CIS 630 Lab Optimization Search Algorithms

Due on Oct 26 at class meeting.

Objectives and grading policy.

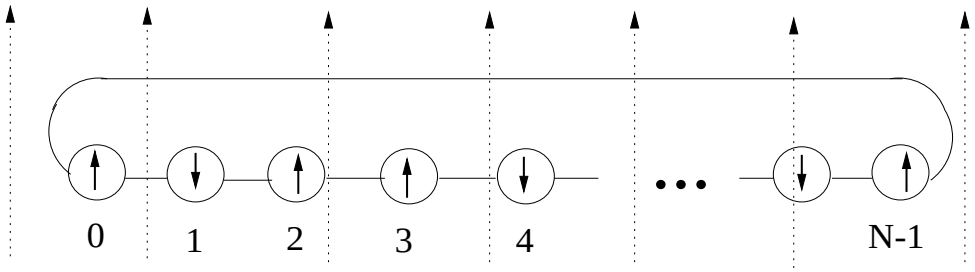
This second lab is designed as an example to walk through the optimization search algorithms. It consists of two parts. One is genetic algorithm which is required for this lab, and the other is a Gibbs sampler which is optional. The Gibbs sampler is an advanced topic in modern search algorithms, you will receive bonus grade by implementing this algorithm. The bonus grade works in the following way. If you compute the right result using Gibbs sampler, your final CIS630 grade will be upgraded by one level. For example, from B to B+, or from A- to A etc. In fact, the code is no more than 80 lines of c code if you do it right !

Background material.

The figure below shows a physical system called the **Ising model**, named after a German physicist. Ising model is a classic mathematical model in statistical physics, and it holds the key to understanding many complex systems as well as to the design and analysis of stochastic algorithms.

Consider a physical system with N spins arranged in a ring. Each spin has two states: upward or downward representing the directions of its magnetic field decided by the clockwise or counter clockwise spinning of electrons. The state of the system is described by a N-vector

$$s = (x_0, x_1, ..., x_{N-1}), \quad x_i \in \{-1, 1\}\ \forall i.$$



In real world case, one can imagine an 2D matrix of spins or 3D volume of spins for magnets. To simplify the problem, let's consider this 1D example for convenience.

The energy of this physical system is decided by two factors. One is the interactions between two neighboring spins x_i and x_{i+1} for all $i = 0, ..., N-1$. If the two spins have the same directions ($x_i = x_{i+1}$) then the system has a relative lower energy. Otherwise if

two neighboring spins have different directions, then the system has higher energy, and we say the magnet has a "crack" at position i. The second factor is the interaction of each spin with the external magnetic field. Without loss of generality, we assume the external magnetic field is upwards in our example (see the figure above). Thus the system has lower energy if the spins are oriented along with the external field. The total energy is the summation of the two factors over all spins.

$$E(s) = -\alpha \sum_{i=0}^{N-1} x_i \cdot x_{i+1} - \beta \sum_{i=0}^{N-1} x_i,$$

where $x_N = x_0$, $\alpha = 0.5, \beta = 0.05$. To make this clearer, if x_i and x_{i+1} have the same signs, then the product $x_i \cdot x_{i+1} = 1$, and thus the system has low energy. Otherwise the product is -1 and system has high energy if they have opposite signs.

We further define a *fitness* function for the system.

$$f(s) = e^{-E(s)}, \quad s \in \Omega = \{-1, 1\}^N.$$

Obviously the lower the energy, the higher the fitness function.

It is not so hard to realize that the lowest energy state is $s^* = (1, 1, 1, ..., 1)$. However, a computer doesn't have this intuition, and it has to do an optimization search. As we define before, the goal of an optimization search algorithm is to find the state s^* with lowest energy (globally optimal) in the configuration space $\Omega = \{-1, 1\}^N$.

$$s^* = \arg\max_{s \in \Omega} f(s).$$

This is really hard, because the search space Ω can be large, imagine $N = 10^{23}$ in real physical systems ! Just remind you: in comparison with the blind and heuristic search, each state s is assigned a fitness function, and we may not be able to find the globally optimal state s^*. We may only find a nearly optimal one.

Now, our goal is to design computer algorithm that can effectively search for lower energy states. As a matter of fact, it is reasonable to think that our computer program is "simulating" the physical process. Because the states of a physical system is dynamically changing overtime to achieve some low energy, and thus stable states!

Experiment 1, Genetic algorithm (required) .

Write a program using your familiar language (C, C++, matlab) to implement the Genetic algorithm as it is stated in the handout and the lecture. Choose $N = 64$ spins, and keep $n = 8$ genotypes (chromosome) for each generation. Note that the handout example has 4 bits for each of the 4 genotypes, and we use 64 bits for each of the 8 genotypes.

Let me briefly overview the algorithm. It consists of three steps to go from generation m to generation $m + 1$: reproduction, crossover, and mutation.

Initialization, you select $n = 8$ genotypes at random: each genotype has $N = 64$ spins and each spin takes value -1 or 1 at even chance. Thus you get generation $m = 0$:

$$G_0 = \{s_1, s_2, ..., s_n\}.$$

Then repeat the following steps for going from generation m to generation $m + 1$.

1. Reproduction. Draw n genotypes from the set G_m. Each genotype s_i in G_m has a chance to be chosen:

$$\frac{f(s_i)}{f(s_1) + f(s_2) + \cdots + f(s_n)}$$

Obviously, genotypes (or systems) with lower energies have higher chances to reproduce themselves. Drawing from G_M n times with a replacement sampling, we obtain a new set

$$g_1 = \{s_1^{(1)}, s_2^{(1)}, ..., s_n^{(1)}\}.$$

Some genotypes may repeat many times in g_1 and some may not present.

2. Cross-over. from the set g_1, we pair each two genotypes $s_{2i+1}^{(1)}$ and $s_{2i+2}^{(1)}$ for $i = 1, 2, ..., n/2$. Each pair has $q = 0.6$ (60 percent chance) probability to exchange genes, with 40% chance to remain unchanged. If it decides to exchange genes, you draw two random positions $b_s, b_e \in \{0, ..., N - 1\}$ for the starting and ending break points, and do the crossover. Thus you obtain a new set

$$g_2 = \{s_1^{(2)}, s_2^{(2)}, ..., s_n^{(2)}\}.$$

3. Mutation. Then for each spin x_i in each genotype at g_2 (we have total $N \times n$ spins), it has a $\gamma = 0.03$ probability (3% chance) to mutate, i.e. to flip its sign.

Then you obtain a new set called generation G_{m+1}. Repeat the three steps

1. Try $M = 64$ generations. For each generation, output the best genotype (print out a 0/1 sequence with o for -1 and 1 for 1). Then show the two performance curves:

the fitness function of the best state s and the average fitness function. Plot the logarithm of the fitness function $\log p(s) = -E(s)$ against the generation $1, 2, 3, ...M$ as it shows in the handout.

2. Run a second experiment and plot the two performance curves for $M = 128, n = 16, q = 0.6, \gamma = 0.03$. (no output for the genotype this time).

3. Run a third experiment for $M = 128, n = 32, q = 0.8, \gamma = 0.03$. Plot the two performance curves (no output for the genotype this time).

4. Run the 4th experiment for $M = 128, n = 32, q = 0.8, \gamma = 0.2$

Explain the differences between the four groups of experiments. You need to write about 100 words to compare the experiments, and the influence of parameters M, n, q, γ.

Hand in your plots and a hardcopy of your code.

Experiment 2, Gibbs sampler and simulated annealing (optional).

Suppose we normalize the fitness function so that $\sum_{s\in\Omega} p(s) = 1.0$ and introduce the temperature parameter T, thus

$$p(s;T) = \frac{1}{Z(T)} e^{-E(s)/T}$$

where $Z(T)$ is a normalization factor which is a function depending on T.

$$Z(T) = \sum_{s\in\Omega} e^{-E(s)/T}.$$

Obviously, $p(s;T)$ is a probability distribution over Ω. Note that we don't need to compute this normalization constant $Z(T)$. Also both the probability p and the constant Z depend on the temperature T.

Now, let's overview a widely used stochastic search algorithm, called the Gibbs sampler. The Gibbs sampler simulates a random walk in the space Ω. It starts with an arbitrary state s_0, and thus generate a sequence over time

$$s_n, s_{n+1}, s_{n+2}, ..., s_{n+t}.$$

Note that we only work with one string of spins each time instead of a population.

After a "burn-in" period n, somebody has proven a theorem for us that is **the sampler visits the state** $s \in \Omega$ **according to** $p(s;T)$ – what a useful conclusion!

At each iteration, suppose s_t is the current state, the Gibbs sampler does the following.

1. It picks up a spin $x_i, i \in \{0, N-1\}$ at random. That means, each spin has equal chance to be chosen.

2. Given the spins $x_j, j \neq i$, compute the conditional probabilities for $x_i = -1$ and $x_i = 1$, denoted by p_{-1}, p_{+1} respectively. $(p_{-1} + p_{+1} = 1.0)$ [You fixed all the other spins except x_i.

3. Draw a random number $d \in [0, 1]$, and set $x_i = -1$ if $d < p_{-1}$. Set $x_i = 1$ otherwise.

4. repeat 1-2-3 N times, and this is called one **sweep**.

One sweep is like one generation in the Genetic algorithm.

Answer the Following Questions.

- State the nature of $p(s;T)$ for two extreme cases: $T = \infty$ and $T = 0$. That is, describe the probability $p(s;T)$ for how the probability mass is distributed among the 2^N states in the two extreme temperatures.
- Explain the purpose of "simulated annealing" (reducing T from a very high temperature to a very low temperature), given your answer to the above question and the property of the Gibbs sampler. Your answer should be less than 60 words. Can the Gibbs sampler find S^* eventually if we play with a simulated annealing strategy? Explain why?
- Choose $N = 64$, Implement the Gibbs sampler for 250 sweeps. For the first 50 sweeps, set $T = 4$. For sweeps (50,100), set $T = 3$. Sweeps (100,150) set $T = 2$. Sweeps (150,200) set $T = 1$. Sweeps (200, 250) set $T = 0.5$.
- Plot the logarithm of the fitness function $-E(s_t)$ for a state s_t at the end of sweep $t = 1, 2, .., 250$.

The above Gibbs sampler is about the same computational complexity as the genetic algorithm, try to compare their performances.

In fact, you can play with the annealing scheme (decide how to low the temperature), soon you can reach the optimal state $s^* = (1, 1, 1, ..., 1)$ — the pure crystal.