Randomized Search Methods

- Random search: a general optimization method.
- Need only objective function; no need gradients.
- Has surprisingly robust behavior.
- Can very quickly obtain a "satisficing" solution.

Simple random search algorithm

- Given the current point $\boldsymbol{x}^{(k)}$, we pick a next candidate point $\boldsymbol{z}^{(k)}$ at random, from a "neighborhood" of $\boldsymbol{x}^{(k)}$ (usually a point close to $\boldsymbol{x}^{(k)}$).
- Compare objective functions: $f(\boldsymbol{z}^{(k)}) < f(\boldsymbol{x}^{(k)})$?
- If f decreased, then jump to $\boldsymbol{z}^{(k)}$; i.e., $\boldsymbol{x}^{(k+1)} = \boldsymbol{z}^{(k)}$.
- Otherwise, stay at $\boldsymbol{x}^{(k)}$ and try again.
- Note that we keep track of the best-so-far point.
- Hence, algorithm has descent property.

- Main problem with previous simple random search algorithm: may get stuck in local minima.
- To prevent getting stuck in local minima, need to have a way of getting out of local minima.
- One way to solve problem: Have very large neighborhood. Under certain conditions, can show convergence to global minimizer!

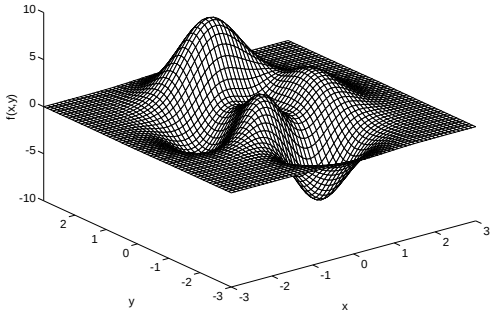
  But this may result in slow convergence.
- Simulated annealing: incorporates a mechanism for climbing out of local minima.

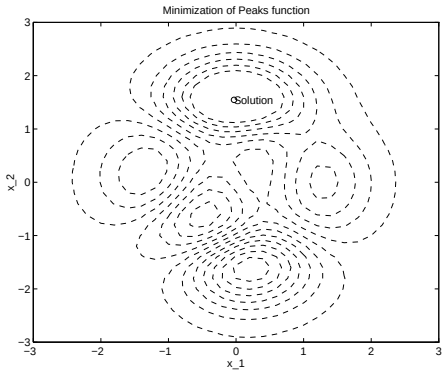
Simulated annealing

- Given the current point $\boldsymbol{x}^{(k)}$, we pick a next candidate point $\boldsymbol{z}^{(k)}$ at random (perhaps a point close to $\boldsymbol{x}^{(k)}$).
- Compare objective functions: $f(\boldsymbol{z}^{(k)}) < f(\boldsymbol{x}^{(k)})$?
- If f decreased, then jump to $\boldsymbol{z}^{(k)}$; i.e., $\boldsymbol{x}^{(k+1)} = \boldsymbol{z}^{(k)}$.

- If f increased, then toss a coin. If HEAD, then jump to $\boldsymbol{z}^{(k)}$; else, stay at $\boldsymbol{x}^{(k)}$.
- This random jumping allows us to "climb out" of local minima.
- Intuitively, we would want the coin to eventually be biased away from HEADs, so that we "jump around" less and less with time.

Example: "Peaks" function





Simulated annealing algorithm:

1. Initialize: $\boldsymbol{x}^{(0)}$; $k := 0$;
2. Compute next point:
   - Generate $\boldsymbol{z}^{(k)}$;
   - Set $P_k = \exp[-(f(\boldsymbol{z}^{(k)}) - f(\boldsymbol{x}^{(k)}))/T_k]$;
   - If $f(\boldsymbol{z}^{(k)}) < f(\boldsymbol{x}^{(k)})$, then $\boldsymbol{x}^{(k+1)} = \boldsymbol{z}^{(k)}$; else
   $$\boldsymbol{x}^{(k+1)} = \begin{cases} \boldsymbol{z}^{(k)} & \text{with prob. } P_k \\ \boldsymbol{x}^{(k)} & \text{with prob. } 1 - P_k \end{cases}$$
3. $k := k + 1$
4. If stopping criterion satisfied, then STOP;
   else go to step 2.

Generation of new point

- Generate the point $\boldsymbol{z}^{(k)}$ according to a probability distribution that may depend on $\boldsymbol{x}^{(k)}$.
- Typically, this probability distribution is centered around $\boldsymbol{x}^{(k)}$, and covers a "neighborhood" of $\boldsymbol{x}^{(k)}$.
- Example: $\boldsymbol{z}^{(k)}$ is chosen uniformly from a square region
$$\{\boldsymbol{x} : x_i^{(k)} - \delta \leq x_i \leq x_i^{(k)} + \delta\}.$$
- To ensure descent, we may keep track of the best-so-far point.

Temperature schedule

- The sequence $\{T_k\}$ is called the *temperature schedule* (or *cooling schedule*).
- If not chosen appropriately, the algorithm will not converge.
- Sufficient condition for convergence to global minimizer:

$$T_k = \frac{\gamma}{\log(k+1)},$$

where γ is a problem-dependent constant.

Genetic algorithm

- A form of randomized search method.
- Many applications: AI, programming, optimization, neural network training, design problems.
- Underlying idea based on genetics.
- Developed by Holland (approx. 60s–70s).
- Think of it as an organized way of doing random search.

Basic idea (§14.1)

- Consider the usual optimization problem

$$\begin{array}{ll} \text{maximize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \Omega. \end{array}$$

- We start with an initial set of points in Ω, denoted by $P(0)$.
- We call $P(0)$ the *initial population*.
- Evaluate the objective function at points in $P(0)$ (evaluate *fitness*).
- Based on this evaluation, we create a new set of points (population) $P(1)$.
- The creation of $P(1)$ involves certain operations on points in $P(0)$, called *crossover* and *mutation*.
- Repeat the above procedure iteratively, generating populations $P(2), P(3), \ldots$, until an appropriate stopping criterion is reached.

- The goal in each iteration is to create a new population with an average objective function value that is higher than the previous population.
- To fully describe the genetic algorithm, we need some special terminology.

Chromosomes and Representation Schemes

- To use genetic algorithms, we need first to "encode" Ω.
- Encode $\equiv$ map Ω onto a set consisting of strings of symbols, all of equal length.
- The strings are called *chromosomes*.
- Each chromosome consists of elements from a chosen set of symbols, called the *alphabet*.
- The choice of chromosome length, alphabet, and encoding is called the *representation scheme* for the problem.

Examples of representation schemes

Binary representation scheme:

- Alphabet = $\{0, 1\}$
- Chromosome = binary string
- Length of chromosome = number of bits (L)
- Encoding = Binary representation of number

Representation scheme for product design problem:

- Alphabet = available components (parts)
- Chromosome = possible design
- Length of chromosome = number of components in design
- Encoding = representation of design

- Suppose we have chosen a representation scheme.
- To begin, we generate an initial population $P(0)$ of chromosomes (at random).
- To generate a new population $P(1)$, there are two stages:

  - Selection: involves evaluating the fitness of each chromosome in $P(0)$.
  - Evolution: involves applying the crossover and mutation operations.

- We then repeat the process.

- For simplicity, we will only consider binary representation scheme.
- Easy to understand, and contains all the general ideas.
- We only describe the genetic algorithm tailored specially for binary schemes.
- We assume henceforth that the fitness function f is ≥ 0 everywhere.

Selection

- Suppose we are given the population at the kth iteration: $P(k)$.
- We first generate a set $M(k)$ with the same number of elements as $P(k)$, as follows:
  - Each element $\boldsymbol{m}^{(k)} \in M(k)$ is equal to $\boldsymbol{x}^{(k)} \in P(k)$ with probability
$$\frac{f(\boldsymbol{x}^{(k)})}{F(k)},$$
where
$$F(k) = \sum f(\boldsymbol{x}_i^{(k)})$$
and the sum is taken over the whole of $P(k)$.

- "Roulette wheel" selection scheme.
- We call $M(k)$ the *mating pool*.
- In other words, we select chromosomes into the mating pool with probabilities proportional to their fitness.
- There are other ways of doing selection; e.g., tournament scheme.

Evolution

- Suppose we are given the mating pool $M(k)$.
- From $M(k)$, we choose (at random), some pairs of chromosomes (called *parents*).
- Crossover takes a pair of parent chromosomes and gives a pair of *offspring* chromosomes.
- Typically, we exchange substrings of the two parent chromosomes.
- There are many ways to do crossover.

One-point crossover

- We first choose a number randomly between 1 and $L-1$ according to a uniform distribution, where L is the length of chromosomes.
- Call this number the *crossing site*.
- Crossover: exchange substrings of parents to the left of the crossing site.
- Example:
  - Parents: 000000 and 111111 ($L = 6$).
  - Crossing site: 4.
  - Offsprings: 000011 and 111100.

Multi-point crossover

- Can also have crossover operations with multiple crossing sites.
- Each crossing site is generated at random, as before.
- Example: (2-point crossover)
  - Parents: 000000000 and 111111111 ($L = 9$).
  - Crossing sites: 3 and 7.
  - Offsprings: 000111100 and 111000011.
- After the crossover operation, we replace each pair of parents by their offsprings.
- Next, we apply the mutation operation.
- Mutation takes a chromosome and randomly changes its symbols.
- For binary, change symbol $\equiv$ complement bit.
- Example: Given a chromosome 0000000, if we complement the 3rd and 7th bits, we get 0010001.
- We first choose some chromosomes at random.
- For each chromosome, we pick some of the bits at random.
- We then complement the randomly chosen bits.
- After applying crossover and mutation to the mating pool $M(k)$, we obtain the new population $P(k+1)$.

- We then repeat the procedure of evaluation, selection, and evolution, iteratively.
- $P(0) \to M(0) \to P(1) \to M(1) \to P(2) \to M(2) \to \cdots$.

Genetic algorithm

1. Set $k := 0$; form initial population $P(0)$
2. Evaluate $P(k)$
3. If stopping criterion satisfied, then stop
4. Select $M(k)$ from $P(k)$
5. Evolve $M(k)$ to form $P(k+1)$
6. Set $k := k + 1$, go to step 2

- As with previous random search techniques, under certain conditions we can prove convergence to global minimizer (in a probabilistic sense).
- Note that we can apply random search techniques to decision spaces other than $\mathbb{R}^n$, including discrete spaces.

Possible modifications:

- We can change the probability parameters of evolution (crossover and mutation) so that they become less frequent as the evolution progresses.

  Analogous to "cooling" in simulated annealing.
- Use other forms of encoding (e.g., real numbers, etc.). Have to define crossover and mutation appropriately for the given encoding scheme.

Genetic algorithm demo

- Apply genetic algorithm to the usual Rosenbrock and Peaks functions.
- We use the basic ideas from genetic algorithms to do a random search.
- Encoding: usual real number computer representation.
- Crossover: given two parents, we compute the average of them, and then randomly perturb the average to get two offsprings.
- Mutation: randomly perturb given chromosome.

