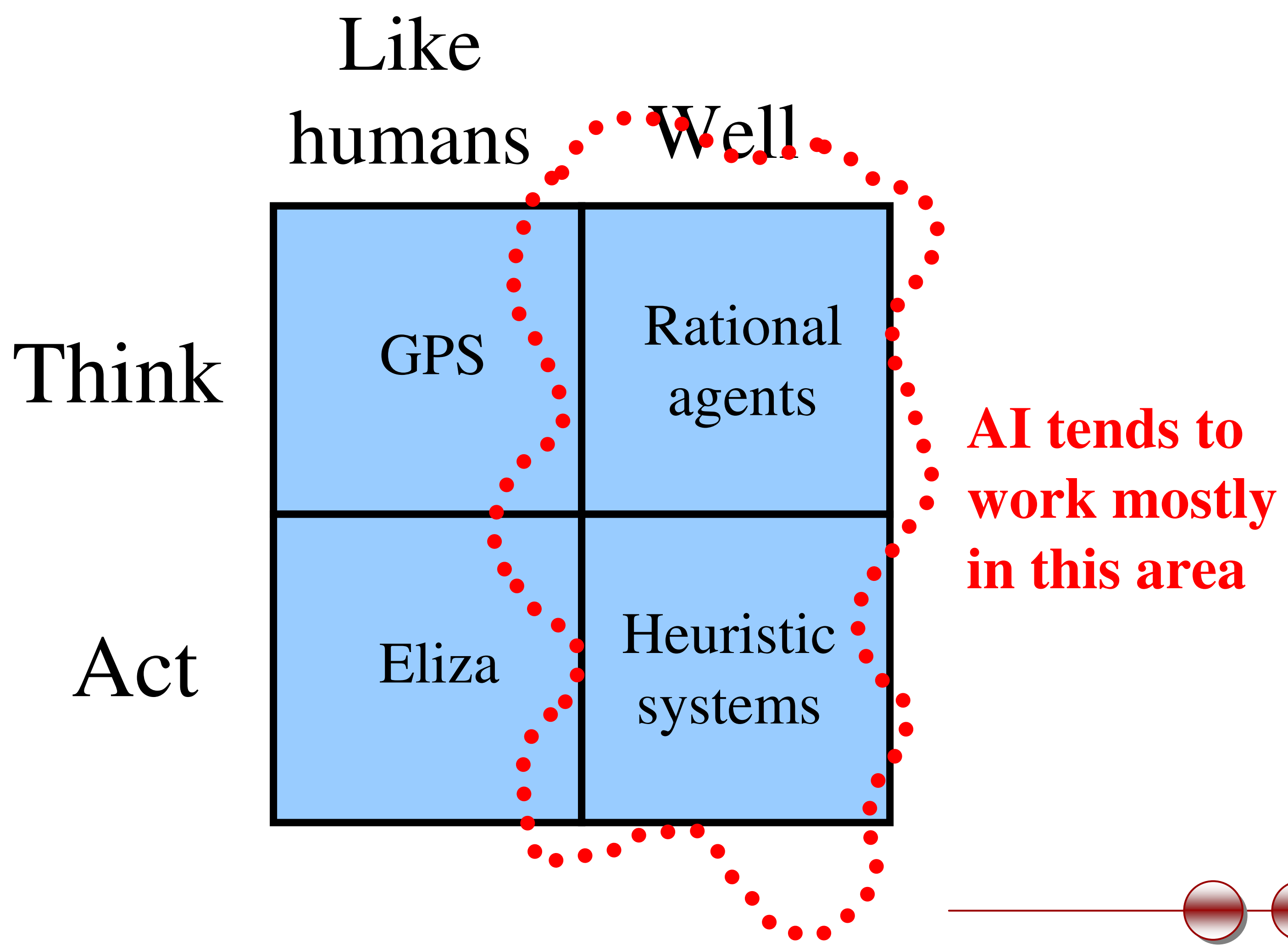
CMSC 671
Fall 2010

Tue 9/23/10

Review: Agents/Search/CSP/Adversarial Search
Advanced Search
Distributed Constraint Satisfaction

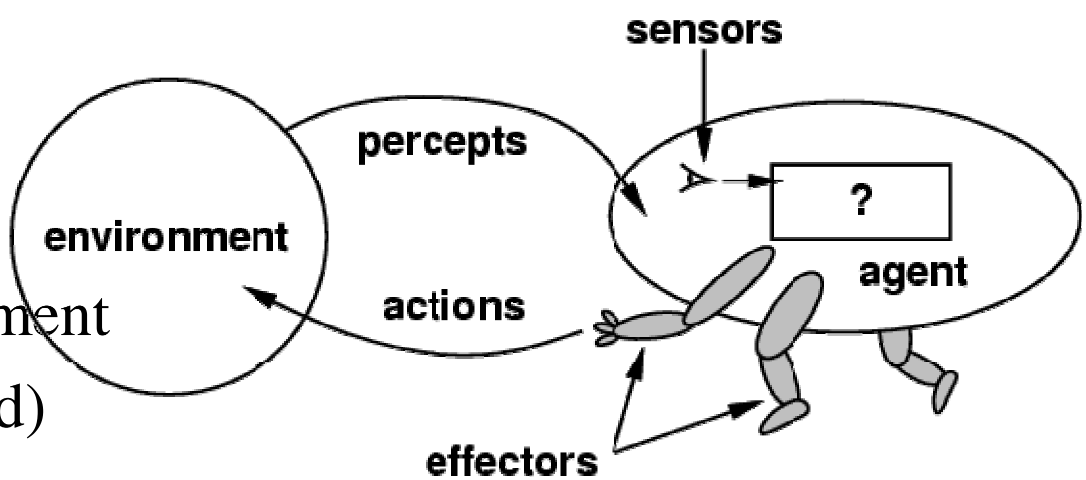
Prof. Laura Zavala, laura.zavala@umbc.edu, ITE 373, 410-455-8775

AI Possible Approaches



How do you design an intelligent agent?

- Definition: An **intelligent agent** perceives its environment via **sensors** and acts rationally upon that environment with its **effectors**.
- A discrete agent receives **percepts** one at a time, and maps this percept sequence to a sequence of discrete **actions**.
- Properties
  - Autonomous
  - Reactive to the environment
  - Pro-active (goal-directed)
  - Interacts with other agents via the environment



Rationality

- An ideal **rational agent** should, for each possible percept sequence, do whatever actions will maximize its expected performance measure based on

  (1) the percept sequence, and

  (2) its built-in and acquired knowledge.

Properties of Environments

- **Fully observable/Partially observable**
- **Deterministic/Stochastic**
- **Episodic/Sequential**
- **Static/Dynamic**
- **Discrete/Continuous**
- **Single agent/Multi-agent**

Fully observable + Deterministic ➔ no need to deal with uncertainty

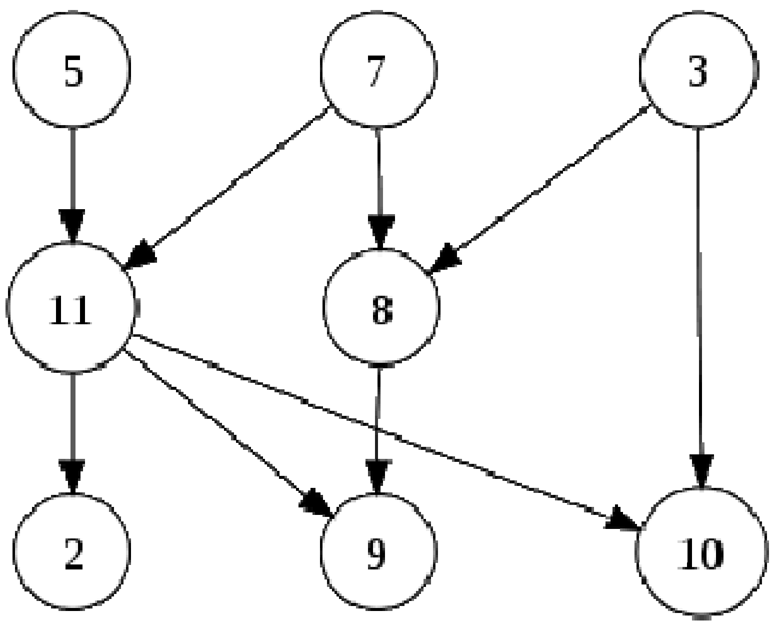
Characteristics of environments

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
Solitaire	No	Yes	Yes	Yes	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes	No
Taxi driving	No	No	No	No	No	No
Internet shopping	No	No	No	No	Yes	No
Medical diagnosis	**No**	**No**	**No**	**No**	**No**	**Yes**

→ Lots of real-world domains fall into the hardest case!

Formalizing search in a state space

- A state space is a **graph** (V, E) where V is a set of **nodes** and E is a set of **arcs**, and each arc is directed from a node to another node



- **cost** of operators
- **successor nodes** (legal operators)
- **expanding** a node
- **goal test**
- **solution** (sequence of operators)
- **cost/length of a solution**

State-space search is the process of searching through a state space for a solution by **making explicit** a sufficient portion of an **implicit** state-space graph to find a goal node.

State-space search algorithm

```
function general-search (problem, QUEUEING-FUNCTION)
;; problem describes the start state, operators, goal test, and operator costs
;; queueing-function is a comparator function that ranks two states
;; general-search returns either a goal node or failure
nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
                problem.OPERATORS))
 end
  ;; Note: The goal test is NOT done when nodes are generated
  ;; Note: This algorithm does not detect loops
```

Key procedures to be defined

- EXPAND
  - Generate all successor nodes of a given node
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion

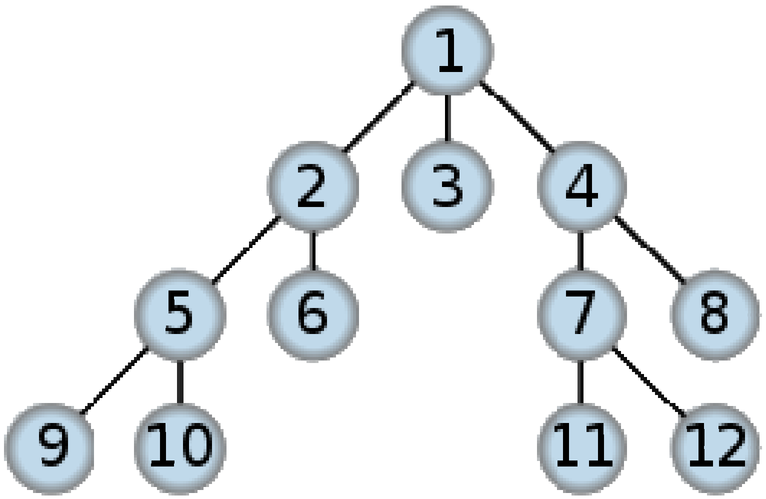
So far …

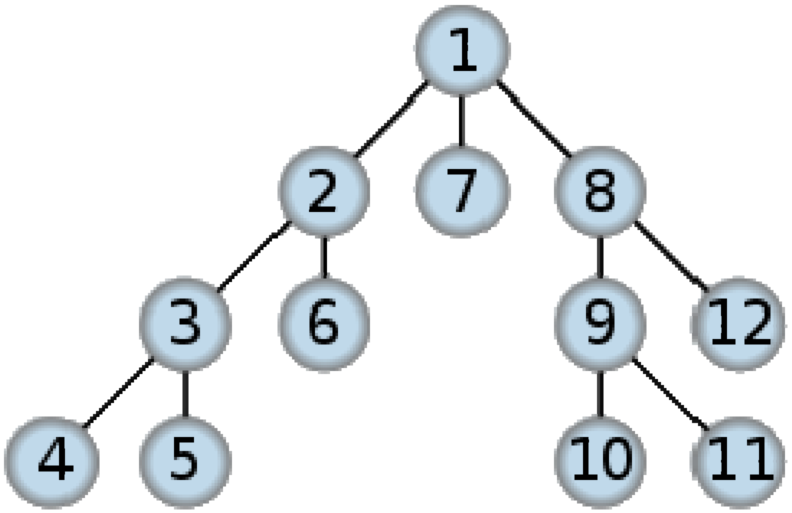
- **Uninformed search strategies**
  - No information about the likely “direction” of the goal node(s)
  - Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional
- **Informed search strategies (heuristic/best-first search)**
  - Use information about the domain to (try to) head in the general direction of the goal node(s)
  - Order nodes on the nodes list by increasing value of an evaluation function $f(n)$
  - Greedy search, beam search, A, A*
- **Local search / optimization problems**
  - No path to the goal
  - Hill-climbing algorithms, simulated annealing, local beam search, stochastic beam search, genetic algorithms, tabu search, online search
- **CSP**
  - **Set of variables** to which we have to assign **values** that satisfy a number of **problem-specific constraints.**
  - Generate and test, backtracking, constraint propagation (k-consistency), variable and value ordering heuristics (MRV, degree heuristic, LCV), forward checking, intelligent backtracking, local search for CSP
- **Adversarial Search**
  - Agents (players) need to consider the actions of other agents
  - Minimax, alpha-beta pruning, expectiminmax

Evaluating strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded
- **Space complexity**
  - How much space (memory) is used by the algorithm? Usually measured in terms of the maximum size of the “nodes” list during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?

Breadth-First vs Depth-First (DFS)





Breadth-First

Exponential time and space $O(b^d)$

Optimal if costs are the same

Depth-First

Exponential time $O(b^d)$

Linear space O(bd)

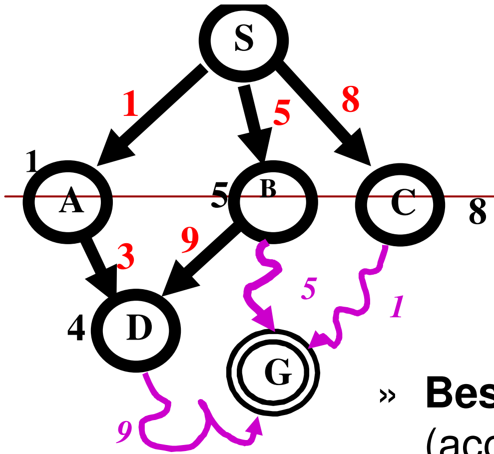
May not terminate

Uniform-Cost (UCS)

complete and optimal

Iterative Deepening

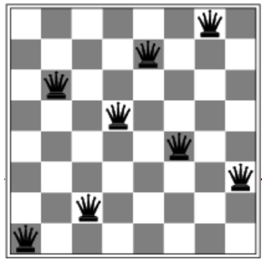
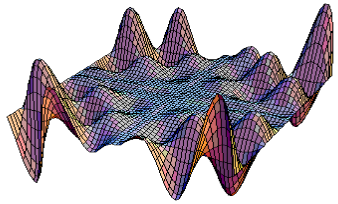
solves the infinite-path problem



Summary: Informed search

» **Best-first search** is general search where the minimum-cost nodes (according to some measure) are expanded first.

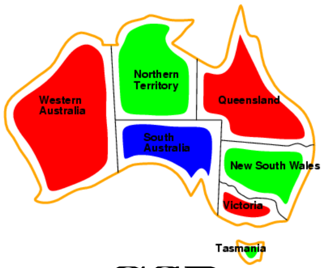
- **Greedy search** uses minimal estimated cost $h(n)$ to the goal state as measure. This reduces the search time, but the algorithm is neither complete nor optimal.
- **A* search** combines uniform-cost search and greedy search: $f(n) = g(n) + h(n)$. A* handles state repetitions and $h(n)$ never overestimates.
  - A* is complete and optimal, but space complexity is high.
  - The time complexity depends on the quality of the heuristic function.
  - IDA* and SMA* reduce the memory requirements of A*.

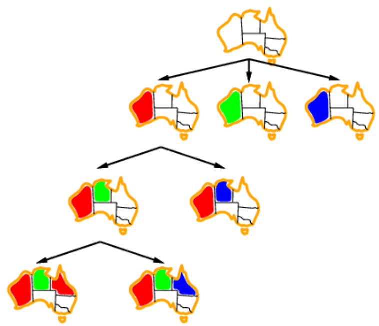
 

Summary: Local search

- **Hill-climbing algorithms** keep only a single state in memory, but can get stuck on local optima.
- **Simulated annealing** escapes local optima, and is complete and optimal given a "long enough" cooling schedule.
- **Local beam search** hill climbing but with the k best states
- **Stochastic beam search** keep a state with *p(h)*
- **Genetic algorithms** can search a large space by modeling biological evolution.
- **Tabu search** local search but with a memory (*k* previously visited states)
- **Online search** algorithms are useful in state spaces with partial/no information
  - Interleave computation and action (search some, act some)

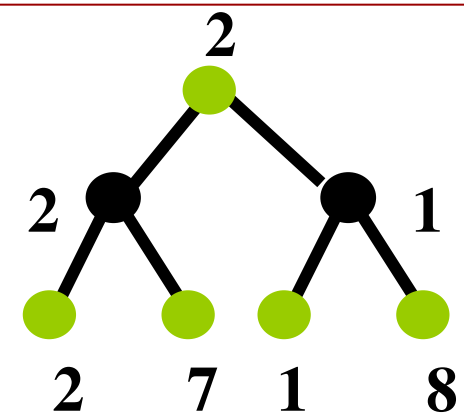
Summary: CSP



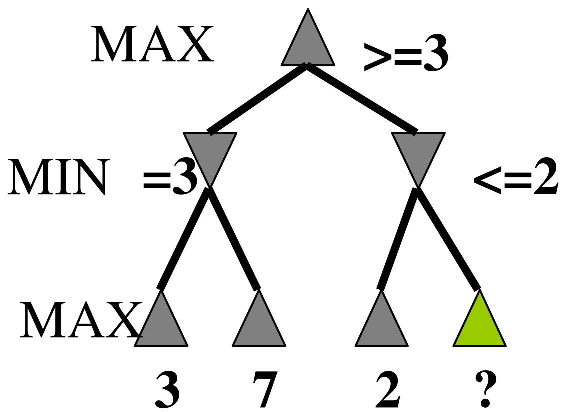


- **CSP**
  - **Generate and test** Try each possible combination
  - **Backtracking** Depth first search (choosing unassigned variable)
  - **Constraint propagation** Using the constraints to reduce the number of legal values for a variable
  - **Variable and value ordering heuristics** Minimum remaining values**,** degree heuristic: largest # of constraints on unassigned vars, Least constraining value
  - **Forward checking** Keep track of remaining legal values for unassigned variables
  - **Intelligent backtracking** Better than chronological (jump to most recent assignment , track of incompatible val. assignments, track of conflicting vars.)
  - **Local search for CSP** Incomplete states; operators reassign variable values
  - **Distributed constraint satisfaction** Dif. agents control dif. subset of the constraint variables

Summary: Adversarial Search



- **Evaluation function** is used to evaluate the "goodness" of a game position
- Game Trees
- Minimax
  - If it is **my turn** to move, then the root is labeled a "**MAX**" node; otherwise it is labeled a "**MIN**" node, indicating **my opponent's turn.**
  - Expand nodes down; "Back up" values
- Alpha-beta pruning
  - Don't compute unnecessary nodes
- Expectiminmax (Games of chance)
  - Use minimax to compute values for MAX and MIN nodes
  - Use **expected values** for chance nodes



Advanced Search

Overview

- Real-time heuristic search
  - Learning Real-Time A* (LRTA*)
  - Minimax Learning Real-Time A* (Min-Max LRTA*)
- Genetic algorithms

REAL-TIME SEARCH

Real-Time Search

Sven Koenig, "Real-time heuristic search: Research issues," In *Proceedings of the AIPS-98 Workshop on Planning as Combinatorial Search: Propositional, Graph-Based, and Disjunctive Planning Methods,* pages 75-79, 1998.

- Interleave search and execution
- Advantages:
  - Provide variable control over amount of search (deliberation) vs. execution (reactivity)
  - Improves performance over successive trials of the same problem: **learn** to improve quality of heuristic function
  - Can solve very large problems (if they have the right problem structure)
  - Short planning time per move (ind. of # states)

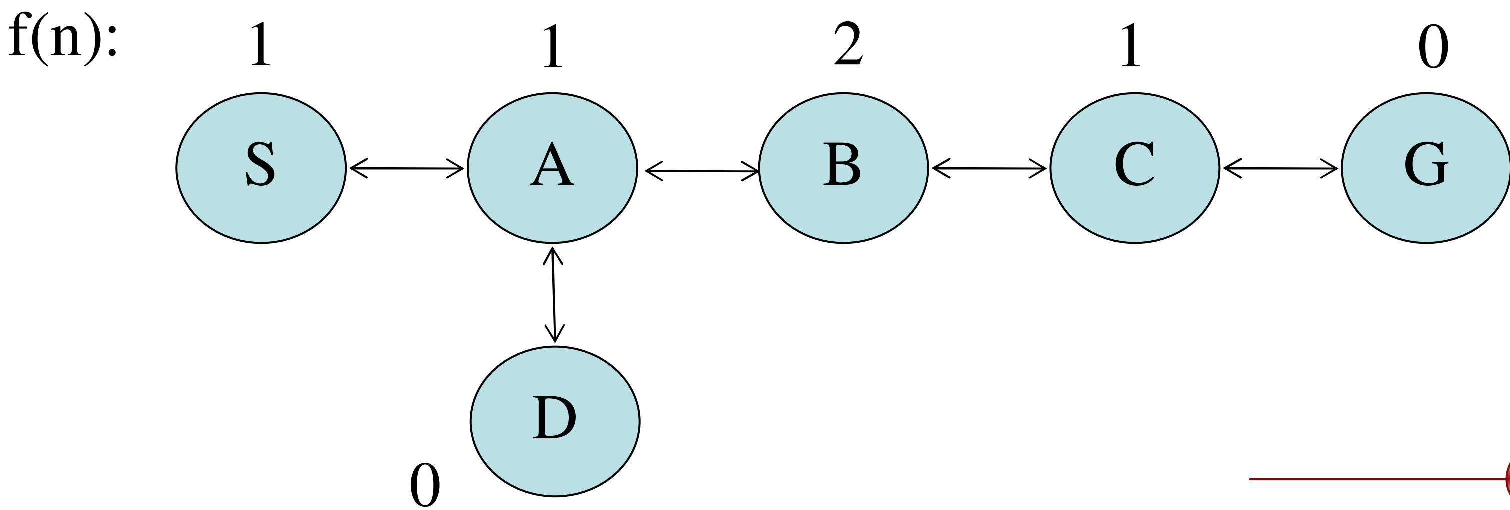
LRTA*

Richard E. Korf, "Real-time heuristic search," *Artificial Intelligence* 42(2-3): 189-211, March 1990.

- Simplest version: one step lookahead with heuristic-value updating:
  1. Initialize s to the start state
  2. If s is a goal state, stop
  3. Choose an action a that minimizes $f(succ(s,a))$
  4. Update $f(s)$ to the max of current $f(s)$, $1+f(succ(s,a))$
  5. Execute action a
  6. Set s to the current state
  7. Go to step 2

Search Example

- What will each algorithm do?
  - Greedy search (with and without repeated states)
  - A* (with and without repeated states)
  - Hill-climbing
  - (One-step-lookahead) LRTA*



Min-Max LRTA*

Sven Koenig, "Minimax real-time heuristic search," *Artificial Intelligence* 129 (1-2): 165-197, June 2001.

- Variation of LRTA* that can be used in nondeterministic domains: the agent is not able to predict with certainty which successor state an action execution results in
- Simulated robot-navigation tasks in mazes
  1. Initialize *s* to the start state
  2. If *s* is a goal state, stop
  3. Choose an action *a* ***whose worst possible outcome*** minimizes *f(succ(s,a))* ***(minimax step)***
  4. Update *f(s)* to the max of current *f(s), 1+f(succ(s,a))* ***(across all possible successors of s when performing a)***
  5. Execute action *a*
  6. Set *s* to the current state
  7. Go to step 2

Incremental Heuristic Search

- Reuse information gathered during A* to improve future searches
- Variations:
  - Failure ➔ restart search at the point where the search failed
  - Failure ➔ update *h*-values and restart search
  - Failure ➔ update *g*-values and restart search
- Fringe Saving A*, Adaptive A*, Lifelong Planning A*, DLite*...

GENETIC ALGORITHMS

Genetic Algorithms

- Active area of research. Many applications. Annual conferences and workshops
- Probabilistic search/optimization algorithm
- Mimic the process of natural evolution
- Start with *k* random states (the *initial population)*
- Generate new states by “mutating” a single state or “reproducing” (combining via crossover) two parent states
- Selection mechanism based on children’s *fitness* values

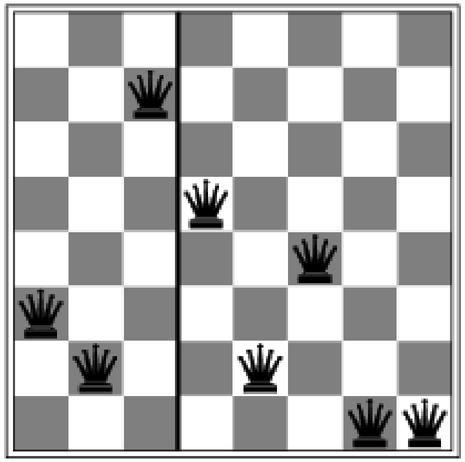
GA: Genome Encoding

- Each variable or attribute is typically encoded as an integer value
  - Number of values determines the granularity of encoding of continuous attributes
- For problems with more complex relational structure:
  - Encode each aspect of the problem
  - Constrain mutation/crossover operators to only generate legal offspring

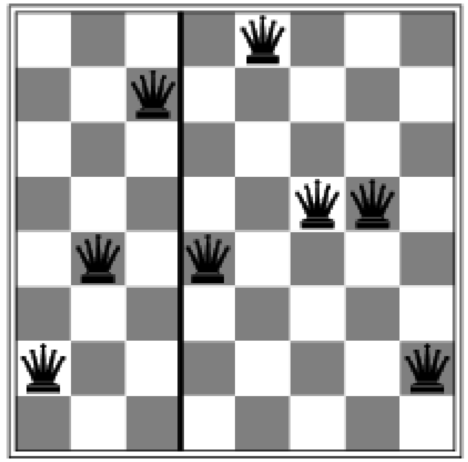
Genetic Algorithms (2)

- Encoding used for the “genome” of an individual strongly affects the behavior of the search
- Most effective in situations, for which a well-defined problem offers a compact encoding of all necessary solution parameters

Genetic algorithms: Example



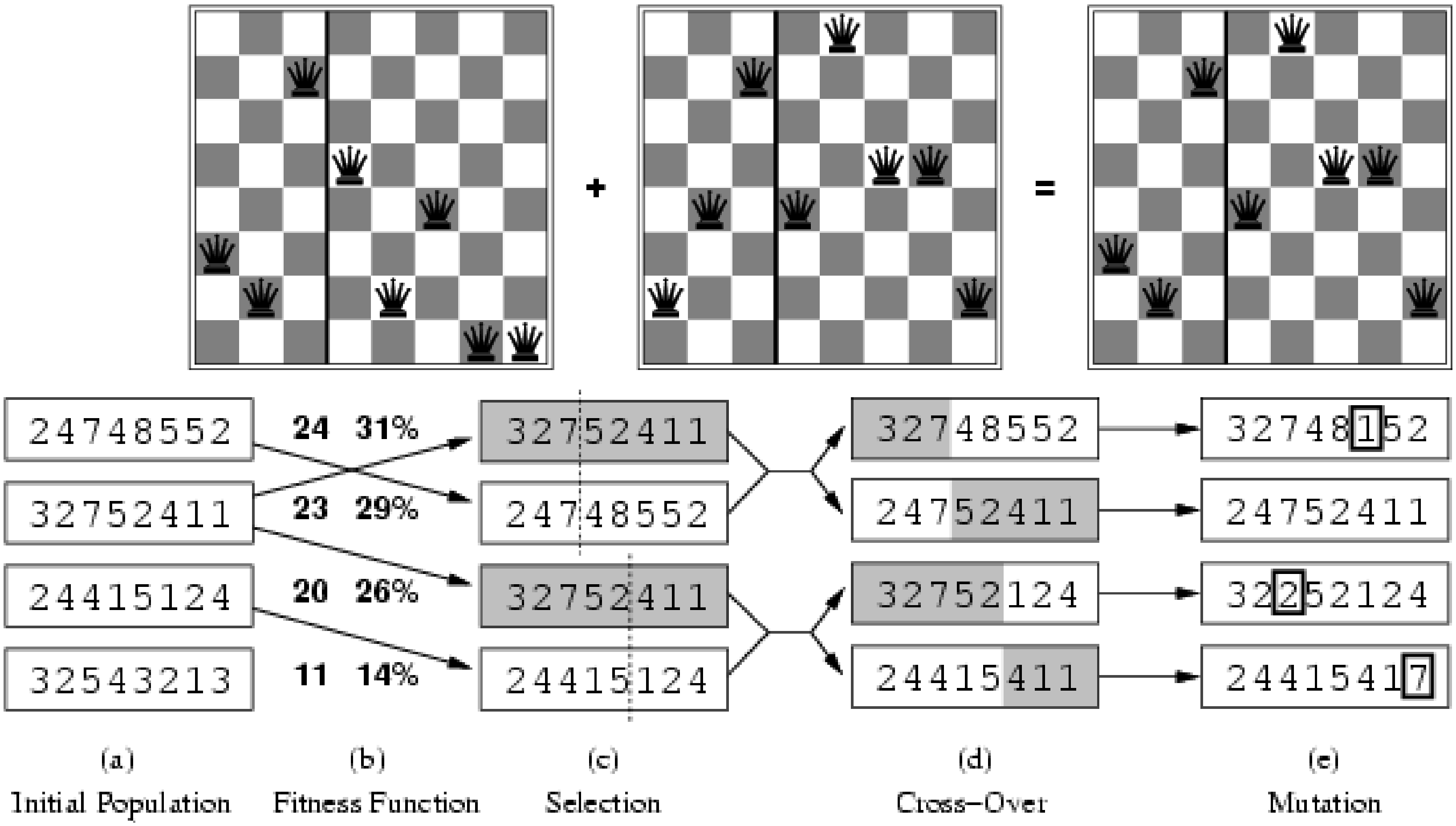
32752411
23



24748552
24

- Fitness function: number of non-attacking pairs of queens
  - min = 0, max = 8 × 7/2 = 28

Genetic algorithms: Example (2)



Fitness-based stochastic selection :

P(24748552) = 24/(24+23+20+11) = 31%

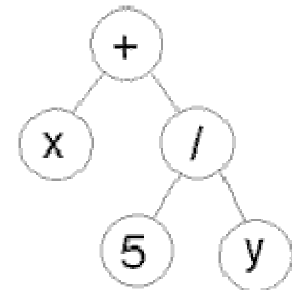
P(32752411) = 23/(24+23+20+11) = 29%

Exercise

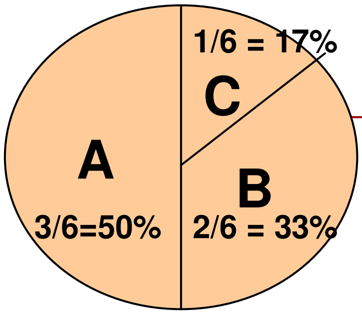
- Design a genetic algorithm to find a perfect Tic-Tac-Toe strategy, which never loses a game it plays.

Encoding

- Binary Encoding
  - 101100101100101011100101
  - 111111100000110000011111
- Permutation Encoding
  - 1 5 3 2 6 4 7 9 8
  - 8 5 6 7 2 3 1 4 9
- Value Encoding
  - 1.2324 5.3243 0.4556 2.3293 2.4545
  - BABDJEIFJDHDIERJFDLDFLFEGT
  - (back), (back), (right), (forward), (left)
- Tree Encoding



(+ x (/ 5 y))



Selection Mechanisms

- Proportionate selection: Each offspring should be represented in the new population proportionally to its fitness
  - Roulette wheel selection (stochastic sampling): Random sampling, with fitness-proportional probabilities. Better individuals get higher chance
  - Deterministic sampling: Exact numbers of offspring (rounding up for most-fit individuals; rounding down for “losers”)
- Tournament selection: Offspring compete against each other in a series of competitions
  - Particularly useful when fitness can’t be readily measured (e.g., genetically evolving game-playing algorithms or RoboCup players)

GA: Crossover

- Selecting parents: Pick pairs at random, or fitness-biased selection (e.g., using a Boltzmann distribution)

- One-point crossover (swap at same point in each parent)
- Two-point crossover
- Cut and splice (cut point could be different in the two parents)
- Bitwise crossover (“uniform crossover”)

- ***Many*** specialized crossover methods for specific problem types and representation choices

GA: Mutation

- Bitwise ("single point") mutation
- Order changing - two numbers are selected and exchanged
- Adding a small number (for real value encoding)
- Change selected nodes (tree representation)

GA: When to Stop?

- After a fixed number of generations
- When a certain fitness level is reached
- When fitness variability drops below a threshold
- ...

GA: Parameters

- Running a GA involves many parameters
  - Population size
  - Crossover rate
  - Mutation rate
  - Number of generations
  - Target fitness value
  - ...

Exercise

- Design a genetic algorithm to find a perfect Tic-Tac-Toe strategy, which never loses a game it plays.
- "On the Genetic Evolution of a Perfect Tic-Tac-Toe Strategy", Gregor Hochmuth, Stanford University

  http://www.genetic-programming.org/sp2003/Hochmuth.pdf

DISTRIBUTED CONSTRAINT SATISFACTION

Distributed Constraint Satisfaction

- Looks at solving CSP when there is a collection of agents, each of which controls a subset of the constraint variables.

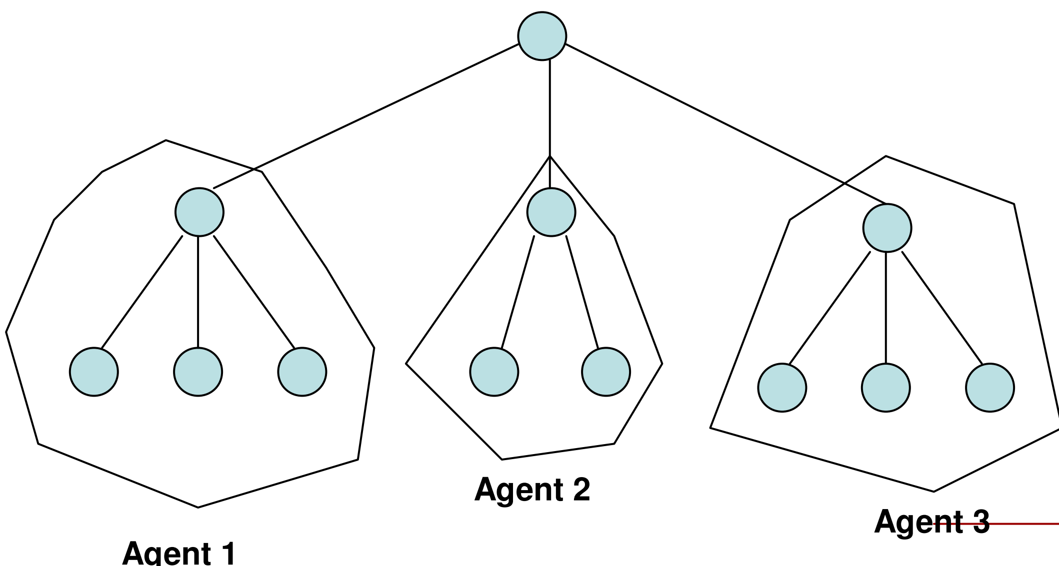
- Active area of research; annual workshops.

Why multiple agents?

- Agents have limited rationality
  - search is often intractable
  - may not have a complete picture of the problem
  - may not have the required computational capability
- Agents may be self interested

DCSP: Approach

- If we represent the search problem as a graph, we can solve it by accumulating local computations for each node in the graph
  - Local computations can be executed asynchronously and concurrently

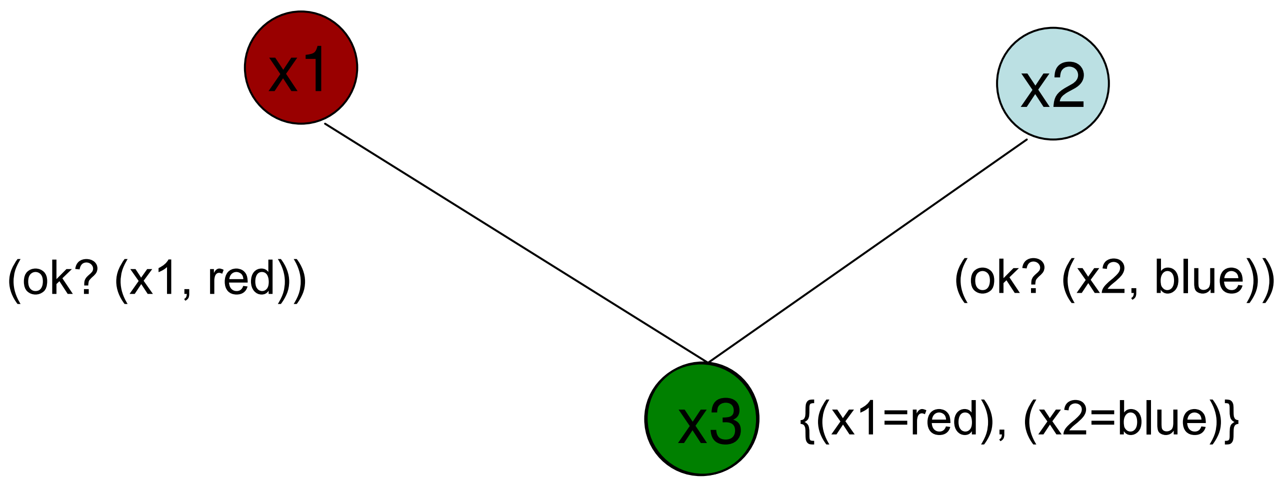


Asynchronous Backtracking

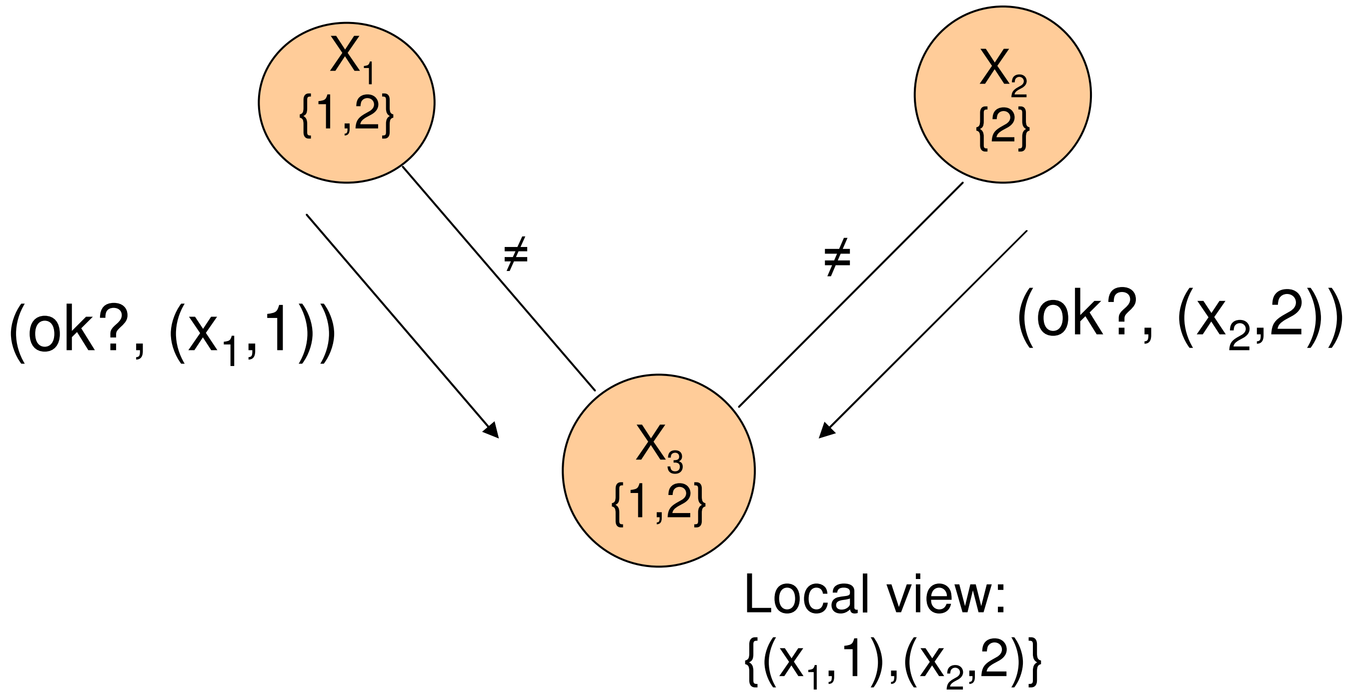
- The processes are priority ordered (by the alphabetical order of the variable identifiers)
- Each process chooses an assignment and communicates it to the neighboring processes (**ok message**)
- Each process maintains the current value of other processes from its viewpoint (**local view**)
  - A value assignment is changed if it is not consistent with the assignments of the higher priority processes
  - If no values are consistent with the higher priority processes, then the process creates a **nogood message** and sends it to the higher priority processes
- All agents wait for and respond to messages

Asynchronous Backtracking Example

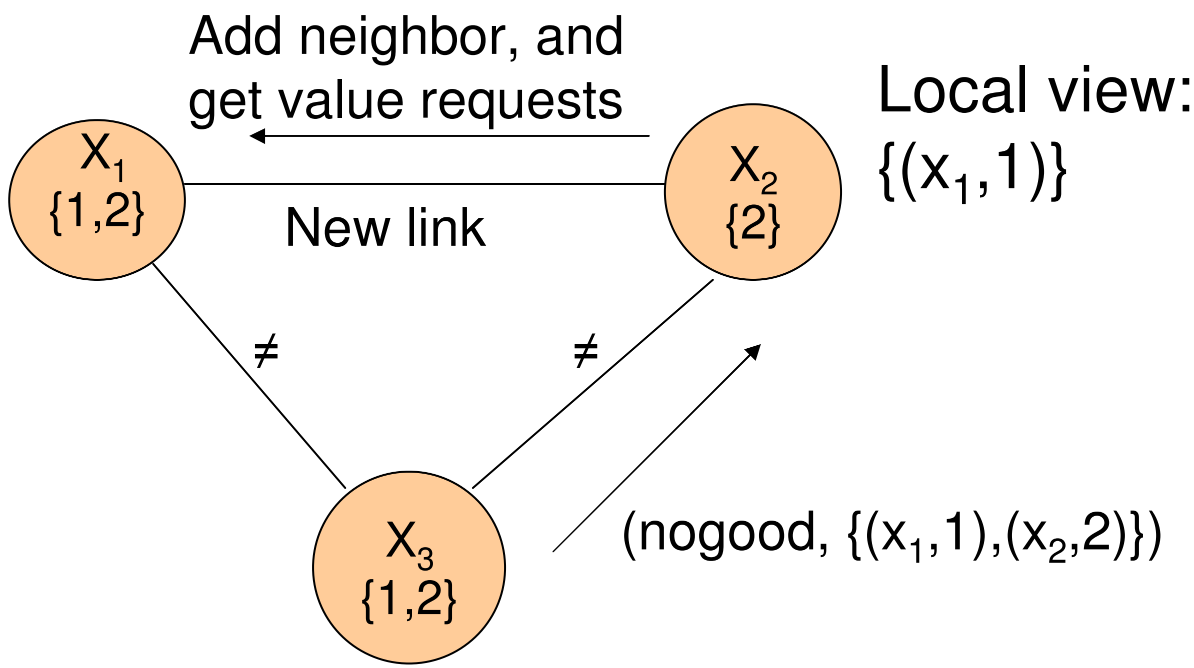
x1, x2, x3 {red, blue, green}
x1≠x3, x2≠x3,



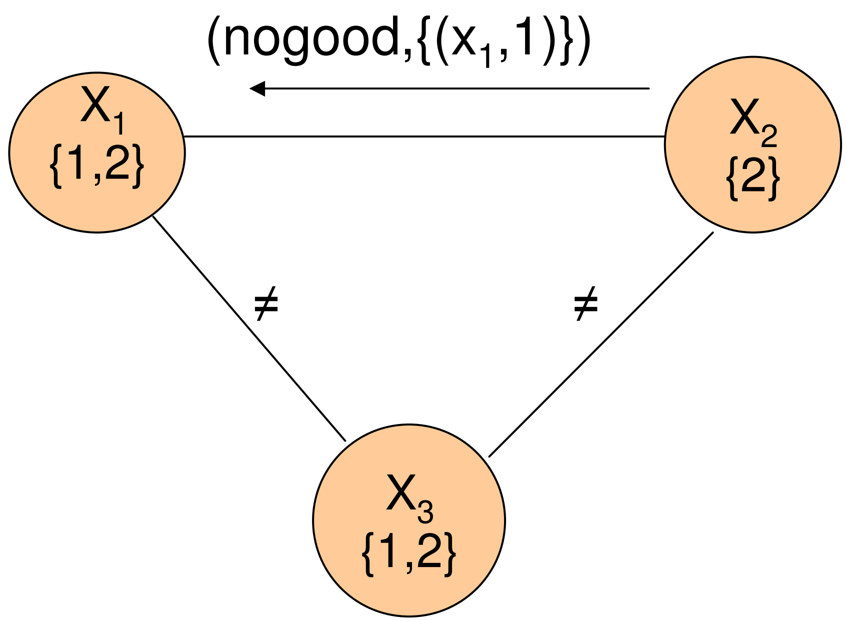
Asynchronous Backtracking Example 2



Asynchronous Backtracking Example 2



Asynchronous Backtracking Example 2



Asynchronous Weak-Commitment

- Asynchronous backtracking
  - Process priorities are statically determined
  - Higher priority processes can make a poor value assignment resulting in the lower level process having to do a long search in order to reverse the higher level process' decision
- An improved alternative: AWC

Asynchronous Weak-Commitment (2)

- AWC allows dynamic reordering of the process priorities so that a bad decision can be revised without an exhaustive search.
- Use a value ordered heuristic
  - i.e. min-conflict heuristic – minimize the number of constraints violations.

DCS: Algorithms

- 1992—Asynchronous Backtracking (ABT), -static ordering, complete
- 1994—Asynchronous Weak-Commitment (AWC), -reordering, fast, complete (only with exponential space)
- 1995—Distributed Breakout Algorithm (DBA), -incomplete but fast
- 2000—Distributed Forward Chaining (DFC), -slow, comparable to ABT
- 2000—Asynchronous Aggregation Search (AAS), -aggregation of values in ABT
- 2001—Maintaining Asynchronously Consistencies (DMAC), -the fastest algorithm
- 2001—Asynchronous Backtracking with Reordering (ABTR), -reordering in ABT with bounded nogoods
- 2002—Secure Computation with Semi-Trusted Servers, -security increases with the number of trustworthy servers
- 2003—Secure Multiparty Computation For Solving DisCSPs (MPC-DisCSP1-MPC-DisCSP4), secure if 1/2 of the participants are trustworthy

Also …

- You can check my implementation of the Asynchronous Weak-Commitment Search for the n-queens problem:
- http://www.cs.umbc.edu/~rzavala/netlogomas.html

- Or other implementations - Graph Coloring, asynchronous backtracking:
- http://jmvidal.cse.sc.edu/netlogomas/ABTgc.html