, x3 must include the entire local view.
  - x2 may have already changed its value.
- Since x2 does not have a link to x1, it adds x1 as a neighbor.
- Requests x1, current value.

# Asynchronous Backtracking

## Example

- $x_1$  returns  $(x_1, 1)$ , which  $x_2$  adds to its local view.
- $x_2$  checks its current assignment and its local view against the list of nogoods.
  - $((x_1, 1), (x_2, 2))$  is a nogood.
- Therefore,  $x_2$ , can't find a consistent value, and so it sends  $(nogood, (x_1, 1))$  back to  $x_1$ .
- $x_1$  receives the nogood. Since there are no other processes included,  $x_1$  knows that it must change its value.
- It chooses  $(x_1, 2)$  and resends  $(ok?, (x_1, 1))$  to  $x_2$  and  $x_3$ .
- $x_2$  can safely also choose 2. When  $x_3$  gets the message, it sends  $(ok?, (x_1, 2), (x_3, 2))$  to  $x_2$ .

# *Asynchronous backtracking*

- Guaranteed to terminate and find a solution if one exists (complete).
- Same process as single-processor backtracking
- If an variable can't be assigned a value, undo the next-most-recent variable.
- More communication, due to asynchronicity.

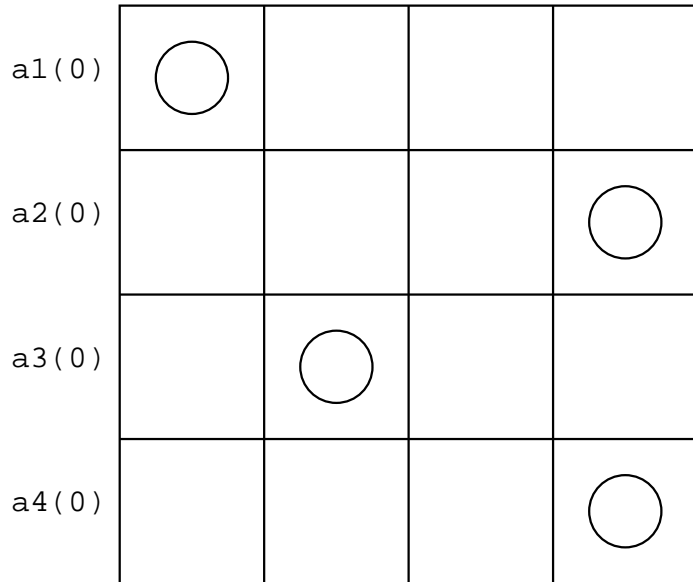
- A problem with asynchronous backtracking is the statically defined priorities.
  - In the example, a poor choice by x1 influenced everything.
  - A bad choice by a high-priority agent might cause a great deal of needless resetting of variables.
- Rather than just sending a nogood to the lowest-priority process, we use a heuristic.

# Asynchronous Weak-commitment

## Search

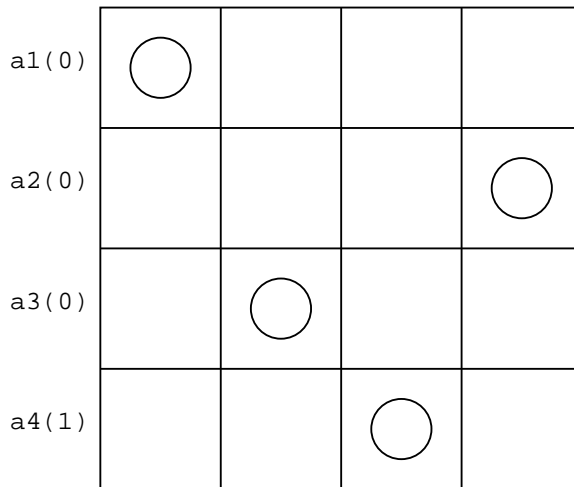
- Each process gets a priority value assigned dynamically.
- When an agent can't find a value that satisfies all constraints, it:
  - chooses a value that minimizes constraint violations
  - increases its priority to  $k + 1$ , where  $k$  is the highest nogood received.
  - Sends nogoods to all lower-priority processes.

# Example



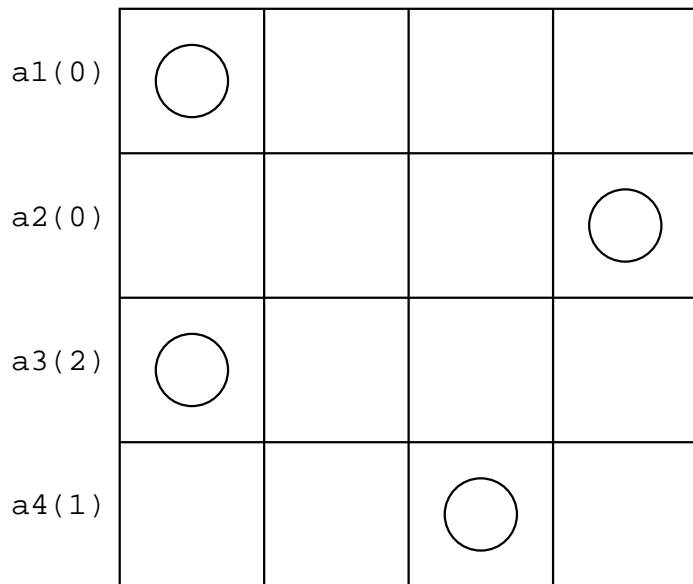
- One agent per row
- Initially all priorities are 0.
- All agents send ok? messages to each other.
- $a4$  is unable to find an assignment consistent with its local view.

# Example



- a4 sends  
 $(nogood(a1, 1), (a2, 4), (a3, 2), (a4, 4))$   
messages to all other agents.
- a4 increments its priority.
- Chooses a new value that  
minimizes constraints, given  
its local view.
  - $a4 = 3$
  - a4 sends ok? messages  
to all other processes.

# Example



- a3 receives a4's ok? and tries to find a new value.
- No consistent value
  - Selects 1 to minimize conflicts
  - Increments priority to 2 (since a4 was 1).
  - Sends ok? to all lower priorities.
  - a2 and a4 are fine with this choice.

# Example

|       |   |   |   |   |
|-------|---|---|---|---|
| a1(0) |   | ○ |   |   |
| a2(0) |   |   |   | ○ |
| a3(2) | ○ |   |   |   |
| a4(1) |   |   | ○ |   |

- a1 has a conflict with a3, and moves to 2.
- Consistent, so no need to send nogoods.
- The solution satisfies all constraints.
- Notice that the original problem stemmed from a bad choice by a1.
  - This algorithm doesn't exhaustively search a2's choices before changing a1.

- Backtracking and weak-commitment are examples of algorithms that construct consistent partial solutions.
  - Each node tries to select values that are consistent with some other subset of nodes.
- We can also use hill-climbing to solve this problem
  - Sometimes called iterative improvement.
- Premise start with a complete but flawed solution and try to apply improvements.

- We can use the min-conflict heuristic to guide hill-climbing.
- Start with a random assignment of values to variables.
- foreach variable:
  - Choose the value that minimizes the number of conflicts
- Repeat until:
  - A solution is found
  - A local optimum is reached.

# ***Breakout***

- *Breakout* helps us jump out of local optima.
- Each constraint starts with a weight of 1.
- When a local optimum is found, all currently-violated constraints have their weights increased by 1.
  - This makes nearby states more appealing, and “pushes” the hillclimber out of the optimum.
- Like all hill-climbing algorithms, this is not complete, but can be tuned to work well in practice.

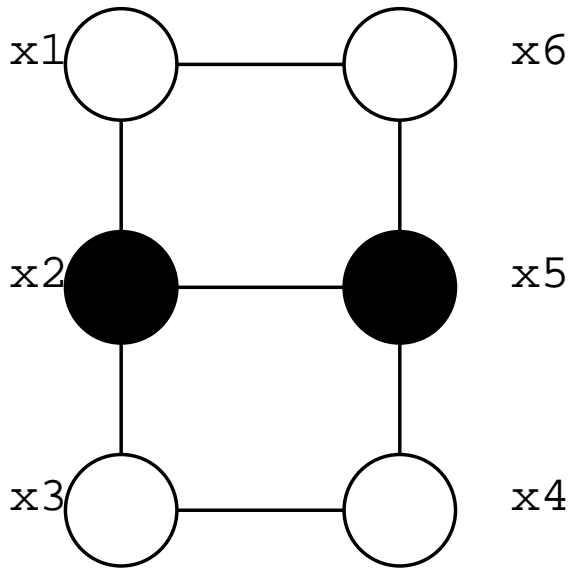
# *Distributed Breakout*

- We can also construct a distributed version of breakout.
- Neighbors exchange possible assignments, and the agent whose assignment produces maximal improvement is chosen.
- Agents detect that subgroups are trapped in a quasi-local minimum. (no need to solve consensus to detect a local minimum).
  - $x_i$  is in a quasi-local minimum if it is violating a constraint, and the improvement of all of its neighbors is 0.
  - In other words, no hill can be climbed in the agent's immediate vicinity.

## ***Distributed Breakout***

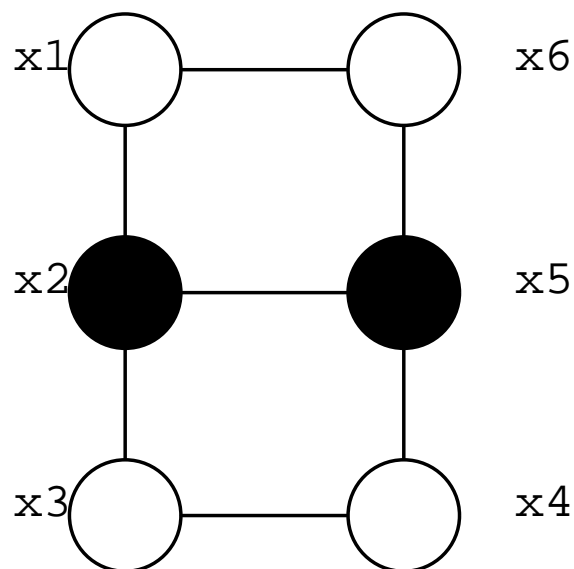
- Agents exchange ok? and improve? messages.
- When an ok? message is received, an agent evaluates currently-violated constraints.
- The value that minimizes the agents' constraints is sent in an improve? message.
- Once replies from all neighbors are received, if this agent's new value maximizes improvement, send the new value in an ok? message to all neighbors.
- Else send the old value in an ok? message.

# *Distributed Breakout Example*



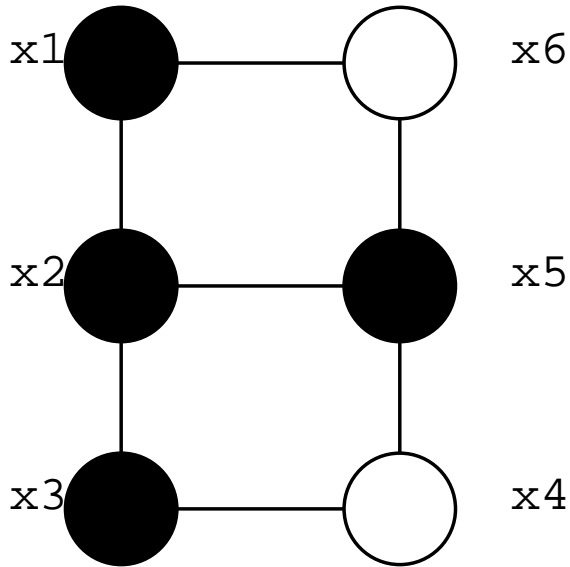
- Map-coloring problem with 2 colors.
- Initially, everyone sends an ok? message to their neighbors.
- All weights set to 1.
- Initially, no agent has a local improvement that will reduce the number of conflicts.

# *Distributed Breakout Example*



- Weights on current constraints are increased:
  - nogood: (x1 = x6 = white): 2
  - nogood: (x3 = x4 = white): 2
  - nogood: (x2 = x5 = black): 2
- Now, x1, x3, x5, x6 have an improve value of 1.
- x1 and x3 have the right to change (lower number)

# *Distributed Breakout Example*



- Now x2 has an improvement value of 4, and all other agents have improves of 0.
- So x2 changes to white, and the problem is solved.

# Summary

- DCSP is a nice example of a medium-coupled problem
  - Each agent has local constraints that it must satisfy.
  - Constraints also exist with other agents.
  - Variables are iteratively changed until no constraints are violated.
- Many problems fit into this framework
  - Scheduling, planning, coordination
- Challenges: expressing complex constraints, intelligent heuristics.