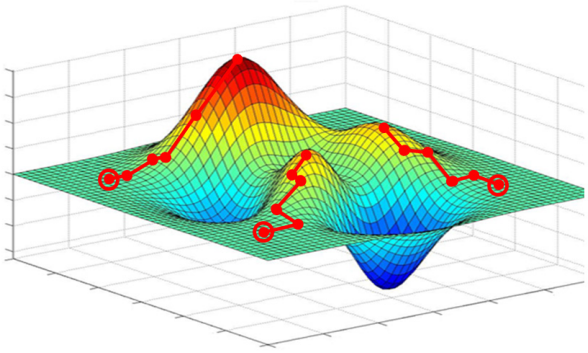
Optimization



An overview of global and local methods, with applications in bioinformatics

Today's Lecture

- What is optimization?
- Why is it important in bioinformatics?
- Stochastic methods vs non-stochastic methods
- A "top hits" selection of algorithms
- How do I choose what algorithm to use?
- Where can I learn more?

What is optimization?

Given an objective function f, find the set of values $\boldsymbol{x}$ that maximize (or minimize) the value of $f(\boldsymbol{x})$.

A simple example: Aligning two DNA sequences (with no gaps)

$f(x) = \sum_{\text{base pairs}} \{1 \text{ if base pairs match, else } 0\}$

x = position (or shift) of one sequence relative to the other

$0 \leq x \leq$ length of the longer sequence

Optimization in Bioinformatics

- Sequence alignment
- Structure prediction / alignment
- Energy minimization
- Biosynthetic pathway analysis
- Virtually all pattern recognition problems

Stochastic vs Non-Stochastic

Stochastic

- Start at multiple points in the function space
- Only need to know how to evaluate the objective function
- Are by far the most common for global optimization tasks

Non-Stochastic

- Typically follow a single trajectory
- Often make use of the particular mathematics of the objective function (e.g. take derivatives)
- Are by the most common for local optimization tasks

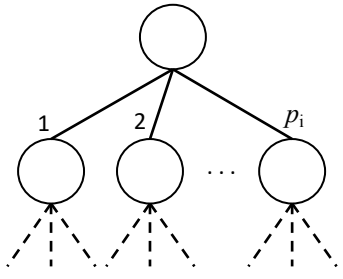
Branch and Bound

Objective function: $f(\boldsymbol{x})$, where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$

and

x_i can only have p_i discrete values

Branch on selecting a value for x_i



Using a bounding function, determine if the best the selection of x_i can yield is worse than the worst any other selection of x_i could yield. If so, prune that branch of the tree.

Leaves of the tree represent all possible solutions

Branch and Bound example

The dead-end elimination theorem and its use in protein side-chain positioning, Desmet et.al., **Nature, 1992**

The potential energy of a protein system consisting of a set of residue side chains i each in a particular rotameric state r and embedded in a template of fixed atoms (for example, main-chain atoms) can be written as

$$E_{\text{global}} = E_{\text{template}} + \sum_i E(i_r) + \sum_i \sum_j E(i_r j_s); \quad i < j \qquad (1)$$

where E_{template} is the template self energy, $E(i_r)$ the potential energy of the side chain atoms of the rotamer i_r in the force field of the template, and $E(i_r j_s)$ the nonbonded pairwise interaction energy between rotamers i_r and j_s. The calculation of the

Branch on the choice of rotamer for residue i, then test the following inequality:

$$E(i_r) + \sum_j \min_s E(i_r j_s) > E(i_t) + \sum_j \max_s E(i_t j_s); \quad i \neq j$$

Lowest energy using rotamer r for residue i

Highest energy using rotamer t for residue i

If true, then rotamer r of residue i is not in the set of rotamers that make up the global minimum solution.

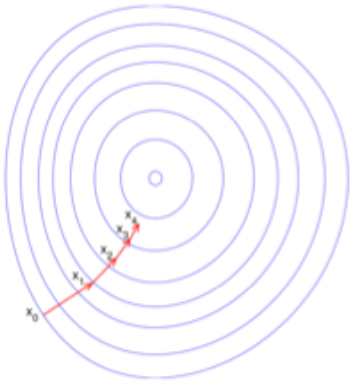
Gradient Descent

Objective function: $f(\boldsymbol{x})$, which is first-order differentiable

Start with point $\boldsymbol{x}_0$ and calculate $f(\boldsymbol{x}_0)$

Choose a new point $\boldsymbol{x}_1 = \boldsymbol{x}_0 - \beta \nabla f(\boldsymbol{x}_0)$

where β is the step size



β can either be fixed, or we can try to be clever about it

Ideally, we want $\beta = \beta_m$, where

$$\beta_m = \min_{\beta} f(\boldsymbol{x}_{i+1} - \beta \nabla f(\boldsymbol{x}_i))$$

One option is to solve for β directly by solving

$$\frac{d\, f(\boldsymbol{x}_{i+1} - \beta \nabla f(\boldsymbol{x}_i))}{d\, \beta} = 0$$

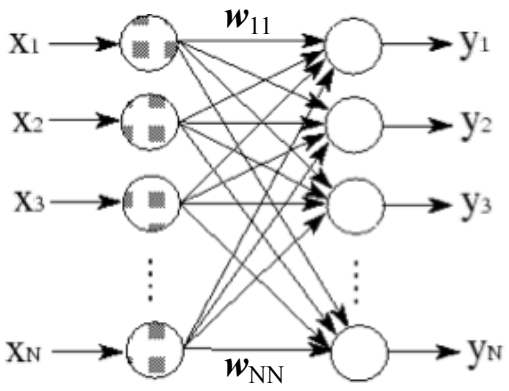
Conjugate Gradient Method

Or we can try to quickly find a β that Lowers $f(\boldsymbol{x}_{i+1} - \beta \nabla f(\boldsymbol{x}_i))$ by an acceptable amount

Typically this is a line search, sometimes with conditions based on the assumed smoothness of f

Gradient Descent example

Optimizing weights in a neural network



Given a set of training data points $(\boldsymbol{x}, \boldsymbol{t})$,

0. Start by giving all of the weights w_{ij} random small values

1. For each weight w_{ij} , set $\Delta w_{ij} = 0$

2. For each point in the training data $(\boldsymbol{x}, \boldsymbol{t})$,

   Set the inputs to $\boldsymbol{x}$
   Compute the values of the outputs $\boldsymbol{y}$
   For each weight w_{ij} , $\Delta w_{ij} = \Delta w_{ij} + (t_i - y_i) y_j$

3. For each weight w_{ij} , set $w_{ij} = w_{ij} + \beta \, \Delta w_{ij}$ β = learning rate

4. If the sum of changes of the weights is below the convergence threshold, stop. Else, go to step 1.

Newton's Method

Expand $f(\mathbf{x})$ by its Taylor series about the point $\mathbf{x}_n$

$$f(\mathbf{x}_n + \delta\mathbf{x}) \approx f(\mathbf{x}_n) + \mathbf{g}_n^\top \delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\top \mathrm{H}_n \, \delta\mathbf{x}$$

where the gradient is the vector

$$\mathbf{g}_n = \nabla f(\mathbf{x}_n) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_N}\right]^\top$$

and the Hessian is the symmetric matrix

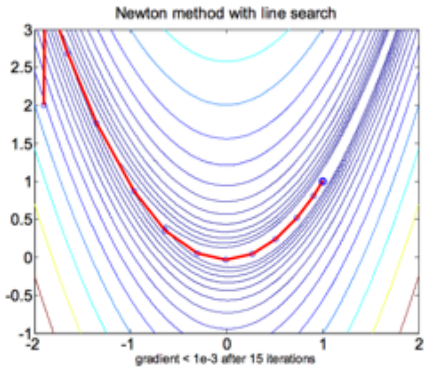
$$\mathrm{H}_n = \mathrm{H}(\mathbf{x}_n) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \vdots & \ddots & \\ \frac{\partial^2 f}{\partial x_1 \partial x_N} & & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix}$$

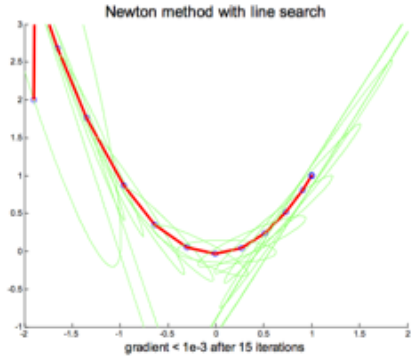
For a minimum we require that $\nabla f(\mathbf{x}) = \mathbf{0}$, and so

$$\nabla f(\mathbf{x}) = \mathbf{g}_n + \mathrm{H}_n \delta\mathbf{x} = \mathbf{0}$$

with solution $\delta\mathbf{x} = -\mathrm{H}_n^{-1}\mathbf{g}_n$. This gives the iterative update

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathrm{H}_n^{-1}\mathbf{g}_n$$



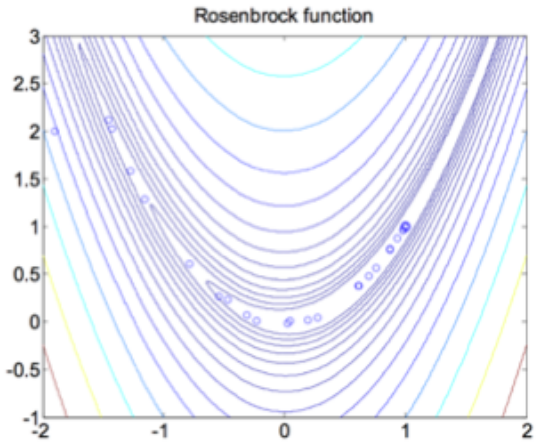


ellipses show successive quadratic approximations

- The algorithm converges in only 15 iterations compared to the 101 for conjugate gradients, and 200 for simplex
- However, the method requires computing the Hessian matrix at each iteration – this is not always feasible

For many practical problems, calculating the Hessian at every step becomes too expensive

There are a host of "Quasi-Newton" methods that keep a rolling estimate of the Hessian that's cheaper to compute



25 iterations to converge instead of 15

Linear Programming

A large number of real-world problems can be expressed as linear programming problems

Maximize $f(\boldsymbol{x}) = c_0x_0 + c_1x_1 + \ldots + c_nx_n$

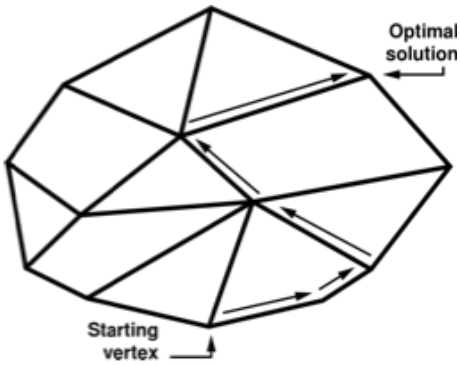
Subject to the constraints

$$\begin{aligned} a_{00}x_0 + a_{01}x_1 + \ldots + a_{0n}x_n &\leq b_0 \\ a_{10}x_0 + a_{11}x_1 + \ldots + a_{1n}x_n &\leq b_1 \\ \cdots & \\ a_{n0}x_0 + a_{n1}x_1 + \ldots + a_{nn}x_n &\leq b_n \end{aligned} \quad \text{and} \quad \begin{aligned} x_0 &\geq 0 \\ x_1 &\geq 0 \\ \cdots & \\ x_n &\geq 0 \end{aligned}$$

Several instances of linear programing (e.g. integer linear programming) are NP-hard

The most widely used algorithm to solve linear programming problems is the simplex method

Move along the vertices of a polytope (simplex) defined by the linear constraints.

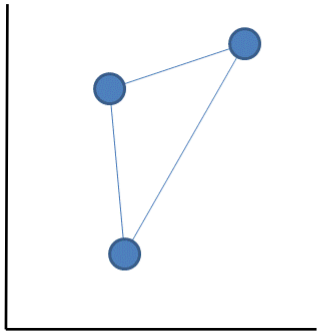


For large linear programming problems, direct implementation is inefficient – most implementations used a revised method that expects the matrix of constraints to be sparse.

Nelder-Mead (aka Downhill Simplex)

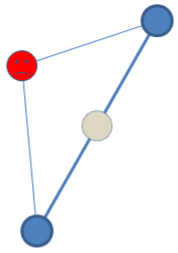
Minimize $f(\boldsymbol{x})$, where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$

Start by defining a "polytope" with n+1 vertices

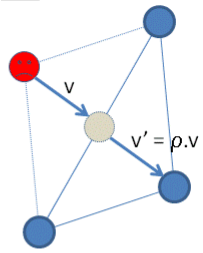


(1) Evaluate f at each of the vertices and find the best and worst vertices

(2) Calculate the centroid of the simplex, leaving out the worst vertex



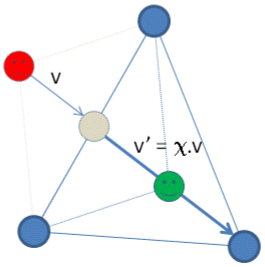
(3) Reflect the worst point through the centroid and calculate its value



Typically, ρ = 1.0

If the new vertex is neither the best or the worst, then keep it, and go back to (1)

(4) If the new vertex is the best, expand the simplex by moving outward further along the line from the centroid

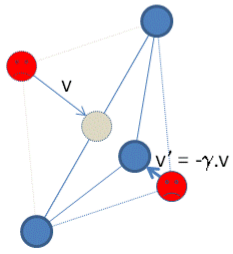


Typically, χ = 1.0

If the new vertex is better than the reflected vertex, then keep it and go back to (1)

Else, keep the reflected vertex and go back to (1)

(5) If the new vertex is still the worst, contract the simplex by moving it towards the centroid

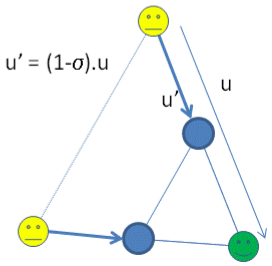


Typically, γ = 0.5

If this new vertex is better than the original worst vertex, then keep it and go to back to (1)

If this new vertex is worse than the original worst vertex, then contract the entire simplex

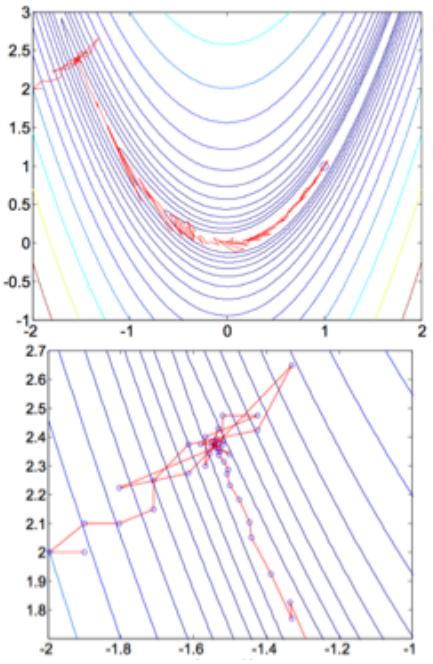
(6) For all vertices except the best one, move towards the best vertex, contracting the overall simplex

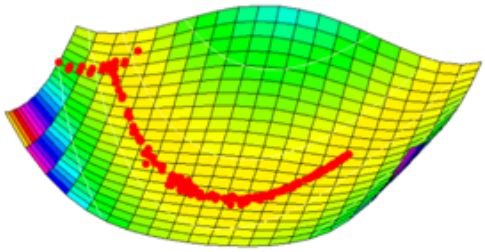


Typically, u = 0.5

Termination conditions vary

- Difference between best and worst vertex values
- Difference of parameters of best and worst vertex
- Overall size of the simplex
- Max number of iterations



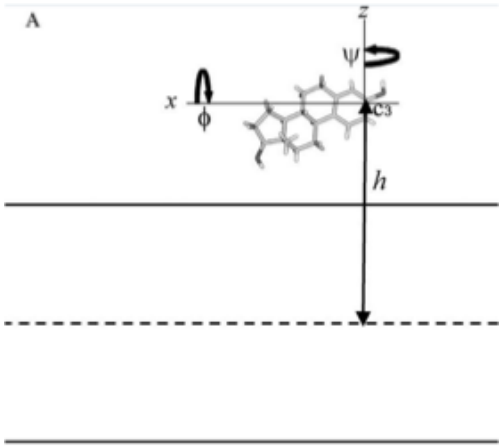


200 iterations in Matlab

- Choice of the initial simplex vertices can have a significant impact – often multiple runs from different starting points are used
- This is a “direct” method (does not require a differentiable function)
- The algorithm handles noisy functions reasonably well
- Does not guarantee a global minimum, but can do some hill climbing

Downhill Simplex example

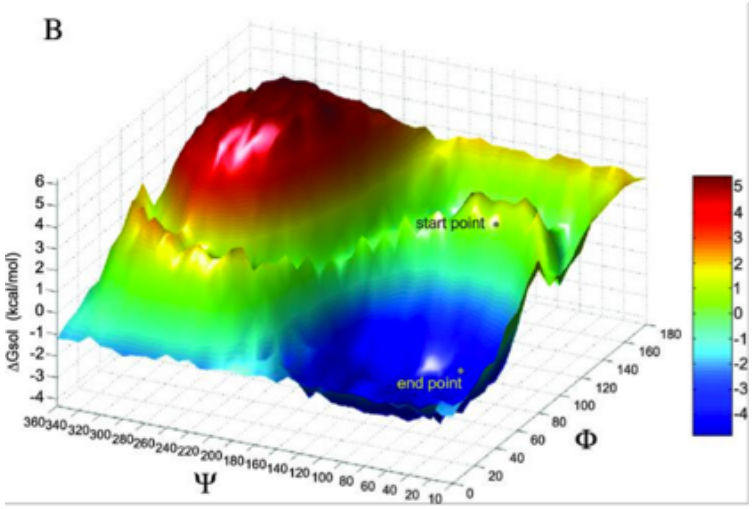
Free Diffusion of Steroid Hormones Across Biomembranes: A Simplex Search with Implicit Solvent Model Calculations, Oren et. al., **Biophys J. Aug 2004**



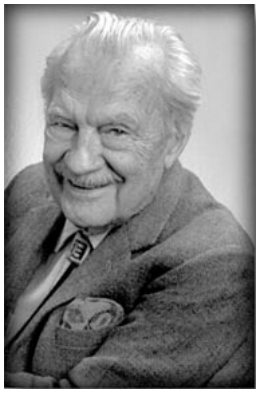
Find the values of Ψ, Φ, and h that minimize ΔG_{tot}

$$\Delta G_{tot} = \Delta G_{sol} + \Delta G_{imm}$$

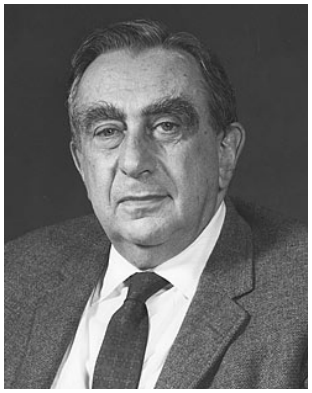
30 manually selected starting configurations
80 iterations per run



Monte Carlo / Simulated Annealing



Nicholas
Metropolis



Edward
Teller

John
von Neumann

Monte Carlo / Simulated Annealing

Minimize $f(\boldsymbol{x})$, where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$

(0) We start in state $\boldsymbol{x}_0$ and calculate its energy, e_0 $(=f(\boldsymbol{x}_0))$

(1) We propose a new state $\boldsymbol{x}_1$ and calculate its energy, e_1

(2) We accept $\boldsymbol{x}_1$ as our new state according to a probability function, $P(e_0, e_1, T)$

(3) If improvement is below threshold, or max number of iterations is reached, stop. Else, go to (1).

In the standard Boltzmann style annealing,

$$g(\vec{x}) = (2\pi T)^{-\frac{D}{2}} \exp\left(-\frac{(\Delta\vec{x})^2}{2T}\right),$$

Probability density for selecting a new state from the current one

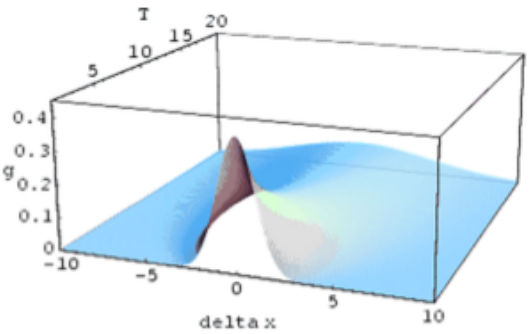
$$h(\vec{x}) = \frac{1}{1+\exp\left(\frac{E_{k+1}-E_k}{T}\right)},$$

Probability of accepting the new state

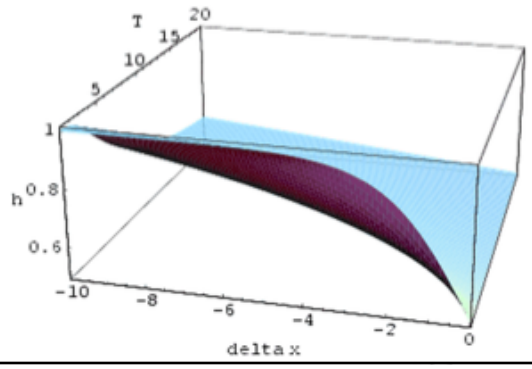
$$T(k) = \frac{T_0}{\ln k}$$

Annealing schedule

There are many, many variants of MC / SA – generally, one tailors the implementation to the specific problem.



Smaller moves more likely as temperature drops



Odds of accepting an uphill move less likely as temperature drops

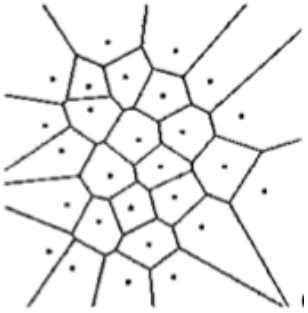
Simulated Annealing example

A simulated annealing approach to define the genetic structure of populations, Dupanloup et. al., **Molecular Biology, 2002**

Given the genetic samples of populations with known geographic distribution, group them in such a way as to maximize the $\mathrm{F_{CT}}$ index, the proportion of total genetic variance due to differences between groups of populations

Preliminary steps

1 A set of Voronoi polygons (Voronoi 1908) is constructed from the geographical location of the n sampled points (see Fig. 1).
2 An arbitrary partition of the n populations into K groups is initially chosen at random (in our case, each group, except one, is composed of a single population and the last group contains the populations not assigned to any other groups).
3 The genetic barrier(s) between the K groups are identified as edges of Voronoi polygons separating groups of populations.
4 The F_{CT} index associated to the K groups is computed.



Simulated annealing steps

5 We select an edge at random on a given barrier.
6 The two populations located on both sides of the selected edge are identified, and one population chosen at random is assigned to the group of the other population.
7 The genetic barrier is modified by updating the list of edges separating the newly defined groups of populations [the edge selected in step 6 is replaced by the edges surrounding the population whose group location has changed (see Fig. 2A)].
8 The new F_{CT} value (noted F_{CT}*) associated with the new partition is computed.
9 The new structure is accepted with probability

$$p = \begin{cases} 1 & \text{if} \quad F_{CT}^* \geq F_{CT} \\ e^{(F_{CT}^* - F_{CT})S^A} & \text{if} \quad F_{CT}^* < F_{CT} \end{cases}$$

where S is the number of steps performed in the simulated annealing process, and A is an arbitrary constant controlling the speed of the cooling process.

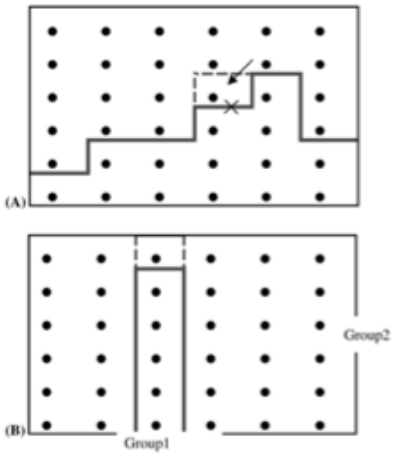


Fig. 2 Modification of the group structure under steps 5 and 6 of the simulated annealing algorithm. (A) The bold line delineates the limit between the two groups before the modification. The cross designates the edge selected at random under step 5. The arrow designates the population whose group location is to be changed under step 6. The dashed line(s) define the new barrier(s) between groups after step 6 has been performed. (A) Normal case leading to geographically adjacent groups. (B) Case in which the allocation of one population from group 1 to group 2 leads to the fragmentation of group 2 into two distinct sets of adjacent populations.

Genetic Algorithms

The function we want to maximize is the evolutionary fitness of a population of potential solutions

Individual = $\boldsymbol{x}^i = \{x^i_1, x^i_2, \ldots, x^i_n\}$

Population = $<\boldsymbol{x}> = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^p\}$

Fitness = $f(<\boldsymbol{x}>) = \{f(\boldsymbol{x}^1), f(\boldsymbol{x}^2), \ldots, f(\boldsymbol{x}^p)\}$

The art of using a genetic algorithm is in

1. Representing a solution to the overall problem as an individual
2. The functions that mimic how individuals interact to create new potential solutions

choose an initial population

determine the fitness of each individual

perform selection

while termination criteria not met

- perform crossover
- perform mutation
- determine the fitness of each individual
- perform selection

A GA is not guaranteed to converge on the global optimum

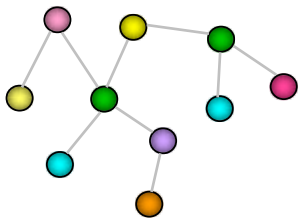
When is a GA a good option?

- You don't need the optimal solution
- You'd like to have multiple "decent" solutions
- The fitness function is noisy
- The fitness function is effectively a black box
- The fitness landscape has lots of local minima
- It's not too expensive to evaluate the fitness function

Genetic Algorithms example

A genetic algorithm for structure-based de novo design, Pegg et. al., **Journal of Computer-Aided Molecular Design, 2001**

Each compound is an acyclic graph of fragments



Up to 16 fragments/compound

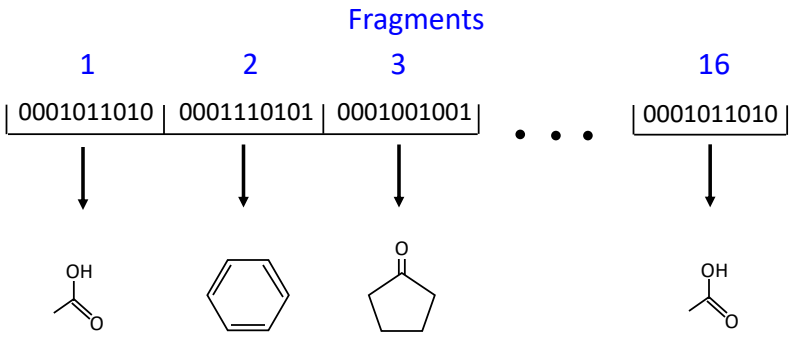
Up to 1023 fragment types

Each compound is has a 'genome' of 310 bits

160 bits for fragment identities
150 bits for connectivity

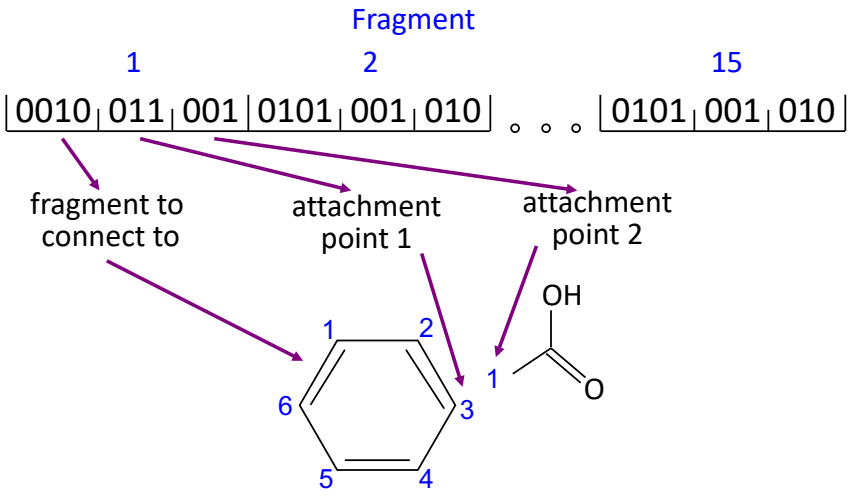
Identity ‘genes’

Each set of 10 bits maps to a fragment type



Connection ‘genes’

16 fragments requires only 15 connections



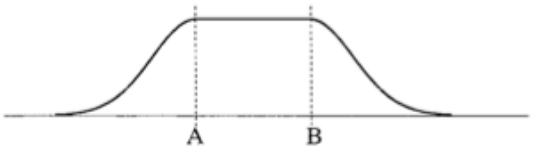
Initial Population

- Diverse
  - Random sizes, random fragments, random attachments (within constraints)
- Compound-based
  - Starting with a user-defined molecule, add/subtract/swap 1-2 fragments at random
- Scaffold
  - User-chosen fragment

The fitness function is a collection of drug design related terms:

$$\text{fitness} = C_{\text{dock}}S_{\text{dock}} + C_{mw}S_{mw} + C_{\text{rot}}S_{\text{rot}} + \\ + C_{h\text{bond}}S_{h\text{bond}} + C_{c\log p}S_{c\log p},$$

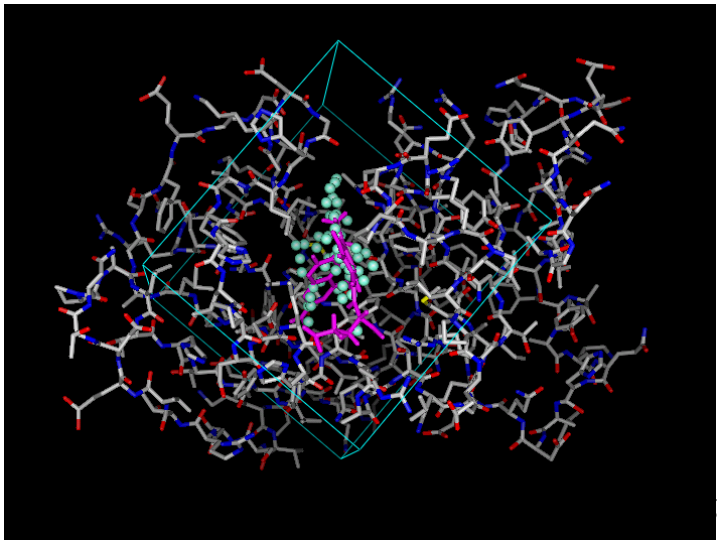
where C_n is the coefficient for score S_n.



$$S(x) = \begin{cases} 1 & \text{if } A < x < B \\ N(\mu, \sigma) & \text{if } x < A \text{ or } x > B \end{cases}$$

DOCK 4.0

A method to estimate the affinity of compounds to a receptor structure



Breeding

Crossover

Swap sets of fragments

Up to half of compound

Parents chosen according to fitness

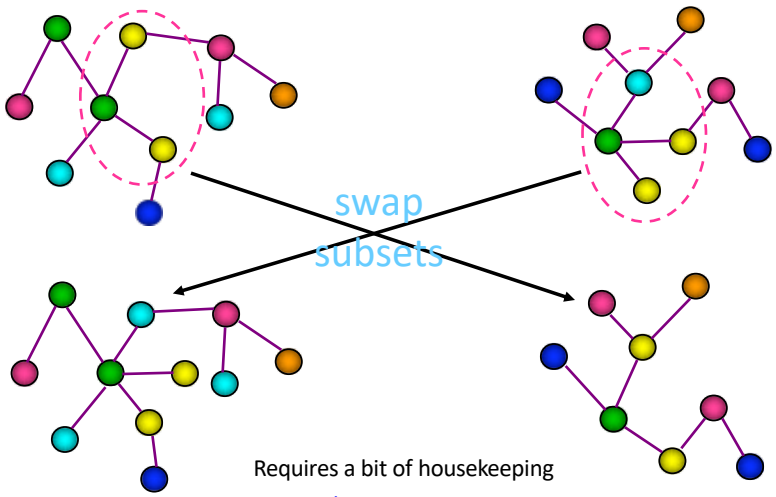
Mutation

Identity changed according to exponential

Local sampling

Connectivity changed by randomly reconnecting edges

Crossover



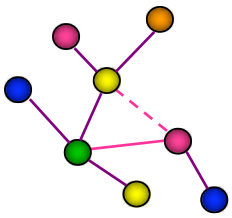
Requires a bit of housekeeping

cycle prevention

valence consistency

Mutation

Connectivity



Reconnect edges at random

without cycles or disconnects

Fragment Identity

Adding or subtracting to the fragment's bitstring

moving up or down in the fragment file

Size of shift according to exponential distribution

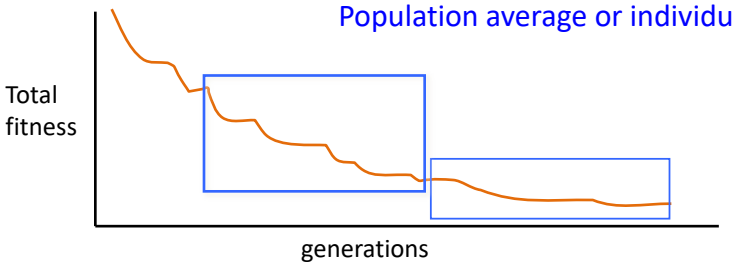
$P(\text{shift} > x) = e^{-\lambda x}$

Termination Conditions

Window-based

Comparison of values from current window to previous

Population average or individual



Goal-based

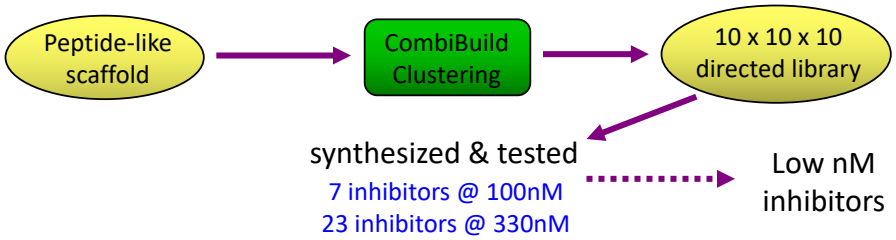
Stop when a specific score has been achieved

Population average or individual

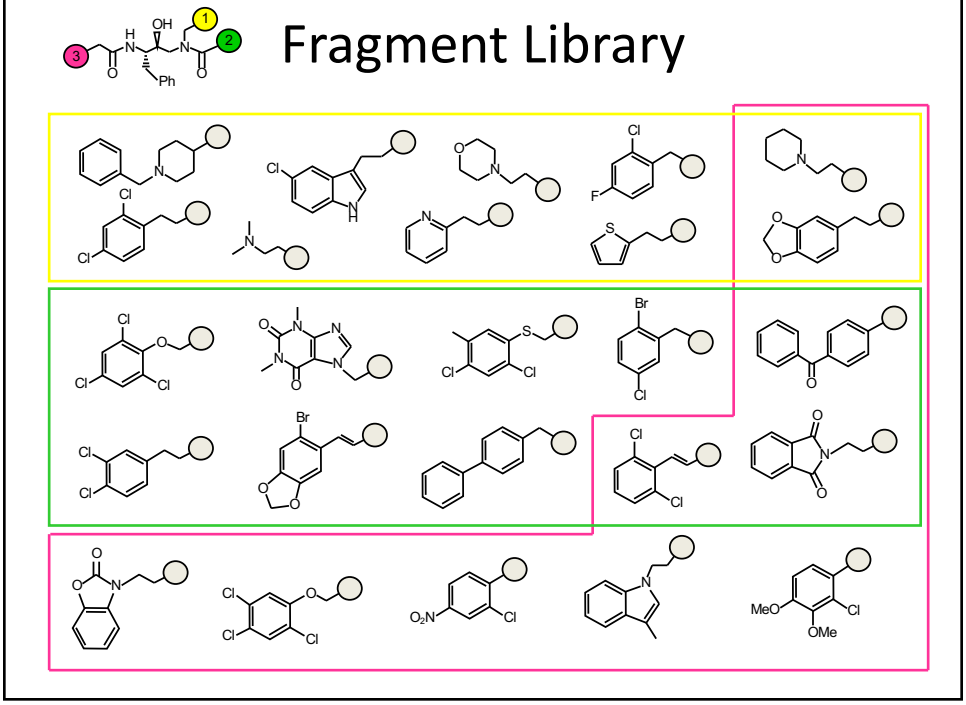
Applying the GA to a control case

Goal: Identify non-peptide inhibitors of Cathepsin D

Implicated in tumor metastasis and Alzheimer's



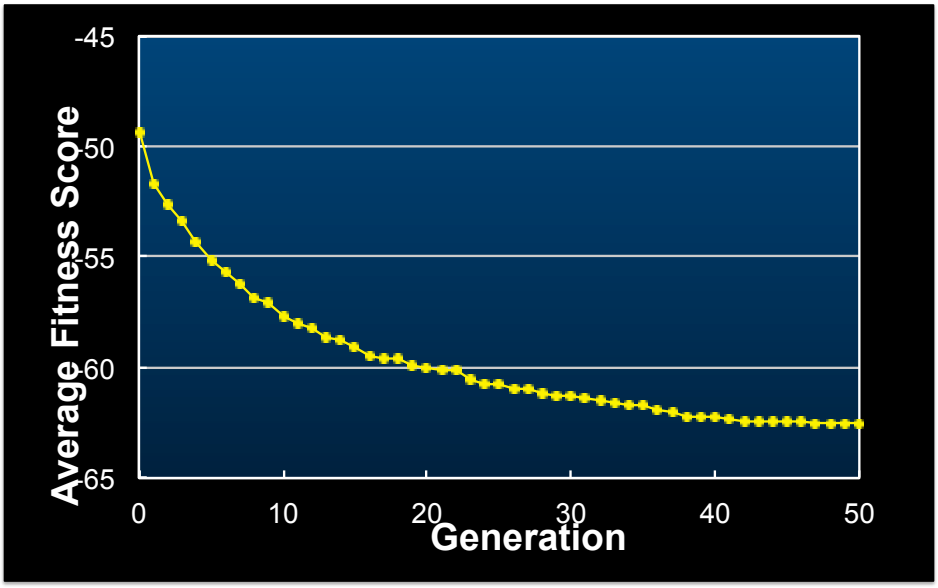
Kick et. al. (1997) Structure-based design and combinatorial chemistry yield low nanomolar inhibitors of cathepsin D. *Chemistry & Biology*, **4**:297-307.



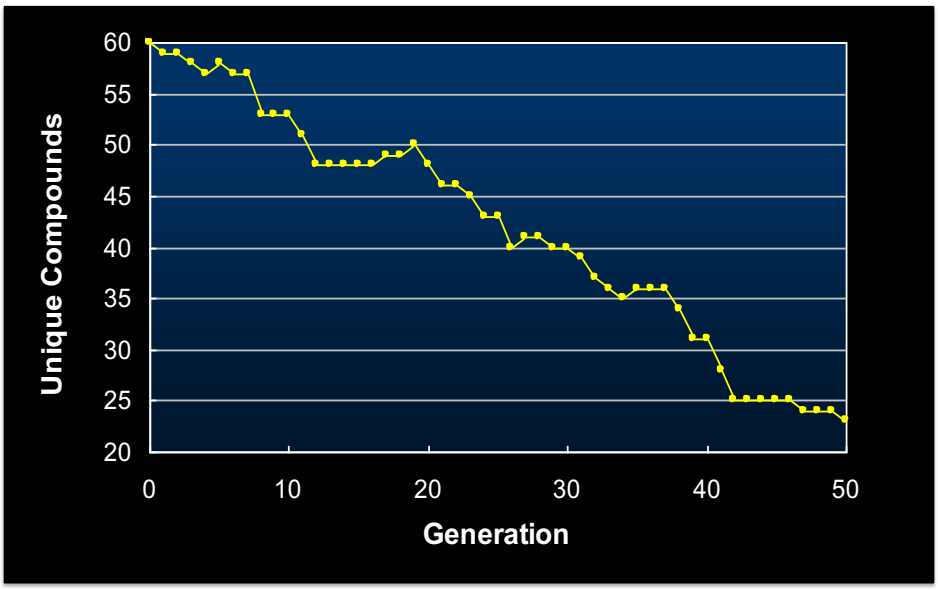
GA Settings

- 12 compounds per generation
  - keep top scoring half
- Crossover & Mutation
- Fitness just DOCK score
- 50 generations
- DOCKed all 15,625 possible compounds

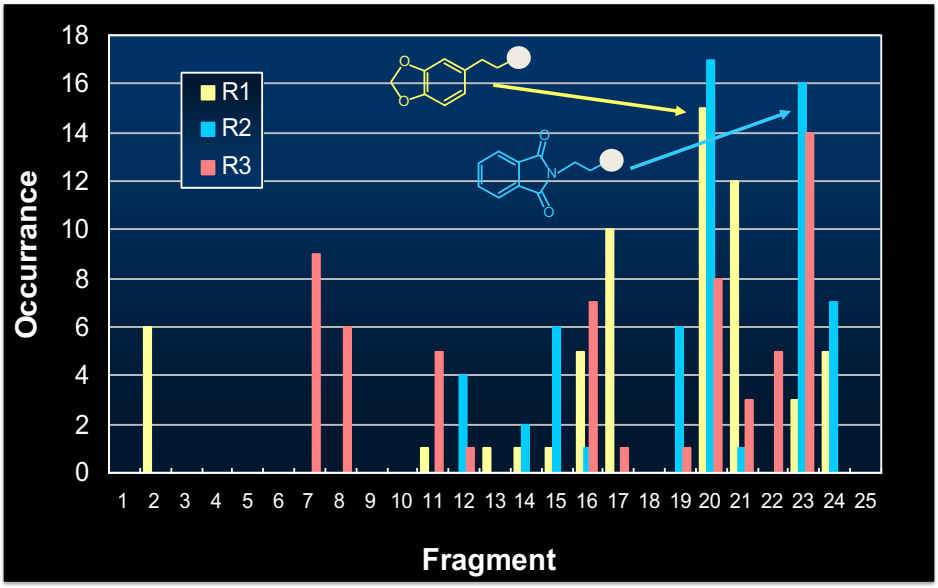
Average Fitness



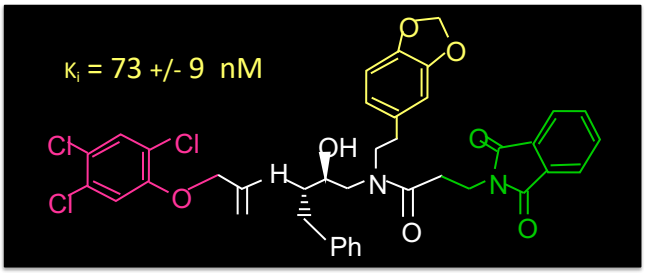
Population Diversity



Fragments of final generation



Real inhibitors



R1: 6/7 (100nM) and 16/23 (330nM) have most common GA fragment

R2: 5/7 (100nM) and 10/23 (330nM) have second most common

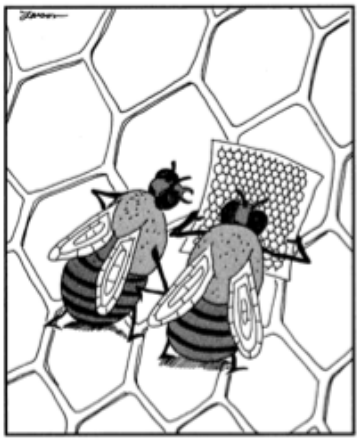
R3: no clear choice from GA or experiment

3 of the 7 fragments from the 330nM inhibitors

A library based on fragments identified by ADAPT (7x7x8) contains the top 4 (of 7) inhibitors at 100nM

Particle Swarms

Introduced in 1995, and inspired by how groups of birds, fish, and insects move towards objectives

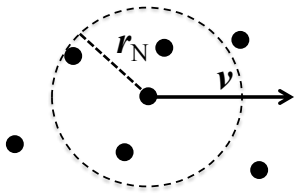


"Face it, Fred—you're lost!"

Minimize $f(\boldsymbol{x})$, where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$

Each "particle" keeps track of

- its current position $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$
- its current velocity $\boldsymbol{v} = \{v_1, v_2, \ldots, v_n\}$
- its previous best position, $\boldsymbol{x}_{lb}$
- the best position in the neighborhood that's been found so far, $\boldsymbol{x}_{gb}$



We start with positions and velocities assigned randomly

At each time step, we update all of the particles

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_{t+1}$$

$$\boldsymbol{v}_{t+1} = \omega \boldsymbol{v}_t + \boldsymbol{C}_1 \varphi_1 (\boldsymbol{x}_{lb} - \boldsymbol{x}_t) + \boldsymbol{C}_2 \varphi_2 (\boldsymbol{x}_{gb} - \boldsymbol{x}_t)$$

ω is an inertial weight factor
φ_1, φ_2 are independent random numbers $[0,1]$
$\boldsymbol{C}_1$ is the "self confidence" constant
$\boldsymbol{C}_2$ is the "swarm confidence" constant

We also place a constant limit on $\boldsymbol{v}$

While particle swarm optimization is not guaranteed to converge on a global minimum, it's an attractive algorithm

- It doesn't require knowledge of the function to be minimized (just a way to evaluate it)
- It's intuitive
- It's easy to implement
- It doesn't have a lot of parameters

It's a good global method for surveying an optimization landscape and/or being combined with a local optimization algorithm

Particle Swarms example

SODOCK: Swarm Optimization for Highly Flexible Protein–Ligand Docking, Chen et. al., **J Comp Chem, 2007**

$$\min E_{\text{total}}(X) = E_{\text{vdw}} + E_{\text{H-bond}} + E_{\text{elec}} + E_{\text{internal}} + E_{\text{desolvation}}$$

$$X = \{t_x, t_y, t_z, n_x, n_y, n_z, \alpha, \phi_1, \ldots \phi_i\}$$

ligand translation | ligand rotation | torsional rotation

Number of particles, N_p	50
Number of immediate neighbors, K	4
Maximal number of function evaluation, $neval_{max}$	250,000
Inertia weight, w	0.9 ~ 0.4 (linear decreasing)
Cognitive weight, c_1	2.0
Social weight, c_2	2.0
Maximal velocity, V_{max}	2.0 Å (for t_x, t_y, and t_z) 1.0 (for n_x, n_y, and n_z) 50° (for α and tor_i)
Maximal steps of local search	50

After each iteration of particle updates, 50 iterations of a local random optimization algorithm is applied to the global best particle

while termination criteria not met,

$$\boldsymbol{y} = \{x_0+\varphi_0, x_1+\varphi_1, \ldots, x_n+\varphi_n\}$$

where each φ_i is an independent random sample from a normal distribution

if $f(\boldsymbol{y}) \leq f(\boldsymbol{x})$, set $\boldsymbol{x} = \boldsymbol{y}$

Minimization by random search techniques, Solis & Wets, **Mathematics of Operation Research, 1981**

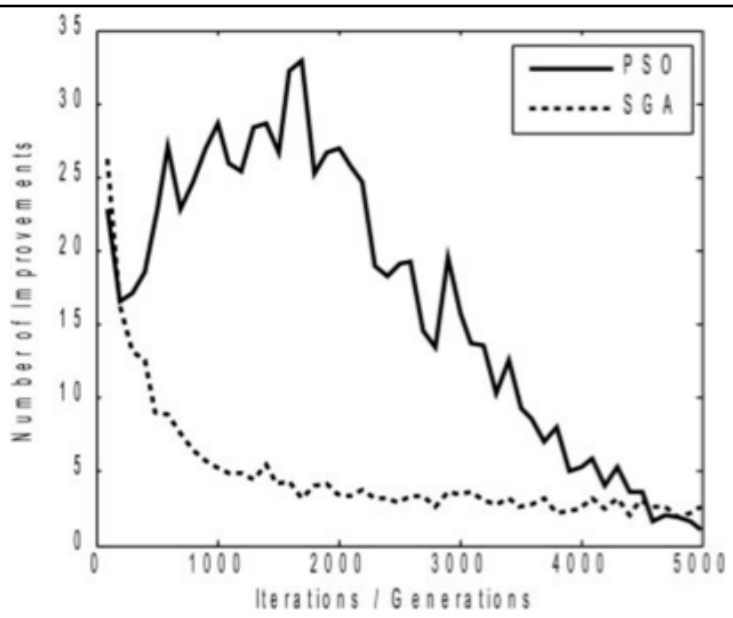


Figure 8. Average number of improvements in terms of docked energy of the best individual in every 100 iterations/generations for 30 independent runs on 1hiv.

Choosing an optimization algorithm

- Can your problem be broken down into independent subproblems?
- Can your problem be described with recursion (dependent subproblems)?
- How good does your solution need to be?
- How many solutions do you want?
- How smooth is your solution landscape?
- How expensive is it to evaluate your objective function?

Some Advice

- Take the time to really understand your problem domain before implementing an algorithm
- Run a simple general algorithm on your problem to get a feel for the landscape
- Consider mixing strategies – many good solutions use a general strategy to find interesting localities, then use a different algorithm to search locally

Where to learn more

Books

- Numerical Recipes in C (Chapter 10) http://www.nrbook.com/a/bookcpdf.php
- An Introduction to Optimization, by Edwin K. P. Chong and Stanislaw H. Zak

Online Courses

- Discrete Optimization (Coursera)https://www.coursera.org/course/optimization
- Introduction to Mathematical Programming (MIT)http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-251j-introduction-to-mathematical-programming-fall-2009/index.htm