Heuristic Search

The Story So Far...

- The central problem of this course:

$$\arg\max_{X} \; Smartness(X)$$

  – Possibly with some constraints on X.

- (Alternatively: $\arg\min_{X} \; Stupidness(X)$)

Properties of *Smartness(X)*

- Possible categories for $\arg\max\limits_{X} Smartness(X)$, in decreasing order of desirability:

  - Efficient closed-form solution.

    - Linear Regression etc.

  - Differentiable and convex.

    - Quadratic programming (SVMs) etc.

  - Differentiable and non-convex.

    - Multi-layer Perceptrons etc.

  - Non differentiable

    - ??

Hill Climbing

- Assume that our search problem allows us to take one of a fixed number of moves.

- Each move transforms the current state to a successor state.

- States can be evaluated by an objective function.

- Hill Climbing Algorithm:

  - Start in an arbitrary state.

  - Choose the move that results in the best successor state.

  - Repeat until converged.

Properties of Hill Climbing

- Easy to program.

- Often finds good solutions quickly.

- Very susceptible to local maxima.

- Can be inefficient if many moves are available.

- Lots of variations:

  - Stochastic hill climbing – randomly choose an uphill move.

  - Random restart hill climbing – redo search until satisfied with result.

  - Simulated Annealing...

Simulated Annealing

- Avoiding local maxima requires us to take some steps downhill.

- Simulated Annealing attempts to find the right trade-off between uphill and downhill moves:

Start in a random state
    ** Select a random move
    * If the move results in improvement, keep it
    * If the move does *not* result in improvement,
       keep it with probability P^{SA}.
    * Return to **.

Computing P^{SA}

$$P^{SA} = e^{\dfrac{-(E_{cur} - E_{move})}{T_{cur}}}$$

- P^{SA} decreases as $(E_{cur} - E_{move})$ increases. I.e. The worse the move, the less likely we are to keep it.

- P^{SA} also decreases as T_{cur} (temperature) decreases.
  - High temperature: random search.
  - Low temperature: randomized hill climbing.

- If we decrease temperature *infinitely* slowly, we can guarantee that the global optimum will be found :)
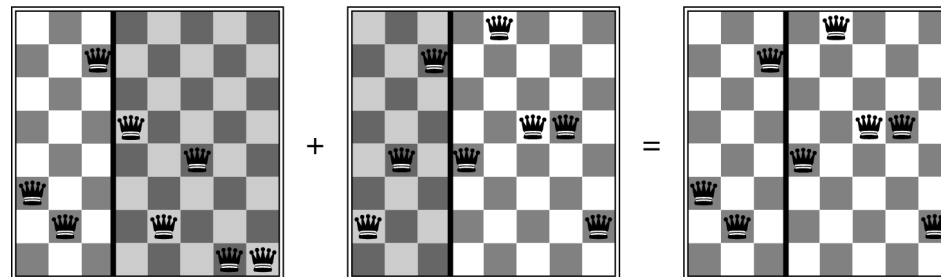
Genetic Algorithm Design

- (Basic algorithm due to John Holland, 1975)
- First, contrive a mapping from bit strings to your problem space.
    - Genotype -> Phenotype.
- Next, contrive an objective function for evaluating solutions:
    - fitness function.
    - F(phenotype) = fitness.

Genetic Algorithm

- Generate *K* bit strings randomly: a population of individuals.

- **Evaluate the fitness of each individual.

- Assign a probability to each individual proportional to it's fitness.

- Generate a new population:

  - Select 2 individuals (parents) according to probability assigned above.

  - Crossover: Pick a random bit position, and swap all bits after that position.

  - Mutation: flip individual bits with a small independent probability.

  - Repeat until we have K new individuals.

- Return to ** unless satisfied.

GA Illustration

24748552	24 31%	32752411	32748552	32748152
32752411	23 29%	24748552	24752411	24752411
24415124	20 26%	32752411	32752124	32252124
32543213	11 14%	24415124	24415411	24415417

| (a) | (b) | (c) | (d) | (e) |
| Initial Population | Fitness Function | Selection | Crossover | Mutation |

(Figure from Russel & Norvig, Artifical Intelligence a Modern Approach.)

Does it Work?

- How well it works has a lot to do with the structure of the fitness surface.

- Getting it to work well requires carefully engineering the genotype->phenotype mapping.

- MANY variations:

  - Different methods for selecting individuals, rank ordered instead of proportional, keep the fittest, etc. etc...

  - Co-evolving parasites – evolving test cases.

  - Independently evolving "island" populations.