Population Based Search

- Ant Colony Optimization
- Evolutionary Algorithms
  - Evolutionary Strategies
  - Genetic Algorithms
  - Genetic Programming

Ant Colony Optimization (ACO) [Dorigo 1992]
- constructive meta-heuristic that shares limited information between multiple search paths
- Inspiration: ants follow pheromone paths of other ants to find food. Over time, the best paths have strongest smell, luring more ants.
- Characteristics:
  - multi-agent/solution exploration (population based),
  - stochastic,
  - memory (individual and colony),
  - adaptive to problem,
  - implicit solution evaluation via “who gets there first”

Core Pseudo-Code for Each Ant

1. initialize
2. until reach solution
   1. read local routing table
   2. compute P(neighbors | routing-table, memory, constraints, heuristics)
   3. select move based on Ps and constraints
   4. update memory for move (pheromone globally and visited arc locally)
3. die

ACO Parameters

- when to terminate
- how many ants
- how to schedule ant activities (ant generation and optionals)
- what to keep in memory locally
- how to select a next state
- how to update memory (e.g., amount as function of solution quality)
- optionals:
  - delayed updating of pheromones on trail?
  - allow pheromone trail evaporation?
  - permit off-line pheromone updates (daemon actions)?
**Evolutionary Algorithms Overview**
- Evolutionary algorithms search a population representing different sample points in the search space.
- Each sample point is represented as a string which can be recombined with other strings to generate new sample points in the space.
- Algorithms are based on biological analogies with “population genetics” and “simulated evolution”.

**Why Evolutionary Algorithms?**
- **No Gradient Information Is Needed.** These algorithms do not search along the contours of the function, but rather by hyperplane sampling in Hamming space.
- **The Resulting Search is Global.** Since they do not hill-climb, they avoid local optima and so can be applied to multimodal functions.
- **Potential for Massive Parallelism.** Can effectively exploit thousands of processors in parallel.
- **They Can Be Hybridized** with conventional optimization methods.

**Issues: Representation**
- **Evolutionary model**
  - **Genotype:** represents information stored in chromosomes
  - **Phenotype:** describes how the individual appears
- **Approaches**
  - **Indirect:** standard data structure, genotype is mapped to phenotype
  - **Direct or natural:** specialized representation using domain specific search operators

**Issues: Representation**
- **Binary:** genotype is encoded as bit strings of length $l$
  - **Grey codes:** similar values have similar representations
  - **Binary numbers:** may be some issue of precision in encoded values
Issues: Representation

Nonbinary (integer, continuous, permutation):
larger alphabets, real-valued encodings, more natural

*Arguments against:*
- tends to have larger search space
- there will be fewer explicit hyperplane partitions
- the alphabetic characters will not be as well represented in a finite population.

Issues: Fitness Function

*Based on the objective function*
- Allows comparison of different solutions
- Domain specific to goals of problem
- Single value output: multi-objective must be combined into single function

Issues: Fitness Function

*Modifications*
- Must be fast! May need to be executed hundreds of thousands of times
- Sometimes approximate to achieve speed
- **Smoothing:** replacing fitness with average of neighbors (useful for plateaus)
- **Penalty functions:** to relax feasibility requirements when infeasible solutions cannot be removed

Issues: Initializing Population

Tension between good solutions and diversity
(low diversity can lead to quick stagnation or large distance from optimum)

*Random:* generate n strings uniform randomly, within encoding requirements.
*Domain specific:* use heuristic method to generate “ok” (greedy) solutions that can be refined.
Evolution Strategies
- Search method like local search but can be population based
- Vector of continuous variables
- Mutates each variable by incrementing value using randomly generated change with zero mean and set standard distribution… Mutation only
- Survival of fittest between new and previous vectors

(1+1)-Evolution Strategy Algorithm
1. Create initial solution \( x \)
2. while termination criterion is not met do
   1. for \( i = 1 \) to \( n \) do
      1. \( x'_i = x_i + \sigma N_i(0,1) \)
   2. If \( f(x') \geq f(x) \) then
      1. \( x = x' \)
   3. Update \( \sigma \)

\( N(0,1) \) indicates a normal distribution with 0 mean and 1 standard deviation.

(1+1)-ES Update
How to update \( \sigma \)?
- Convergence proofs on two simple problems define a 1/5 rule
  
  \[
  \frac{\text{# search steps that find a better solution}}{\text{all or last } \tau \text{ steps}}
  \]

  - If ratio is > 1/5, increase \( \sigma \); else decrease
  - Keep it fixed, which is better for escaping local optima

(\( \mu+\lambda \))-ES
- \( \mu \) solutions in population
- Generate \( \lambda \) new solutions
- Choose the best \( \mu \) from superset at each iteration
- Can add recombination before mutation
- Can have different \( \sigma \) per variable
Evolution Strategies (cont.)

Self adaptive mutation and rotation
- Log-normal distribution for mutation
- Adaptive through strategy ($\sigma$) and rotation angle ($\alpha$) parameters added to chromosome

$$\{x_1, x_2, \sigma_1, \sigma_2, \alpha_1, \alpha_2\}$$

Simple Mutations  Correlated Mutations via Rotation

Figures courtesy of D. Whitley

Simple Genetic Algorithm (Alg 7 in Rothlauf)

1. Create initial population $P$ with $N$ solutions $x^i \in \{1, \ldots, N\}$
2. for $i$ from 1 to $N$ do
3.   Calculate $f(x^i)$
4. while (termination criterion is not met) and (population has not yet converged) do
5.   $\text{ind} = 1$
6.   repeat
7.     if random(0,1) $\leq p_c$ then recombine $x^{m} \in M$ and $x^{n+1} \in M$ and place the offspring in $P'$
8.       Else copy $x^{m} \in M$ and $x^{n+1} \in M$ to $P'$
9.     $\text{ind} = \text{ind} + 2$
10. until $\text{ind} > N$
11. for $i$ from 1 to $N$ do
12.     for $j$ from 1 to $l$ do
13.       if random(0,1) $\leq p_m$ then mutate($x^i$) where $x^i \in P'$
14.       Calculate $f(x^i)$ where $x^i \in P'$
15.     $P = P'$

Genetic Algorithm Process

Selection (Duplication)  Recombination (Crossover)

Proportionate Selection

- Population is evaluated according to a fitness function.
- Parents are selected for reproduction by ranking according to their relative fitness

$$\frac{f_i}{\sum_{j=1}^{N} f_j}$$
Proportionate Selection Process

- **Stochastic sampling with replacement**
  - Map individuals to space on a roulette wheel, more fit individuals are allocated proportionally more space.
  - Spin wheel repeatedly until desired population size is achieved

Population Example, 
**Stochastic Sampling w/Replacement**

<table>
<thead>
<tr>
<th>String</th>
<th>Fit</th>
<th>Space</th>
<th>copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>001000000</td>
<td>2.0</td>
<td>.995</td>
<td></td>
</tr>
<tr>
<td>101010101</td>
<td>1.9</td>
<td>.186</td>
<td></td>
</tr>
<tr>
<td>111100011</td>
<td>1.8</td>
<td>.271</td>
<td></td>
</tr>
<tr>
<td>010001100</td>
<td>1.7</td>
<td>.352</td>
<td></td>
</tr>
<tr>
<td>111000000</td>
<td>1.6</td>
<td>.429</td>
<td></td>
</tr>
<tr>
<td>101001110</td>
<td>1.5</td>
<td>.5</td>
<td></td>
</tr>
<tr>
<td>011001110</td>
<td>1.4</td>
<td>.567</td>
<td></td>
</tr>
<tr>
<td>001111000</td>
<td>1.3</td>
<td>.629</td>
<td></td>
</tr>
<tr>
<td>000101000</td>
<td>1.2</td>
<td>.686</td>
<td></td>
</tr>
<tr>
<td>100000001</td>
<td>1.1</td>
<td>.738</td>
<td></td>
</tr>
<tr>
<td>010000111</td>
<td>1.0</td>
<td>.786</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>Fit</th>
<th>Space</th>
<th>copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>011001111</td>
<td>0.9</td>
<td>.829</td>
<td></td>
</tr>
<tr>
<td>000100110</td>
<td>0.8</td>
<td>.867</td>
<td></td>
</tr>
<tr>
<td>110001101</td>
<td>0.7</td>
<td>.9</td>
<td></td>
</tr>
<tr>
<td>110000111</td>
<td>0.6</td>
<td>.929</td>
<td></td>
</tr>
<tr>
<td>101000100</td>
<td>0.5</td>
<td>.952</td>
<td></td>
</tr>
<tr>
<td>011011011</td>
<td>0.4</td>
<td>.971</td>
<td></td>
</tr>
<tr>
<td>000111000</td>
<td>0.3</td>
<td>.986</td>
<td></td>
</tr>
<tr>
<td>001100100</td>
<td>0.2</td>
<td>.995</td>
<td></td>
</tr>
<tr>
<td>100111010</td>
<td>0.1</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>010010011</td>
<td>0.0</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>

Random #s: .93, .65, .02, .51, .20, .93, .20, .37, .79, .28, .13, .70, .80, .51, .76, .45, .61, .07, .76, .86, .29

Other Selection Methods

- **Tournament Selection**: randomly select two strings, place the best into the new population, repeat until intermediate population is full
- **Ranking**: order individuals by rank rather than fitness value

Another Fitness Selection Method

- **Remainder stochastic sampling**
  - Again map to roulette wheel, but this time add outer wheel with N evenly spaced pointers.
  - Spin once to determine all population members
Population Example,  
Remainder Stochastic Sampling

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>Random copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>001000000</td>
<td>2.0</td>
<td>2</td>
</tr>
<tr>
<td>101010101</td>
<td>1.9</td>
<td>2</td>
</tr>
<tr>
<td>111100001</td>
<td>1.8</td>
<td>2</td>
</tr>
<tr>
<td>010001000</td>
<td>1.7</td>
<td>2</td>
</tr>
<tr>
<td>111100000</td>
<td>1.6</td>
<td>2</td>
</tr>
<tr>
<td>101000101</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>010001101</td>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>011110000</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>001101000</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>100000111</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>010000111</td>
<td>1.0</td>
<td>2</td>
</tr>
</tbody>
</table>

Random #s: .93, .55, .50, .20, .93, .20, .37, .79, .28, .13, .70, .80, .51, .76, .45, .61, .07, .76, .86, .29

Reproduction: Recombination/ Crossover

Two parents: binary strings representing an encoding of 5 parameters that are used in some optimization problems.

```
10010101 11110101 01110100 10100101 10000101
xwwwxyx wwwxyxww xwwwxyww ywwwxyxx xwwwxyxx
```

Recombination occurs as follows:

```
10010 / 111101011001101101001100 / 001000101011000101
xxyyy / yyyyyyyyyyyyyyyyyy / xyyyyyyyyyyyyyyyyyy
```

Producing the following offspring:

```
10010yxxxywwwxxyxxwyyw001010101111000101
```

Genetic Algorithm Process

Selection (Duplication)  
Recombination (Crossover)

Desirable Characteristics of Recombination Operators

- **Respect**: any commonalities of the parents are inherited by the offspring
- **Transmission**: all components of the offspring must have come from a parent
- **Assortment**: all components of the offspring must be compatible/feasible
- **Ergodicity**: can reach any combination from all possible starting points
### Issues: Combination Operators

1 **point crossover**: pick single crossover point, split strings at this point, recombine

2 **point crossover**: pick two crossover points, split strings at these points, recombine (think of ring for string)

**Uniform crossover**: randomly pick each element from one of the two parents

### Crossover and Hypercube Paths

![Graph of hypercube paths](image)

### Other Combination Operators

**HUX**: exactly half of the differing bits are swapped
Given parents:
- 101101111000010110
- 1011001110000110001

A new individual is:
- 10010011110000110010

### More Combination Operators

**Reduced Surrogate**: Crossover points chosen within differing bits only
- 1001101111000010110
- 1011001110001100001

May become:
- 100110111100001100001

**Domain specific**: operations designed to match demands of the domain (e.g., reorder portions for scheduling application)
Why Might Reduced Surrogate Be Important

- Closely related to HUX
- Key idea: look only at portions of strings that differ
  001111011010011
  0010011010010010
- How does probability of new string change with reduced surrogate versus 1-point crossover?

Mutation

- For each bit in population, mutate with probability $p_m$
  - $p_m$ should be low, typically < .01
  - can mean randomly select a value for the bit or flip the bit

Issues: Which Strings in New Generation

- Replace with offspring
  - Assumption of canonical GA
  - Best of offspring and parents
  - Alternative view which guarantees always keep best and puts intensive pressure on population for improvement

Issues: Termination Criteria

- Quality of solution: best individual passes some pre-set threshold
- Time: certain number of generations have been created and tested
- Diminishing returns: improvement over each generation does not exceed some threshold
Alternatives to the Simple GA Model

- Genitor
- CHC

Genitor (Whitley et al.)

Differences with canonical genetic algorithm:
- Reproduction proceeds one individual at a time.
- Worst individual in population is replaced by new offspring.
- Fitness is assigned by rank.
- Steady State GA

Genitor Algorithm (Whitley)

CHC (Eshelman)

- Crossover using generation elitist selection
- Heterogeneous recombination by incest prevention
  - Origin of HUX terminology
- Cataclysmic mutation, when population starts to converge
**CHC Process**

Parent1 → Parent2 → Parent3 → Parent4 → Parent n → Child1 → Child2 → Child3 → Child4 → Child n

- *via Random Selection and HUX*
- *Best n strings from parents and children*

**Genetic Algorithms for Scheduling**

- Represent solution as permutation of tasks
- Requires a schedule builder to convert solution into schedule and assess objective function. [Indirect search]
- Syswerda’s permutation crossover was used for scheduling.

**GA applied to TSP**

- Applet showing simple GA for TSP

**Genetic Programming**

- Combine and/or parameterize code blocks (functions and terminals) to produce program to solve some problem.
- Follows GA functionality
- Solution is often a parse tree of functions and terminals.
  - Interior nodes are functions (e.g., “+”, “or”), control structures (e.g., “if”, “while”) or functions with side effects (e.g., “read”, “print”).
  - Terminals are variables and constants.
GP Steps
1. Initialize population of programs
2. Loop
   1. Assess fitness
   2. Construct new generation through selection and
      - Reproduction,
      - Mutation or
      - Crossover

GP initialization
- Grow: start with empty tree and iteratively assign nodes to be function or terminal. All nodes at depth $k_{\text{max}}$ are terminals.
- Full: start with empty tree, randomly add functions up to depth $k_{\text{max}-1}$, terminals at depth $k_{\text{max}}$.
- Ramped-half-and-half: divide population into $(k_{\text{max}}-1)$ parts, half of each is created by grow and other by full, depth of nodes in ith part is varied from 2 to $k_{\text{max}}$

GP: Assessing Fitness
- Use program to solve problem
  - Quality of solution
  - Efficiency of solution
- May require simulation, e.g., UAV

GP: Representation
Domian Dependent!
- E.g., Simple Lisp functions (math, cons, list, append...)

```
+ 3 c a
  * a b
```
GP: Mutation
- Substitute one function (terminal) for another
- Substitute one subtree for another


GP: Crossover


GP Representation for UAVs I

Project of Whitley, Beveridge, Richards, Mytkowicz

GP Representation for UAVs II

```latex
{add
{if-in-turning-radius
{div2(ifgteq {closest-friend} {sweep-east})
 {sweep-south}{sweep-west})}
{unit {closest-beacon})
{if-in-turning-radius {closest-beacon})
{closest-friend} {closest-beacon})}
{if-in-turning-radius
{ifdot {sweep-south}
{if-in-turning-radius {sweep-east})
{mul2 {if-in-turning-radius {sweep-east}
{closest-beacon} {last})}
{closest-beacon})
{add {left-friend}) {sweep-west})
{neg {left-friend})}
{div2 {neg {closest-friend})
{last})}
```
UAV Movies

Another View of Hyperplane Sampling

Underlying Theory: Hyperplane Sampling

Population Example for Hyperplane Sampling

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>Random</th>
<th>copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>001100_00</td>
<td>2.0</td>
<td>--</td>
<td>2</td>
</tr>
<tr>
<td>101100_00</td>
<td>1.9</td>
<td>0.03</td>
<td>2</td>
</tr>
<tr>
<td>111100_00</td>
<td>1.8</td>
<td>0.65</td>
<td>2</td>
</tr>
<tr>
<td>010100_00</td>
<td>1.7</td>
<td>0.02</td>
<td>1</td>
</tr>
<tr>
<td>110100_00</td>
<td>1.6</td>
<td>0.51</td>
<td>2</td>
</tr>
<tr>
<td>101100_00</td>
<td>1.5</td>
<td>0.20</td>
<td>1</td>
</tr>
<tr>
<td>011100_00</td>
<td>1.4</td>
<td>0.93</td>
<td>2</td>
</tr>
<tr>
<td>001100_00</td>
<td>1.3</td>
<td>0.20</td>
<td>1</td>
</tr>
<tr>
<td>100100_00</td>
<td>1.2</td>
<td>0.37</td>
<td>1</td>
</tr>
<tr>
<td>100100_00</td>
<td>1.1</td>
<td>0.79</td>
<td>1</td>
</tr>
<tr>
<td>010100_00</td>
<td>1.0</td>
<td>--</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>Random</th>
<th>copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>001100_00</td>
<td>0.9</td>
<td>0.28</td>
<td>1</td>
</tr>
<tr>
<td>101100_00</td>
<td>0.8</td>
<td>0.13</td>
<td>0</td>
</tr>
<tr>
<td>111100_00</td>
<td>0.7</td>
<td>0.70</td>
<td>1</td>
</tr>
<tr>
<td>010100_00</td>
<td>0.6</td>
<td>0.80</td>
<td>1</td>
</tr>
<tr>
<td>110100_00</td>
<td>0.5</td>
<td>0.51</td>
<td>1</td>
</tr>
<tr>
<td>101100_00</td>
<td>0.4</td>
<td>0.76</td>
<td>1</td>
</tr>
<tr>
<td>011100_00</td>
<td>0.3</td>
<td>0.45</td>
<td>0</td>
</tr>
<tr>
<td>001100_00</td>
<td>0.2</td>
<td>0.61</td>
<td>0</td>
</tr>
<tr>
<td>100100_00</td>
<td>0.1</td>
<td>0.07</td>
<td>0</td>
</tr>
<tr>
<td>010100_00</td>
<td>0.0</td>
<td>--</td>
<td>0</td>
</tr>
</tbody>
</table>
Some Schemata and Fitness Values

<table>
<thead>
<tr>
<th>Schema</th>
<th>Mean</th>
<th>Count</th>
<th>Expect</th>
<th>Observe</th>
</tr>
</thead>
<tbody>
<tr>
<td>10**...*</td>
<td>1.70</td>
<td>2</td>
<td>3.4</td>
<td>3</td>
</tr>
<tr>
<td>11**...*</td>
<td>1.70</td>
<td>2</td>
<td>3.4</td>
<td>4</td>
</tr>
<tr>
<td><strong>0</strong>...*</td>
<td>1.70</td>
<td>4</td>
<td>6.8</td>
<td>7</td>
</tr>
<tr>
<td><strong>01</strong>...*</td>
<td>1.38</td>
<td>5</td>
<td>6.9</td>
<td>6</td>
</tr>
<tr>
<td><strong>01</strong>...*</td>
<td>1.30</td>
<td>10</td>
<td>13.0</td>
<td>14</td>
</tr>
<tr>
<td><strong>1</strong>...*</td>
<td>1.22</td>
<td>5</td>
<td>6.1</td>
<td>8</td>
</tr>
<tr>
<td>11**...*</td>
<td>1.175</td>
<td>4</td>
<td>4.7</td>
<td>6</td>
</tr>
<tr>
<td>00**...*</td>
<td>1.166</td>
<td>3</td>
<td>3.5</td>
<td>3</td>
</tr>
<tr>
<td>0<em>0**...</em></td>
<td>1.089</td>
<td>9</td>
<td>9.8</td>
<td>11</td>
</tr>
<tr>
<td>0<em>01**...</em></td>
<td>1.033</td>
<td>6</td>
<td>6.2</td>
<td>7</td>
</tr>
<tr>
<td>1<em>0**...</em></td>
<td>1.020</td>
<td>5</td>
<td>5.1</td>
<td>5</td>
</tr>
<tr>
<td><strong>0</strong>...*</td>
<td>1.010</td>
<td>10</td>
<td>10.1</td>
<td>12</td>
</tr>
<tr>
<td><strong>0</strong>...*</td>
<td>1.000</td>
<td>21</td>
<td>21.0</td>
<td>21</td>
</tr>
</tbody>
</table>

Hyperplane Deception

Since genetic algorithms are driven by hyperplane sampling a misleading problem can be constructed as follows.

\[ f(0** > f(1**) \]
\[ f(*0*) > f(*1*) \]
\[ f(**0) > f(**1) \]
\[ f(00*) > f(01*), f(10*), f(11*) \]
\[ f(0*0) > f(0*1), f(1*0), f(1*1) \]
\[ f(*00) > f(*01), f(*10), f(*11) \]

BUT \[ f(111) > f(000) \]

where \( f(x) \) gives the average fitness of all strings in the hyperplane slice represented by \( x \).

Fitness Landscape Analysis

- A fitness landscape \( (X,f,d) \) of a problem instance consists of a set of solutions \( X \), an objective function \( f \) and a distance measure \( d \).

- We can analyze problems based on characteristics derived from this definition and determine (roughly) problem difficulty.

Locality

- How well distances correspond to differences in fitness between solutions
- High locality if nearby solutions have similar fitness
- Low locality problems are difficult for local search
Fitness-Distance Correlation

- Metric of locality
  \[ \rho_{FDC} = \frac{c_{fg}}{\sigma(f)\sigma(d_{fg})} \]
  where
  \[ c_{fg} = \frac{1}{m} \sum_{i=1}^{m} (f_i - \langle f \rangle)(d_{i,fg} - \langle d \rangle) \]

- \( c \) is the covariance, \(< >\) are means.
- Positive correlation is easy, uncorrelated is difficult, negative correlation is misleading.

Ruggedness

- Statistical property on relation of objective values to intra-solution distances
- Random walk correlation between solutions separated by \( s \) steps
  \[ r(s) = \frac{\langle f(x_i)f(x_{i+s}) \rangle - \langle f \rangle^2}{\langle f^2 \rangle - \langle f \rangle^2} \]

- \( x_i \) is the solution at step \( i \)
- Correlation length…high means smooth, easy for guided search
  \[ l_{cor} = -\frac{1}{\ln\left|f(1)\right|} \]

Big Valley Topology

- Original observation from (Boese, Kahng & Muddu 1994)
- Figure: benchmark FlowShop scheduling problem from Watson et al. 1999

No Free Lunch…

- Wolpert and Macready, 1997
  “For any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.”
- Proven for finite optimization problems solved with deterministic “black-box” algorithms
NFL Theorem I

For any pair of algorithms $a_1$ and $a_2$:

$$\sum_f P(d^y_m \mid f, m, a_1) = \sum_f P(d^y_m \mid f, m, a_2)$$

$f$ is an optimization problem, $m$ is number of distinct points sampled by the algorithm, $y$ is the cost value, and $d$ is the best cost value over the sample.

Meaning… average performance for an algorithm over all problems is the same as for any other algorithm, assuming equal sampling size.

Implications of NFL

- Emphasizes need for different algorithms
- Does not preclude superior performance on some interesting class(es) of problem!

$$P(d^y_m \mid m, a) = \sum_f P(d^y_m \mid f, m, a) P(f)$$

- However, if no problem knowledge is used, then $P(f)$ is essentially uniform.

Implications (cont.)

- Structure of problem must be understood and used to inform choice of search.
- Knowledge of cost function properties is folded into $P(i)$ or $\bar{P}$

$$P(d^y_m \mid m, a) = \tilde{V}_{d^y_m, m, a} \cdot \bar{P}$$

- “Performance of an algorithm is determined by … how aligned $\tilde{V}_{d^y_m, m, a}$ is with the problems $\bar{P}$”

Comparing Performance

- Assume we can construct a histogram of cost values produced by a run of an algorithm $\tilde{c}$, then use these to gauge performance:

  - Over all cost functions:
    - Average of $P(\min(\tilde{c}) > \varepsilon \mid f, m, a)$
    - $P(\min(\tilde{c}) > \varepsilon \mid f, m, a)$ for the random algorithm
  - Given $f$ and $m$, % of algorithms with $\min(\tilde{c} > \varepsilon)$
Minimax & Fixed f

- NFL focuses on average performance.
  - One algorithm may be much better than another on a subset of f (the other is a little better on remaining to even out average). [Minimax]
  - May occur when samples intersect
- If f is fixed, observing performance until time t does not indicate what will happen later.
  - Intelligent choice requires knowledge of both f and a

Phase Transitions

Cheeseman, Kanefsky & Taylor (1991) observed that:

- The hardest problems in an NP-complete class (e.g., TSP, SAT) were at the boundary between feasibility and infeasibility.
- Problems that are clearly feasible (or not) are easy to solve (or easy to recognize there’s no solution).
- “Phase transition” depends on identifying a key scale up parameter (e.g., constraints per variable, standard deviation of costs)