Lecture 6: Search - 5

Victor Lesser

CMPSCI 683
Fall 2004

Today's lecture

Local Search

- **Hill-Climbing/Iterative Improvement**
  - Simulated Annealing (Stochastic Hill Climbing)
- **Beam Search**
  - Genetic Algorithm

Local Search

- **Operate using a single current state**
  - Paths followed by search are not retained
    - Contrast with open and closed node lists; search tree
  - Little memory required -- usually a constant
- **Sometimes the path to the goal is irrelevant:**
  - 8-queens problem, job-shop scheduling
  - circuit design, computer configuration
  - automatic programming, automatic graph drawing
- **Optimization problems may have no obvious "goal test" or "path cost".**
  - Continuous functions

Advantages of local search

- **Very simple to implement.**
- **Very little memory is needed.**
- **Can often find reasonable solutions in very large (continuous) state spaces for which systematic algorithms are not suitable.**

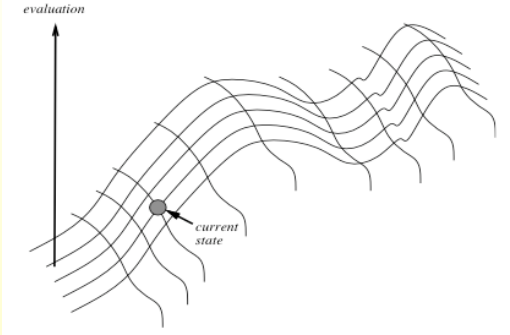
Stochastic vs. Systematic Search

- **Unsolvability -- Is there a solution?**
  - Systematic: can require exhaustive examination of exponential search space
  - Stochastic: cannot determine unsolvability
- **Completeness/Optimality**
  - Systematic: complete
  - Stochastic: incomplete
- **Speed**
  - Neither is uniformly superior; each does better for different sorts of problems

Local Search is an example of Stochastic Search

Iterative Improvement
(Smart version of Generate & Test)

- **Start Search with complete but non-optimal solution**
- **Modify incorrect/non-optimal solution to move it closer to correct/optimal solution**



Iterative Improvement Algorithms
(Smart version of Generate & Test)

- **What is the search space**
  - Search space of complete solutions vs. partial solutions
- **When useful:**
  - "reasonable" complete solution can be generated
  - Operator to modify complete solution
  - Some notion of progress
    - Measure of complete solution in terms of constraint violations or an evaluate function

Hill-Climbing Search

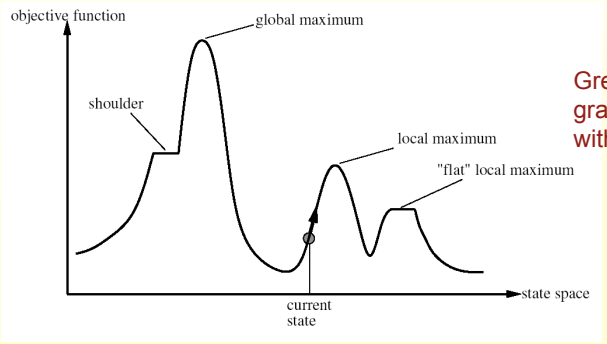
- **The main iterative improvement algorithm is hill-climbing:**

  Continually move in the direction of increasing value of all successor states until a maximum is reached.

  - Trade-off between time to select a move (more) and time to reach goal state (less)
- **This is sometimes called steepest-ascent HC, and is called gradient descent search if the evaluation function is based on cost rather than quality.**

Hill-climbing search

A simple form of local search: Continually move in the direction of increasing value.



Greedy Local search; grabs good neighbor without thinking ahead

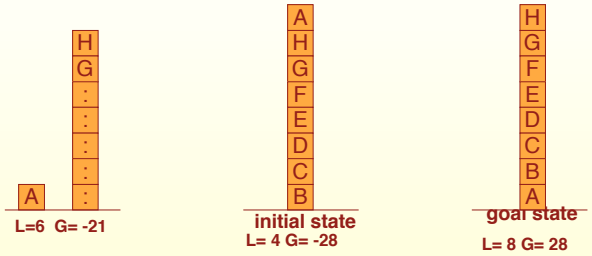
Steepest Ascent Hill-Climbing

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  inputs: *problem*, a problem
  local variables: *current*, a node
    neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[neighbor] < VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*
  end

Looks at all successors

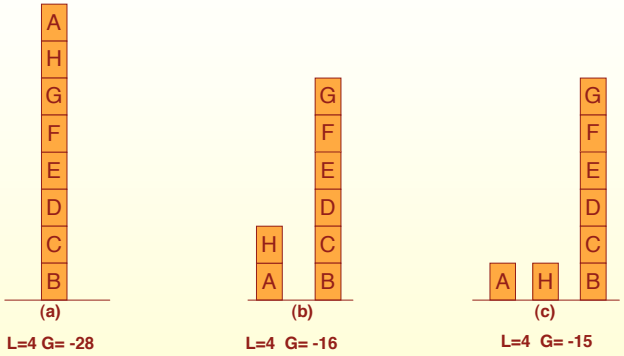
An Example of Hill-Climbing Problems



Local: Add 1 point for every block that is resting on the thing it is supposed to be resting on. Subtract 1 point for every block that is sitting on the wrong thing.

Global: For each block that has the correct support structure (i.e., the complete structure underneath it is exactly as it should be), add 1 point for every block in the support structure. For each block that has an incorrect support structure, subtract one point for every block in the existing support structure.

An Example of Hill-Climbing Problems (cont'd)



Local criterion results in no available moves that increase evaluation

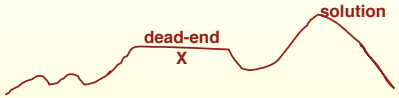
Problems with Hill Climbing

- Can get stuck at a local maximum.

Local maximum

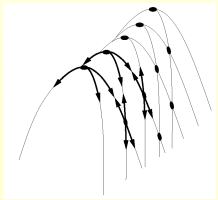
- Unable to find its way off a plateau.

Plateau



- Cannot climb along a narrow ridge when almost all steps go down (continuous space).

Ridge/Knife edges



Variants of hill-climbing

Ways to overcome the weaknesses:

- Stochastic hill-climbing (Simulated Annealing): choose at random an uphill move among the successors
  - Sometimes take downhill steps
  - *You may have to get worse to get better!!*
- First-choice hill climbing: generate successors randomly until finding an uphill move
- Random-restart hill climbing: restart search from randomly generated initial states

Simulated Annealing

"Simulated annealing is a variation of hill climbing in which, at the beginning of the process, some downhill moves may be made. The idea is to do enough exploration of the whole space early on so that the final solution is relatively insensitive to the starting state. This should lower the chances of getting caught at a local maximum, a plateau, or a ridge."

- Randomly jump to alternative non-locally optimal moves based on $p' = e^{-\Delta E/T}$
  - more time goes, less willing to explore non-optimal path

Simulated Annealing Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Initialize *BEST-SO-FAR* as the current state.
3. Initialize *T* according to the annealing schedule

Simulated Annealing Algorithm (cont'd)

4. **Loop until a solution is found or until there are no new operators left to be applied in the current state.**

   a) Select an operator (randomly) that has not yet been applied to the current state and apply it.

   b) Evaluate the new state. Compute
   ΔE = (value of current) - (value of new state)
   - **If the new state is a goal state, return it and quit.**
   - **If it is not a goal state but is better than the current state, then make it the current state.**
     - **set *BEST-SO-FAR* to this new state if better than current *BEST-SO-FAR* .**
   - **If it is not better than the current state, then make it the current state with probability *p´*.**

   c) Revise *T* according to the annealing schedule

5. **Return *BEST-SO-FAR* as the answer.**

Simulated Annealing Algorithm

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Hill Climbing vs Beam Search

- **Keep track of K states rather than just one**
  - Modified breadth-first, contour created dynamically
- **Start with K randomly generated states**
- **Stop if any goal state**
- **Multiple Successors generated for each of the k states**
- **Choose top K successor states in next cycle**

- **Contrast with Random Restart**
  - Positive -- Sharing information across different searches
  - Negative - May eliminate diversity coming from random starting points

Evolutionary Computation

- **Beam Search patterned after biological evolution**
  - Learning as Search

- **Metaphor of Natural Selection**
  - Offspring are similar to their parents
  - The more fit are more likely to have children
  - Occasionally random mutations occur

Genetic (Search) Algorithms

- **Localized Beam Search**
  - Specialized approach for generating successors and for selecting next states
  - An individual solution is represented by a sequence of "genes".
  - The selection strategy is randomized with probability of selection proportional to "fitness".
  - Individuals selected for reproduction are randomly paired, certain genes are crossed-over, and some are mutated.

Basic Operation of Genetic Search

- **Selection**
  - More fit members are likely to be in next generation
- **Mutation**
  - Random altering of characteristics
- **Crossover**
  - Combine two members of population
    - Cross-over is probably the key idea
    - ***Exploit relative independence of certain subproblem solutions imbedded in different members***

Key Questions

–What is the fitness function?

–How is an individual represented?

–How are individuals selected?

–How do individuals reproduce?

Solutions as Binary Strings

Represent

$$(Outlook = Overcast \vee Rain) \wedge (Wind = Strong)$$

by

Outlook	*Wind*
011	10
→ Sunny(no)	→ Weak(no)

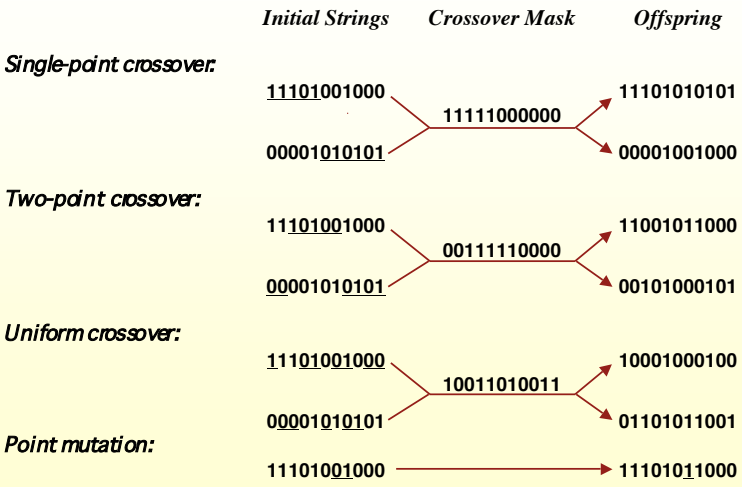
Represent

IF *Wind=Strong* THEN *PlayTennis=yes*

by

Outlook	*Wind*	*PlayTennis*
111	10	10

Operators for Genetic Algorithms



GA (*Fitness, Fitness_threshold, p, r, m*)

GA(*Fitness*, *Fitness_threshold*, *p*, *r*, *m*)

- *Initialize*: $P \leftarrow p$ random hypotheses
- *Evaluate*: for each h in P, compute $Fitness(h)$
- While $[\max_h \ Fitness(h)] < Fitness\_threshold$

1. *Select*: Probabilistically select $(1 - r)p$ members of P to add to P_s.

$$\Pr(hi) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

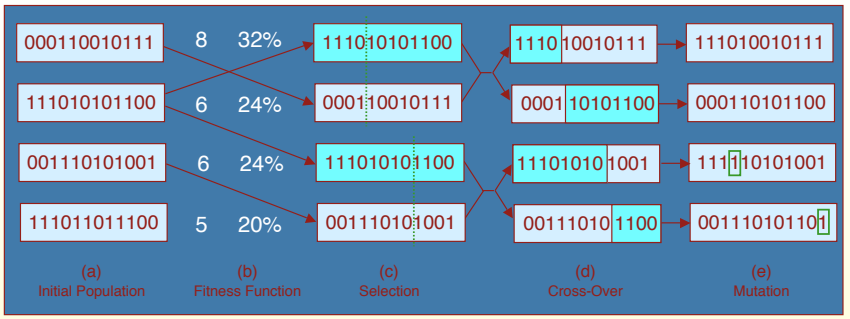
2. *Crossover*: Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add al offspring to P_s.

...continued from previous slide

3. ***Mutate:*** **Invert a randomly selected bit in $m \cdot p$ random members of P_s**
4. ***Update:*** **$P \leftarrow P_s$**
5. ***Evaluate:*** **for each *h* in *P*, compute *Fitness(h)***

- **Return the hypothesis from *P* that has the highest fitness.**

Figure: The genetic algorithm



In (a) we have an initial population of 4 individuals. They are scored by the fitness function in (b); the top individual scores an 8 and the bottom scores a 5. It works out that the top individual has a 32% chance of being chosen on each selection. In (c), selection has given us two pairs of mates, and the cross-over points (dotted lines) have been chosen. Notice that one individual mates twice; one not at all. In (d), we see the new offspring, generated by cross-over of their parents' genes. Finally in (e), mutation has changed the two bits surrounded by boxes. This gives us the population for the next generation.

Selecting Most Fit Hypotheses

***Fitness proportionate selection*:**

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

…can lead to *crowding (lack of diversity)*

- ***Tournament Selection*:**
  - Pick h_1, h_2 at random with uniform probability.
  - With probability p, select the more fit
- ***Rank Selection*:**
  - Sort all hypotheses by fitness
  - Probability of selection is proportional to rank

Gabil [DeKong et al. 1993]

Learn disjunctive set of propositional rules, competitive with C4.5

Fitness:

$$Fitness(h)=(correct(h))^2$$

Representation:

IF a_1=T $\wedge$ a_2=F THEN c=T; IF a_2=T THEN c=F

represented by

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

Genetic Operators:

- Want variable length rule sets as solutions
- Want only well-formed bitstring hypotheses

Crossover with Variable Length Bitstrings

Start with

	a_1	a_2	c	a_1	a_2	c
h_1:	10	01	1	11	10	0
h_2:	01	11	0	10	01	0

1. Choose crossover points for h_1, e.g. after bits 1, 8
2. Now restrict points in h_2 to those that produce bitstrings with well-defined semantics, e.g., ⟨1, 3⟩, ⟨1, 8⟩, ⟨6, 8⟩.

If we choose ⟨1, 3⟩ , result is

	a_1	a_2	c	a_1	a_2	c
h_3:	11	10	0	11	10	0
	a_1	a_2	c	a_1	a_2	c
h_4:	00	01	1	10	01	0

Gabil Extensions

Add new genetic operators, also applied probabilistically:

1. ***Add Alternative:*** **generalize constraint on a_i by changing a *0* to *1***
2. ***Drop Condition:*** **generalize constraint on a_i by changing every *0* to *1*.**

And, add new field to bitstring to determine whether to allow these

a_1	a_2	c	a_1	a_2	c	AA	DC
01	11	0	10	01	0	1	0

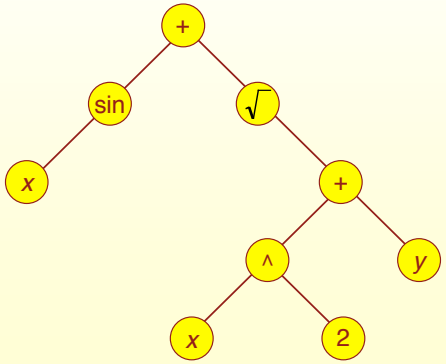
So now the learning strategy evolves!

GABIL Results

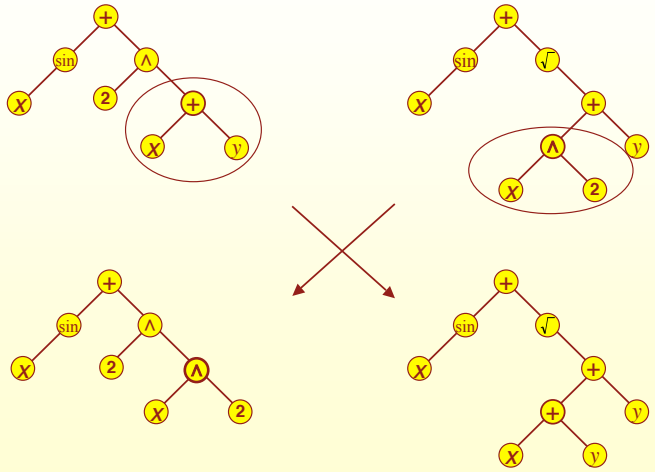
- Performance of GABIL comparable to symbolic rule/tree learning methods C4.5, ID5R, AQ14
- Average performance on a set of 12 synthetic problems:
  - GABIL *without* *AA* and *DC* operators: 92.1% accuracy
  - GABIL *with* *AA* and *DC* operators: 95.2% accuracy
  - Symbolic learning methods ranged from 91.2 to 96.6

Genetic Programming

Population of programs represented by trees $\sin(x) + \sqrt{x^2 + y}$



Crossover



Five Major Preparatory Steps

- Determining the set of terminals
- Determining the set of functions
- Determining the fitness measure
- Determining the parameters
- Determining the method for designating a result and the criterion for terminating a run.

Learning Boolean Even-k-Parity Function

1. TERMINAL SET T

T = {D0, D1, D2, … , Dk}

2. FUNCTION SET F

F = {AND, OR, NAND, NOR}

3. FITNESS MEASURE

The fitness cases are the 2^k combinations of the k terminals in T.

The standardized fitness of an S-expression is the sum, over the 2^k fitness cases, of the error (Hamming distance) between the Boolean value returned by the S-expression and the correct value of the target function (Boolean even-k-parity function).

Learning Boolean Even-k-Parity Function, p2

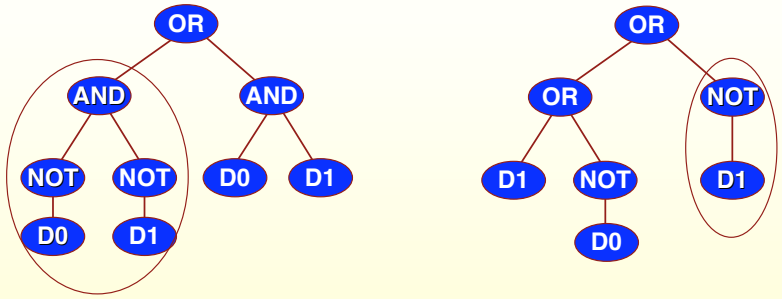
4. CONTROL PARAMETERS

- Population Size M = 4,000
- Generations G = 51

5. TERMINATION CRITERIA AND RESULT DESIGNATION

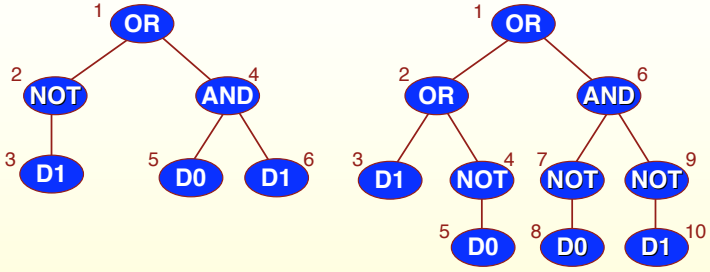
- Terminate when fitness equals 0 (2^k hits) or after G = 51 generations
- Designate best-so-far individual as the solution

Two Offspring



(OR (AND (NOT D0) (NOT D1))
(AND D0 D1))
(OR (OR D1 (NOT D0))
(NOT D1))

Two Parents in Crossover



(OR (NOT D1)
(AND D0 D1))
(OR (OR D1 (NOT D0))
(AND (NOT D0) (NOT D1))

Genetic Programming

More interesting example: design electronic filter circuits

- Individuals are programs that transform beginning circuit to final circuit, by adding/subtracting components and connections
- Use population of 640,000, run on 64-node parallel processor
- Discovers circuits competitive with best human designs

Summary: Genetic algorithms

- Have been applied to a wide range of problems.
- Results are sometimes very good and sometimes very poor.
- The technique is relatively easy to apply and in many cases it is beneficial to see if it works before thinking about another approach.

Next lecture

- **Repair/Debugging**
  - GSAT Interacting Subproblems
- **More on CSP problems**
  - texture measures
- **Multi-level Search**
  - blackboard