Iterative Improvement Search

Hill Climbing, Simulated Annealing, WALKSAT, and Genetic Algorithms.

Andrew Moore, Justin Boyan, Vince Cicirello

www.cs.cmu.edu/~awm

http://ic-www.arc.nasa.gov/people/jboyan

www.cs.cmu.edu/~vincent

Informal characterization

These are problems in which...

- There is some combinatorial structure being optimized
- There is a cost function: Structure --> Real, to be optimized, or at least a reasonable solution is to be found.
- (So basic CSP methods only solve part of the problem...they satisfy constraints but don't look for optimal constraint-satisfier)
- Searching all possible structures is intractable.
- Depth first search approaches are too expensive.
- There's no known algorithm for finding the optimal solution efficiently.
- Very informally, similar solutions have similar costs.

Iterative Improvement

Intuition: consider the configurations to be laid out on the surface of a landscape. We want to find the highest point.

(Unlike other AI search problems like 8-puzzle, we don't care how we get there.)

"Iterative Improvement" methods:

```
Start at a random configuration;
repeatedly consider various moves;
accept some & reject some.
When you're stuck, restart.
```

We must invent a moveset that describes what moves we will consider from any configuration. Let's invent movesets for our four sample problems.

Hill-climbing

Hill-climbing: Attempt to maximize Eval(X) by moving to the highest configuration in our moveset. If they're all lower, we are stuck at a "local optimum."

1. Let X := initial config
2. Let E := Eval(X)
3. Let N = moveset_size(X)
4. For (i = 0 ; i < N ; i := i + 1)
   Let E_i := Eval(move(X,i))
5. If all E_i's are $\leq$ E, terminate, return X
6. Else let i* = argmax(E_i)
7. X := move(X,i*)
8. E := E_{i*}
9. Goto 3

(Not the most sophisticated algorithm in the world.)

Hill-climbing Issues

- Trivial to program
- Requires no memory (since no backtracking)
- MoveSet design is critical. This is the real ingenuity---not the decision to use hill-climbing.
- Evaluation function design often critical.
  - Problems: dense local optima or plateaux
- If the number of moves is enormous, the algorithm may be inefficient. What to do?
- If the number of moves is tiny, the algorithm can get stuck easily. What to do?
- It's often cheaper to evaluate an incremental change of a previously evaluated object than to evaluate from scratch. Does hill-climbing permit that?
- What if approximate evaluation is cheaper than accurate evaluation?
- Inner-loop optimization often possible.

Randomized Hill-climbing

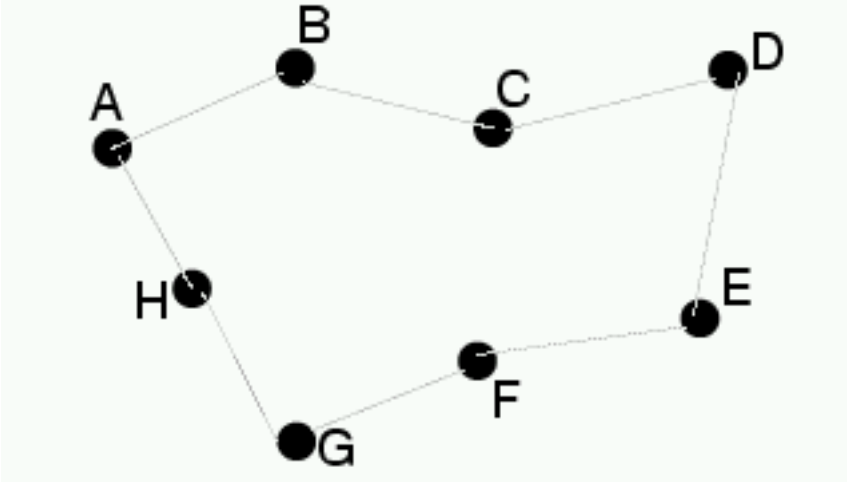
1. Let X := initial config
2. Let E := Eval(X)
3. Let i = random move from the moveset.
4. Let E_i := Eval(move(X,i))
5. If E < Ei then
   X := move(X,i)
   E := E_i
6. Goto 3 unless bored.

What stopping criterion should we use?

Any obvious pros or cons compared with our previous hill climber?

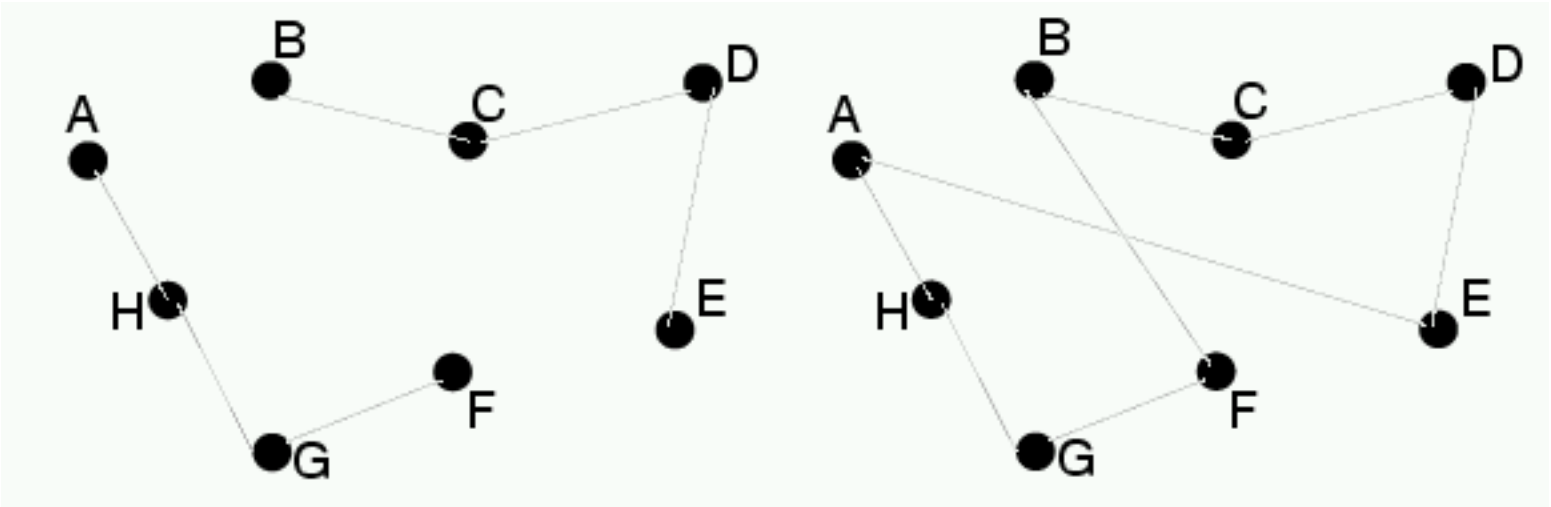
Hill-climbing Example: TSP

Minimize: Eval(Config) = length of tour



MoveSet: 2-change ... k-change

Example: 2-change



Simulated Annealing

1. Let X := initial object
2. Let E := Eval(X)
3. Let i = random move from the moveset.
4. Let E_i := Eval(move(X,i))
5. If E < E_i then

   X := move(X,i)

   E := E_i

   Else with some probability, accept the move even though things get worse:

   X := move(X,i)

   E := E_i
6. Goto 3 unless bored.

Simulated Annealing

1. Let X := initial object
2. Let E := Eval(X)
3. Let i = random move from the moveset.
4. Let E_i := Eval(move(X,i))
5. If E < E_i then

   X := move(X,i)

   E := E_i

   Else with some probability, accept the move even though things get worse:

   X := move(X,i)

   E := E_i
6. Goto 3 unless bored.

This may make the search fall out of mediocre local minima and into better local maxima.

How should we choose the probability of accepting a worsening move?

- *Idea One.* Probability = 0.1
- *Idea Two.* Probability decreases with time
- *Idea Three.* Probability decreases with time, and also as E - E_i increases.

Simulated Annealing

If $E_i >= E$ then definitely accept the change.

If $E_i < E$ then accept the change with probability

$$\exp(-(E - E_i)/T_i)$$

(called the Boltzman distribution)

..where T_i is a "temperature" parameter that gradually decreases. Typical cooling schedule: $T_i = T_0 \cdot r^i$

High temp: accept all moves (Random Walk)
Low temp: Stochastic Hill-climbing

When enough iterations have passed without improvement, terminate.

This idea was introduced by Metropolis in 1953. It is "based" on "similarities" and "analogies" with the way that alloys manage to find a nearly global minimum energy level when they are cooled slowly.

Simulated Annealing Issues

- MoveSet design is critical. This is the real ingenuity---not the decision to use simulated annealing.
- Evaluation function design often critical.
- Annealing schedule often critical.
- It's often cheaper to evaluate an incremental change of a previously evaluated object than to evaluate from scratch. Does simulated annealing permit that?
- What if approximate evaluation is cheaper than accurate evaluation?
- Inner-loop optimization often possible.

SA Discussion

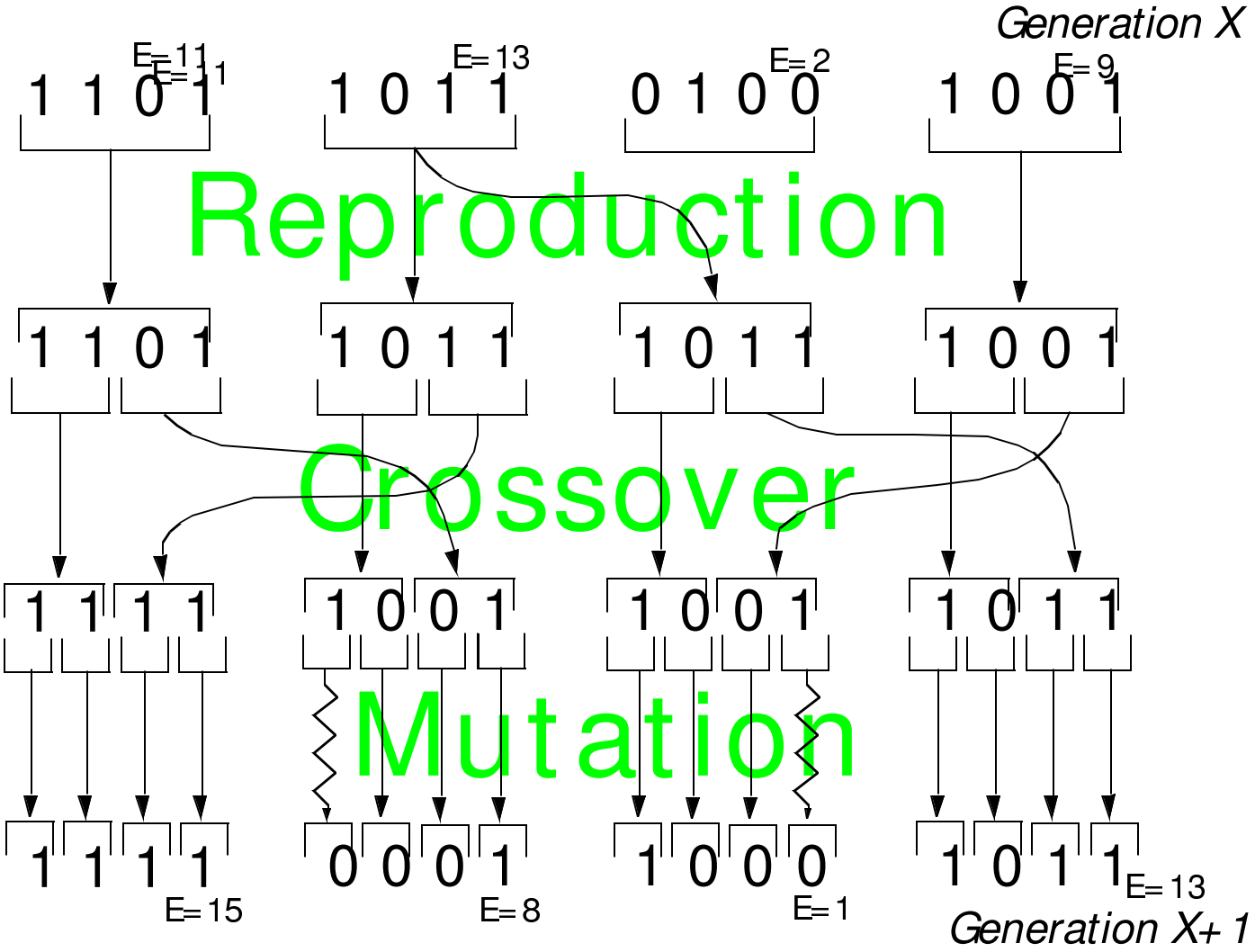
Simulated annealing is sometimes empirically much better at avoiding local minima than hill-climbing. It is a successful, frequently-used, algorithm. Worth putting in your algorithmic toolbox.

Sadly, not much opportunity to say anything formal about it (though there is a proof that with an infinitely slow cooling rate, you'll find the global optimum).

There are mountains of practical, and problem-specific, papers on improvements.

Genetic Algorithms

In the basic GA, objects are coded up (carefully) as binary strings. Goal is to optimize some function of the bit-strings.



(Diagram shamelessly copied from “Dean et al: AI: Theory and Practice”.)

Genetic Algorithm

A set of bitstrings. This set is called a Generation. The algorithm produces a new generation from an old generation thusly:

- Let G be the current generation of N bitstrings
- For each bitstring (call them $b_0, b_1, \ldots b_{N-1}$) define
  $p_i = Eval(b_i) / \Sigma_j Eval(b_j)$.
- Let G' be the next generation. Begin with it empty.
- For $k = 0 ; k < N/2 ; k = k+1$
  - Choose two parents each with probability
    $Prob(Parent = b_i) = p_i$
  - Randomly swap bits in the two parents to obtain two new bitstrings
  - For each bit in turn in the new bitstring, randomly invert it with some low probability
  - Add the two new bitstrings to G'

Let your first generation consist of random bitstrings.

GA Issues

- Bitstring representation is critical. This is the real ingenuity---not the decision to use genetic algorithms. *(How to encode TSP?)*
- Evaluation function design often critical.
- It's often cheaper to evaluate an incremental change of a previously evaluated object than to evaluate from scratch. Do Genetic Algorithms permit that?
- What if approximate evaluation is cheaper than accurate evaluation?
- Inner-loop optimization often possible.

Numerous twiddles:

- Use rankings not evaluations in creating your pi parent selection probabilities.
- Cross over contiguous chunks of the string instead of random bits?
- Needn't be bit strings...could use strings over other finite alphabets.
- Optimize over sentences from a grammar representing functions or programs. Called Genetic Programming.

General Discussion

- Often, the “second best way” to solve a problem.
- But relatively easy to implement. Can save a great deal of programming effort.
- But great care is needed in designing representations and movesets. If someone tells you that SA/Hillclimbing solved their problem, that person is probably not giving enough credit to their own problem-formulation-ability.
- DON’T solve a problem with these methods that could be solved by Linear Programming, A-Star search or Constraint Propagation!
- What if evaluating the objective function is really expensive?

What you should know about Iterative Improvement algs.

- Hill-climbing
- Simulated Annealing
- SAT and Channel Routing domains
- Given a simple problem (e.g. graph coloring from the CSP lectures) be able to give sensible suggestions as to how to code it up for the above algorithms.

References:

Simulated Annealing: See *Numerical Recipes in C,* or for practical details of Modified Lam schedule etc: Ochotta 1994 Ph.D. thesis, CMU ECE.

Hillclimbing: Discussion in Russell and Norvig.

GSAT, WALKSAT: papers by Bart Selman and Henry Kautz (www.research.att.com)

Channel Routing: Wong et.al., *Simulated Annealing for VLSI Design*, Kluwer 1988.