LECTURE 12: Randomized Feature Selection

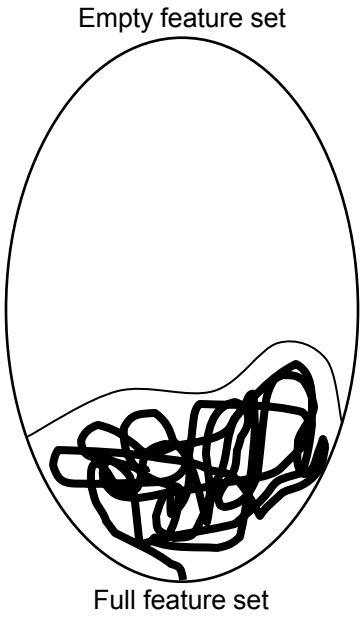
- **Exponential search methods**
  - Branch and Bound
  - Approximate Monotonicity with Branch and Bound
  - Beam Search
- **Randomized algorithms**
  - Random Generation plus Sequential Selection
  - Simulated Annealing
  - Genetic Algorithms

Branch and Bound (B&B) (1)

- **The Branch and Bound algorithm [Narendra and Fukunaga, 1977] is guaranteed to find the optimal feature subset under the monotonicity assumption**
  - The monotonicity assumption states that the addition of features can only increase the value of the objective function, this is

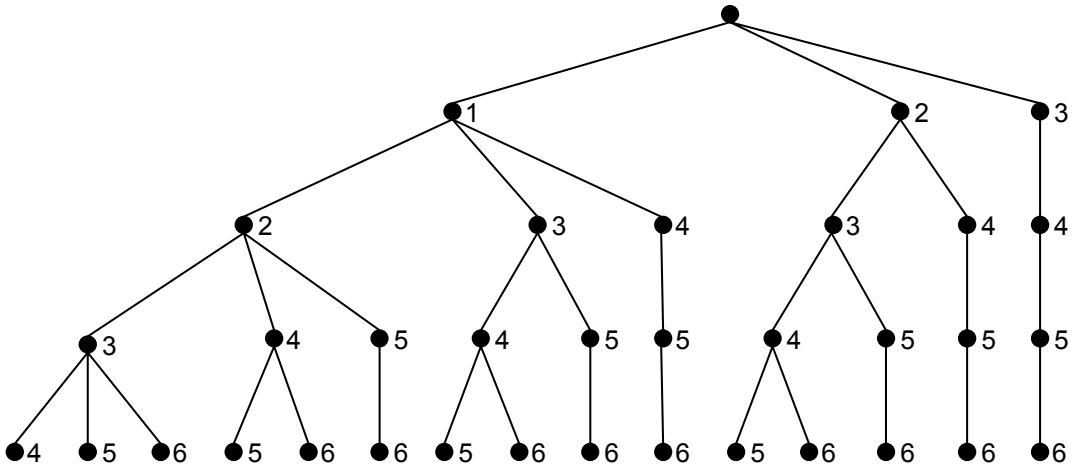
$$J(x_{i_1}) < J(x_{i_1}, x_{i_2}) < J(x_{i_1}, x_{i_2}, x_{i_3}) < \cdots < J(x_{i_1}, x_{i_2}, \cdots, x_{i_N})$$

  - Branch and Bound starts from the full set and removes features using a depth-first strategy
    - Nodes whose objective function are lower than the current best are not explored since the monotonicity assumption ensures that their children will not contain a better solution



Branch and Bound (2)

- **Algorithm [Fukunaga, 1990]**
  - The algorithm is better explained by considering the subsets of M'=N-M features already discarded, where N is the dimensionality of the state space and M is the desired number of features
  - Since the order of the features is irrelevant, we will only consider an increasing ordering $i_1<i_2<...i_{M'}$ of the feature indices, this will avoid exploring states that differ only in the ordering of their features
  - The Branch and Bound tree for N=6 and M=2 is shown below (numbers indicate features that are being removed)
    - Notice that at the level directly below the root we only consider removing features 1, 2 or 3, since a higher number would not allow sequences ($i_1< i_2< i_3< i_4$) with four indices

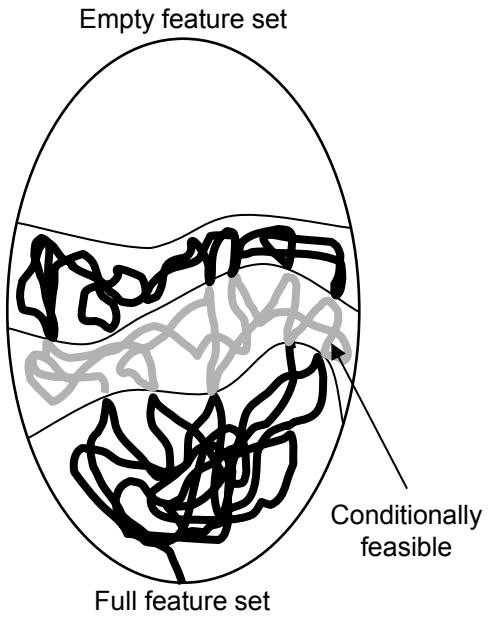


Branch and Bound (3)

1. Initialize: $\alpha=-\infty$, k=1
2. Generate successors of the current node and store them in LIST(k)
3. Select new node
   - if LIST(k) is empty
     - go to Step 5
   - else
     - $i_k = \arg\max_{j \in \text{LIST}(k)} \left[J(x_{i_1}, x_{i_2}, \ldots x_{i_{k-1}}, j) \right]$
     - delete i_k from LIST(k)
4. Check bound
   - if $J(x_{i_1}, x_{i_2}, \ldots x_{i_k}) < \alpha$
     - go to 5
   - else if k=M' (we have the desired number of features)
     - go to 6
   - else
     - k=k+1
     - go to 2
5. Backtrack to lower level
   - set k=k-1
   - if k=0
     - terminate algorithm
   - else
     - go to 3
6. Last level
   - Set $\alpha = J(x_{i_1}, x_{i_2}, \ldots x_{i_{k-1}}, j)$ and $Y^*_{M'} = \{x_{i_1}, x_{i_2}, \ldots x_{i_k}\}$
   - go to 5

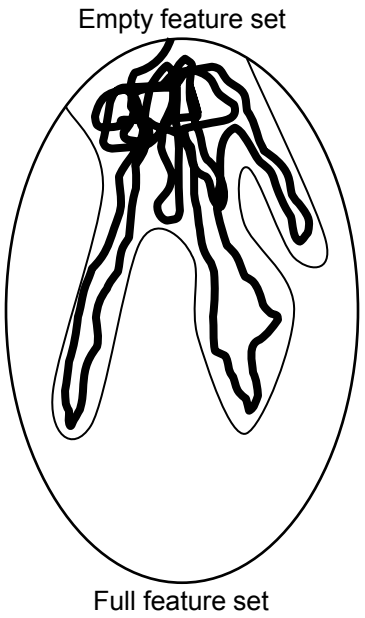
Approximate Monotonicity with B & B (AMB&B)

- **AMB&B is a variation of the classical Branch and Bound algorithm**
  - AMB&B allows non-monotonic functions to be used, typically classifiers, by relaxing the cutoff condition that terminates the search on a specific node
- **Assume that we run B&B by setting a threshold error rate $E(Y)=\tau$ rather than a number of features M**
  - Under AMB&B, a given feature subset Y will be considered
    - **Feasible** if $E(Y)\leq\tau$
    - **Conditionally feasible** if $E(Y)\leq\tau(1+\Delta)$
    - **Unfeasible** if $E(Y)\geq\tau(1+\Delta)$
  - Δ is a tolerance placed on the threshold to accommodate for non-monotonic functions
  - Rather than limiting the search to feasible nodes (like B&B does), AMB&B allows the search to explore conditionally feasible nodes with the hope that these nodes will lead to a feasible solution
    - However, AMB&B will not return conditionally feasible nodes as solutions, it only allows the search to explore them!
    - Otherwise it would not be any different than B&B with a higher threshold of $\tau(1+\Delta)$



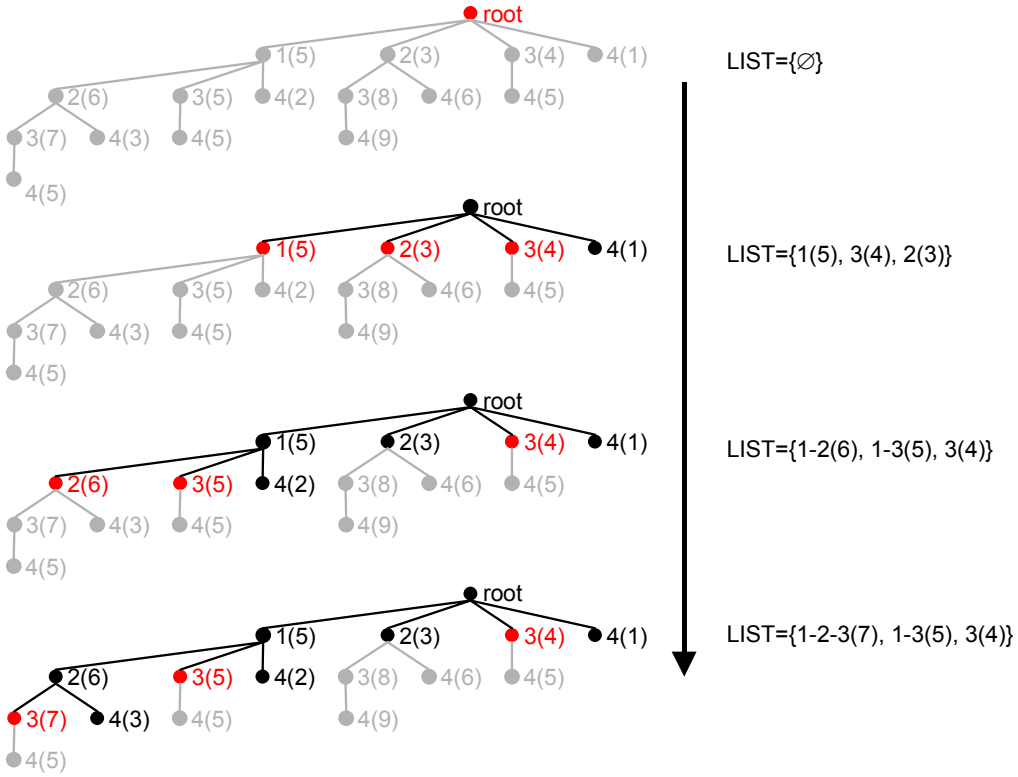
Beam Search (1)

- **Beam Search is a variation of best-first search with a bounded queue to limit the scope of the search**
  - The queue organizes states from best to worst, with the best states placed at the head of the queue
  - At every iteration, BS evaluates all possible states that result from adding a feature to the feature subset, and the results are inserted into the queue in their proper locations
  - Notice that BS degenerates to Exhaustive search if there is no limit on the size of the queue. Similarly, if the queue size is set to one, BS is equivalent to Sequential Forward Selection



Beam Search (2)

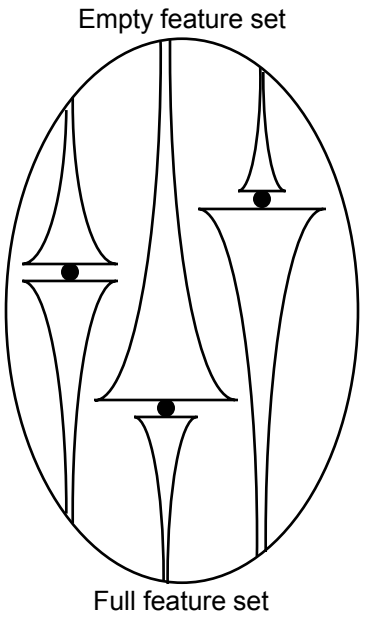
- **The example below illustrates BS for a 4-dimensional search space and a queue of size 3**
  - BS cannot guarantee that the optimal subset is found:
    - In the example, the optimal is 2-3-4 (J=9), which is never explored
    - However, with the proper queue size, Beam Search can avoid getting trapped in local minimal by preserving solutions from varying regions in the search space



Random Generation plus Sequential Selection

- **RGSS is an attempt to introduce randomness into SFS and SBS in order to escape local minima**
- **The algorithm is self-explanatory**

1. Repeat for a number of iterations
   1a. Generate a random feature subset
   1b. Perform SFS on this subset
   1c. Perform SBS on this subset



Simulated Annealing (1)

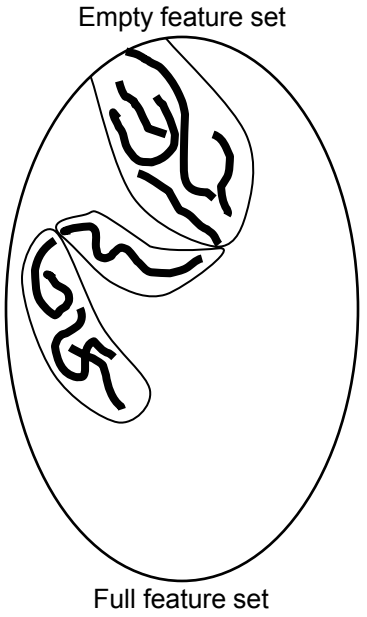
- **Simulated Annealing is a stochastic optimization method that derives its name from the annealing process used to re-crystallize metals**
  - During the annealing process in metals, the alloy is cooled down slowly to allow its atoms to reach a configuration of minimum energy (a perfectly regular crystal)
    - If the alloy is annealed too fast, such an organization cannot propagate throughout the material. The result will be a material with regions of regular structure separated by boundaries. These boundaries are potential fault-lines where fractures are most likely to occur when the material is stressed
  - The laws of thermodynamics state that, at temperature T, the probability of an increase in energy ΔE in the system is given by the expression

$$P(\Delta E) = e^{-\frac{\Delta E}{kT}}$$

    - where k is known as Boltzmann's constant

- **The SA algorithm is a straightforward implementation of these ideas**

1. Determine an annealing schedule T(i)
2. Create an initial solution Y(0)
3. While $T(i) > T_{MIN}$
   3a. Generate a new solution Y(i+1) which is a neighbor of Y(i)
   3b. Compute $\Delta E = [\, J(Y(i)) - J(Y(i+1)) \,]$
   3b. If $\Delta E < 0$
       then
           always accept the move from Y(i) to Y(i+1)
       else
           accept the move with probability $P = \exp(-\Delta E / T(i))$

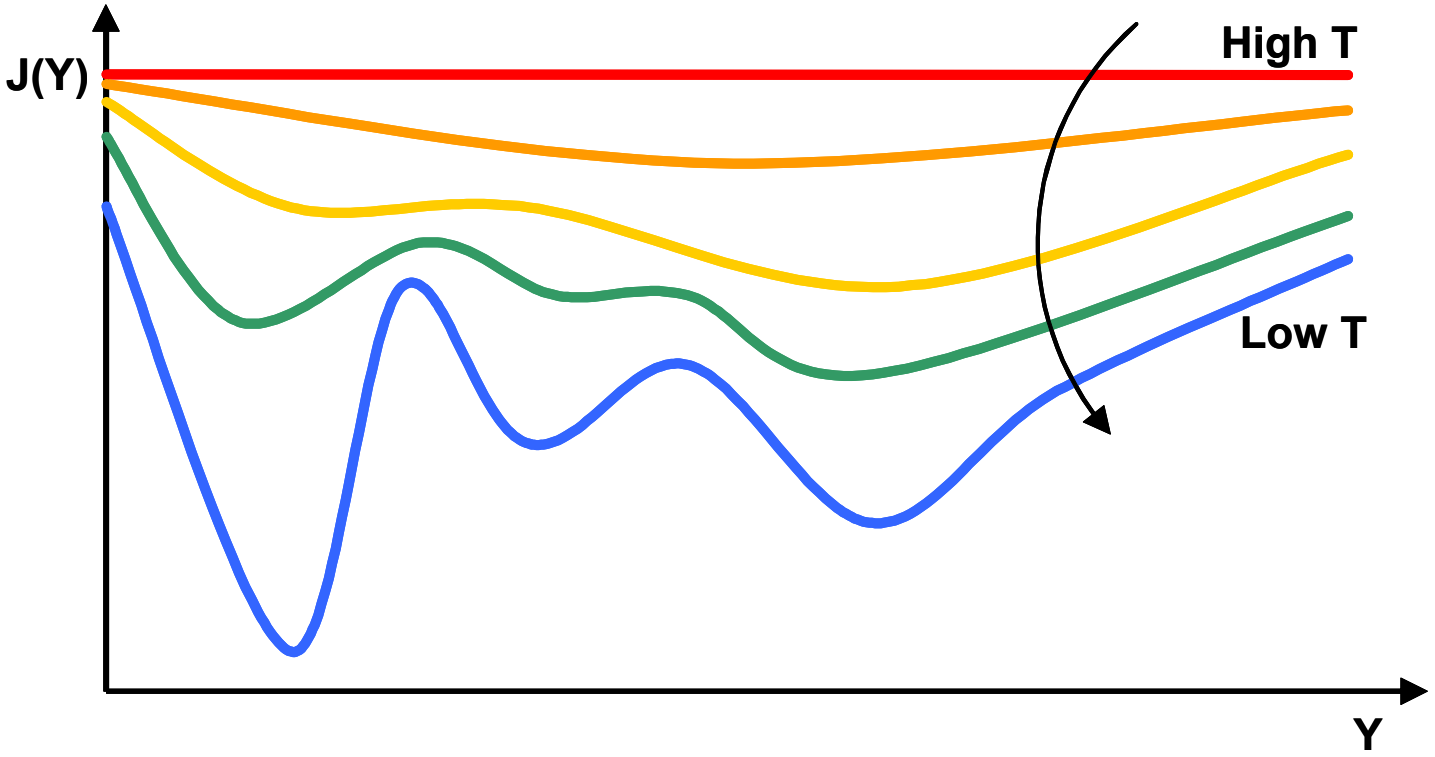


Simulated Annealing (2)

- **Simulated annealing is summarized with the following idea**
  - "When optimizing a very large and complex system (i.e., a system with many degrees of freedom) , instead of always going downhill, try to go downhill most of the times" [Haykin, 1999]
- **The previous formulation of the Simulated Annealing algorithm can be used for any type of minimization problem and it only requires specification of**
  - A transform to generate a local neighbor from the current solution (i.e. add a random vector)
    - For Feature Subset Selection, the transform will consist of adding or removing features, typically implemented as a random mutation with low probability
  - An annealing schedule, typically $T(i+1)=r\times T(i)$, with $0.0 \leq r \leq 1.0$
  - An initial temperature T(0)
- **Selection of the annealing schedule is critical**
  - If 'r' is too large, the temperature decreases very slowly, allowing moves to higher energy states to occur more frequently. This results in slow convergence
  - If 'r' is too small, the temperature decreases very fast, and the algorithm is likely to converge to a local minima

Simulated Annealing (3)

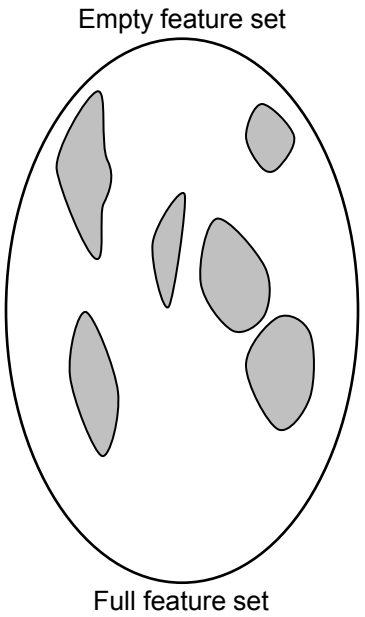
- **A unique feature of simulated annealing is its adaptive nature**
  - At high temperature the algorithm is only looking at the gross features of the optimization surface, while at low temperatures, the finer details of the surface start to appear



Genetic Algorithms

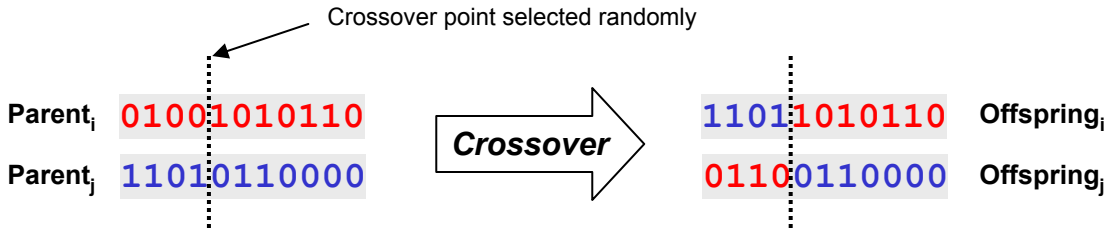
- **Genetic algorithms are optimization techniques that mimic the evolutionary process of "survival of the fittest"**
  - Starting with an initial random population of solutions, evolve new populations by mating (crossover) pairs of solutions and mutating solutions according to their fitness (objective function)
  - The better solutions are more likely to be selected for the mating and mutation operations and therefore carry their "genetic code" from generation to generation
  - For the problem of Feature Subset Selection, individual solutions are simply represented with a binary number (1 if the given feature is selected, 0 otherwise), which is the original representation proposed by Holland in 1974
- **Algorithm**

1. Create an initial random population
2. Evaluate initial population
2. Repeat until convergence (or a number of generations)
   2a. Select the fittest individuals in the population
   2b. Perform crossover on the selected individuals to create offspring
   2c. Perform mutation on the selected individuals
   2d. Create the new population from the old population and the offspring
   2e. Evaluate the new population

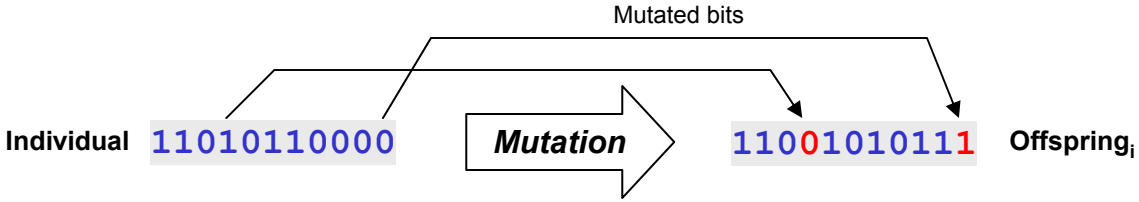


Genetic operators

- **Single-point crossover**
  - Select two individuals (parents) according to their fitness
  - Select a crossover point
  - With probability P_c (0.95 is reasonable) create two offspring by combining the parents



- **Binary mutation**
  - Select an individual according to its fitness
  - With probability P_M (0.01 is reasonable) mutate each one of its bits

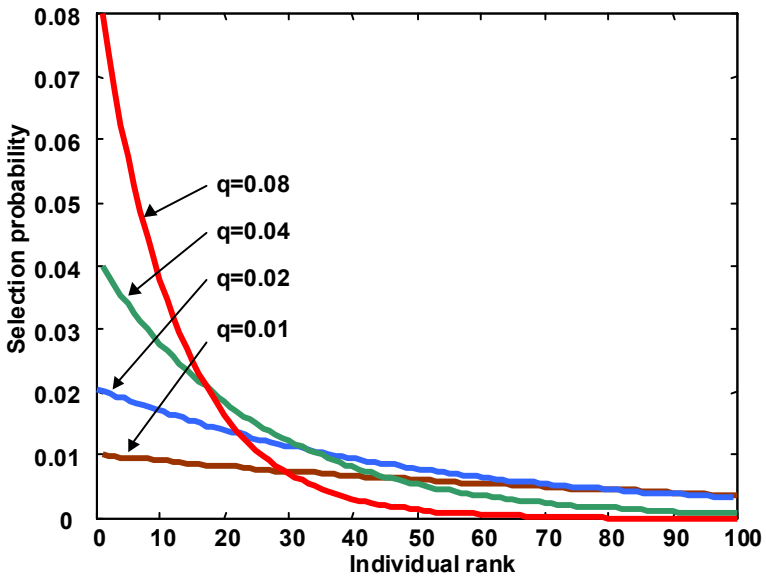


Selection methods

- **The selection of individuals is based on their fitness (the value of the objective function)**
- **We will describe a selection method called Geometric selection**
  - Several other methods are available: Roulette Wheel, Tournament Selection, etc.
- **Geometric selection**
  - The probability of selecting the r^{th} best individual is given by the geometric probability mass function

$$P(r) = q(1-q)^{r-1}$$

    - q is the probability of selecting the best individual (0.05 is a reasonable value)
  - Therefore, the geometric distribution assigns higher probability to individuals ranked better, but also allows unfit individuals to be selected
- **In addition, it is typical to carry the best individual of each population to the next one**
  - This is called the Elitist Model



GA parameter choices for Feature Selection

- **The choice of crossover rate P_C is important**
  - You will want a value close to 1.0 to have a large number of offspring
  - The optimal value of P_C is inversely proportional to population size [Doak, 1992]
- **The choice of mutation rate P_M is very critical**
  - An optimal choice of P_M will allow the GA to explore the more promising regions while avoiding getting trapped in local minima
    - A large value (i.e., P_M>0.25) will not allow the search to focus on the better regions, and the GA will perform like random search
    - A small value (i.e., close to 0.0) will not allow the search to escape local minima
- **The choice of 'q', the probability of selecting the best individual is also critical**
  - An optimal value of 'q' will allow the GA to explore the most promising solution, and at the same time provide sufficient diversity to avoid early convergence of the algorithm
- **In general, poorly selected control parameters will result in sub-optimal solutions due to early convergence**

Search Strategies, summary

	Accuracy	Complexity	Advantages	Disadvantages
Exhaustive	Always finds the optimal solution	Exponential	High accuracy	High complexity
Sequential	Good if no backtracking needed	Quadratic $O(N_{EX}^2)$	Simple and fast	Cannot backtrack
Randomized	Good with proper control parameters	Generally low	Designed to escape local minima	Difficult to choose good parameters

- **A highly recommended review of the material presented in these two lectures is**

Justin Doak
"An evaluation of feature selection methods and their application to Computer Security"
University of California at Davis, Tech Report CSE-92-18