

# Programming for Chemical and Life Science Informatics

I573 - Week 6  
(Optimization and Machine Learning)

Rajarshi Guha

17<sup>th</sup> February, 2009

- The roles of optimization and machine learning
- What can we optimize?
  - ▶ Overview of some numerical optimization methods
- Search algorithms
  - ▶ Why do we need them?
  - ▶ Overview of searching algorithms
  - ▶ Relation to optimization

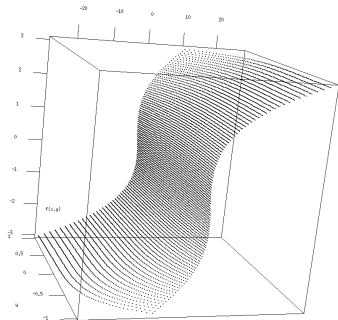
# The Role of Optimization

- Why optimize?
  - ▶ Time is limited
  - ▶ Resources are limited
- What can we optimize?
  - ▶ running time, memory
  - ▶ values of a function, parameters to a function
- What is the optimal condition?
  - ▶ User or problem defined

# Optimization as Searching

- Optimization can be considered a form of search
- Here the *space* we are searching is a parameter space
- Consider  $f(x, y) = \cos^2 x + y^2 - e^{xy} \sin x$

- The optimal value of this function is determined by a specific value of  $x$  and  $y$
- This is a 2D parameter space
- Essentially, we search over the surface



# Unconstrained Optimization

- No bounds on the function parameters
- *Find the positions of 3 carbons so C-C-C has the minimum energy*
  - ▶ Here the parameters are the  $(x, y, z)$  coordinates of the carbons
  - ▶ There is no limit on the values of the individual coordinates
  - ▶ We want the minimal energy

# Constrained Optimization

- Some parameters are to be bounded
- We can have two types of constraints
  - ▶ Equality constraint
  - ▶ Inequality constraints (lower or upper bounds)
- *Find the minimum energy conformer*
  - ▶ Parameters are angles and atom positions
  - ▶ But all angles cannot be rotated

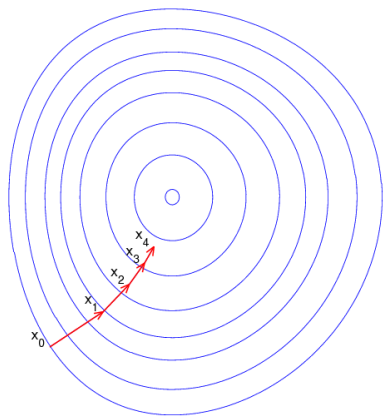
- Line search (1D) - **Descent methods**
- We can do faster, better if we look at the slope of the function - **Gradient based methods**
- We can further improve by approximating second derivatives - **Conjugate gradient methods**
- In general, the goal is to somehow move to the lowest point of the function
  - ▶ Boils down to finding the right direction!

# Descent Methods

- Goal: *minimize*  $f(x)$
- Start with a value of  $x$
- Choose a direction,  $\Delta x$  and a step size  $t$ 
  - ▶ So the new value,  $x' = x + \Delta x t$
- Direction should be such that  $f(x') < f(x)$
- Repeat till we reach the stopping condition, usually defined as  $f(x') - f(x) \leq \epsilon$ , where  $\epsilon$  is some small number
- This method usually has a very slow convergence

# Gradient Descent Methods

- Similar to descent method
- Here we use  $\Delta x = -\nabla f(x)$
- Better than before, but still slow
- Steepest descent
  - ▶ We find the most negative directional derivative
  - ▶ Still, not much better than before

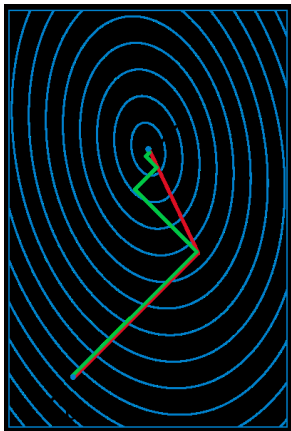


# Newtons Method

- Based on Newtons method of root finding

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

- For multi-dimensional problems we use the gradient and the inverse Hessian
  - ▶ Requires that the function be twice differentiable
  - ▶ Requires that we can actually invert the Hessian
- See [this](#) article for a good explanation of conjugate gradient methods



## BFGS

- Based on the idea of Newton's method
- Uses an approximation to the Hessian using a conjugate gradient method
- very fast convergence, accurate
- Memory intensive

## Truncated Newton

- Uses a Taylor series expansion of Newton's equation
- Does not need to go to the full solution

## More Than Just Parameters?

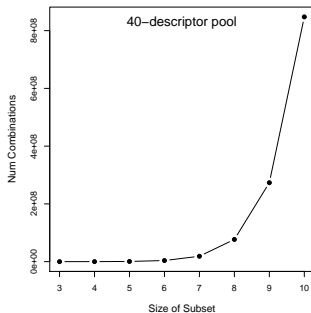
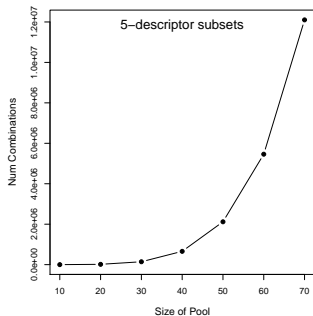
- Traditional optimization considers a function and its parameters
- What if we can't define the gradient of a function?
- What if we need to choose an *optimal function*
  
- Lets say we have 10 variables,  $x_1, \dots, x_{10}$
- Find  $x_i, x_j, x_k$  such that  $f(x_i, x_j, x_k)$  is minimum
- What if we have 3 different functional forms:  $f()$ ,  $g()$  and  $h()$

# Genetic Algorithms

- Can be used to optimize the values of a specific set of parameters
  - ▶ Find  $x, y$  such that  $f(x, y)$  is minimum
- Can be used to search for an optimal set of parameters from a pool of parameters
  - ▶ Select  $x_i, x_j \in \{x_1, \dots, x_n\}$  such that  $f(x_i, x_j)$  is minimum
- Based on the principles of Darwinian evolution
- One example of the broader field of evolutionary programming

# The Example Problem

- We have 20 molecular descriptors for 100 molecules
- We have observed LD<sub>50</sub> values for these molecules
- Goal: *find a subset of 4 descriptors such that we can build an OLS model to predict the LD<sub>50</sub> with minimal error*
- This is a trivial example since there are  $\binom{20}{4}$  (i.e., 4845) possible combinations of 4 descriptors



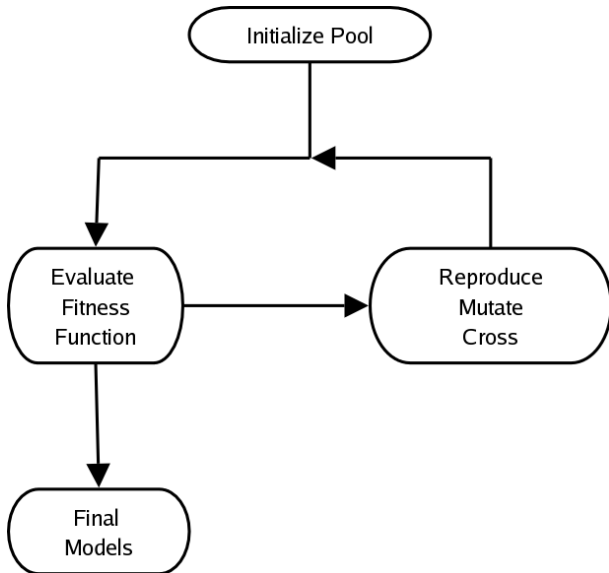
# Components of a GA

- Each descriptor is a gene
- A set of descriptors (i.e., a subset) represents the chromosome
  - ▶ Each descriptor is a gene
  - ▶ In this case the chromosome is of fixed length
- We consider an individual to have a single chromosome
  - ▶ Each individual represents a possible solution to the problem
- An initial random collection of individuals makes up the population
  - ▶ Here “random” implies that the descriptors comprising an individual's chromosome are randomly selected from the pool
- The population *evolves*
  - ▶ cross-over
  - ▶ mutation

# The Goal of the GA

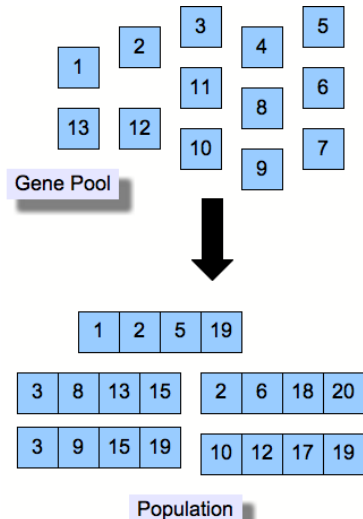
- Find the *fittest* individual(s)
- Fitness is defined by an objective function
  - ▶ This takes the individual's chromosome and returns a numerical value
  - ▶ The objective function could be anything
  - ▶ In our case, we use the chromosome (i.e., the descriptors) to build an OLS model and return the RMSE as the value of the objective function
- The GA optimizes the value of the objective function
- Implies that the fittest individual will have the set of genes (i.e., descriptors) that leads to an OLS model with the lowest RMSE

# GA Flowchart



# Starting the GA

- Create 100 random individuals
- Evaluate the fitness of everybody
- Get the average fitness of the population



# The Mating List

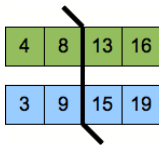
- Select individuals to mate
- The list is equal in size to the population

## Procedure

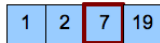
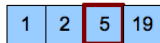
- Select all individuals that have *above average* fitness
- This results in maximization of fitness
- If we want minimization, reformulate the objective function
- Remainder can be filled in a variety of ways
  - ▶ Random
  - ▶ Roulette wheel selection

# Reproducing

- We want to create a child population
- Choose 2 individuals from the mating list randomly
- Perform cross-over
- Perform mutation
- Usually done probabilistically



Cross-over



Mutation

# The Next Generation

- Once we have the required number of children they replace the original population
- The cycle is repeated  $N_{gen}$  times
- Over time the population gets dominated by the fittest individuals
  - ▶ Objective function converges
- At this point the genes (i.e., descriptors) of the fittest individuals represent the 4 descriptors that give the OLS model with the lowest error

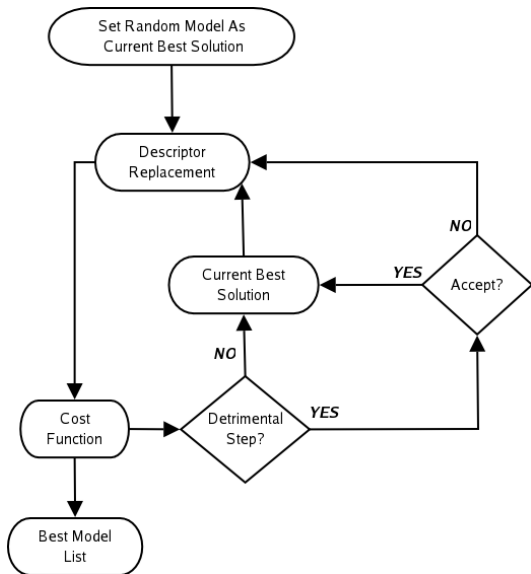
# Issues with GA's

- Very flexible
- Good for an uneven parameter space
- Optimality is not guaranteed
  - ▶ The best solution provided by the GA may not be a globally optimal solution
- The algorithm can converge prematurely
  - ▶ Indicates that it's stuck in a local minima
- Good idea to investigate the top  $N$  models after the run is done

# Simulated Annealing

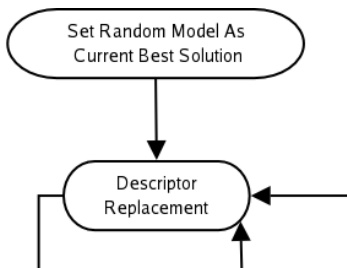
- A stochastic optimization method
- Based on the idea of physical annealing of glasses, metals
- Heat the glass to a high temperature - becomes randomly ordered
- Let it cool slowly (annealing schedule)
- At the end of the cooling process, it should be in the most ordered (lowest energy) state

# SA Workflow



# How Does an SA Optimize?

- Define a cost function,  $C$
- Start with an initial configuration (descriptor set),  $X$
- Make a small change to it (randomly change one descriptor), giving  $Y$

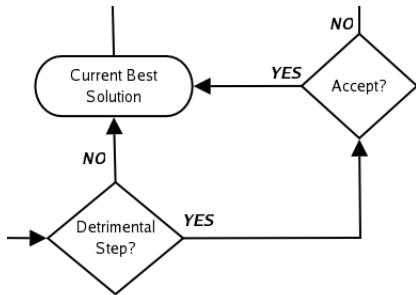


# How Does an SA Optimize?

- If  $C(X) < C(Y)$  it's a better set of descriptors
  - ▶ This is the current best solution
  - ▶ Go back to the beginning
- If  $C(X) > C(Y)$ , we may or may not accept it

- Acceptance is controlled by a Boltzmann type probability

$$\exp\left(-\frac{C(X) - C(Y)}{kT}\right)$$



- Similar in nature to GA's
- Good for uneven parameter spaces
- Combinatorial problems
- Good for functions with no well defined gradient
- Doesn't necessarily get the global optimum

- Ant colony methods
  - ▶ Tutorial
  - ▶ Izrailev, S. et al.
  - ▶ Shen, Q. et al.
- Particle swarm methods
  - ▶ Tutorial
  - ▶ Agrafiotis, D. et al.
  - ▶ Lin, W.Q. et al.
- Tabu Search

- Numerical optimization
  - ▶ [netlib.org](http://netlib.org) - huge repository of algorithm implementations
  - ▶ [GNU Scientific Library](#)
  - ▶ Matlab, Mathematica, Maple
- Genetic algorithms
  - ▶ [Genetic Algorithm Utility Library](#)
  - ▶ GA packages in BioPython ([Bio.GA](#))
  - ▶ [Free software](#)
  - ▶ [Commercial software](#)