Stochastic optimization of an objective function to build a 3D chromosomal model

Caiwei Wang, Xiaokai Qian, Sean Lander,
Haipei Fan, Puneet Gaddam, Brett Koonce

University of Missouri - Columbia

May 7, 2014

Abstract

First, we construct a congugate gradient (CG) descent searcher to build a 3D model of a human chromosome using the method advanced by Trieu and Cheng in a recent paper. Next, we utilize two different stochastic methods, simulated annealing (SA) and Markov-Chain Monte-Carlo (MCMC), to duplicate our CG results. Then, we build a PDB model of the final chromosome and visualize the process of solving the objective function. Ultimately, we compare our results with other research in this domain, discuss difficulties we encountered and potential future improvements that could be made.

1 Introduction

The human genome has been sequenced, but modelling the human chromosome remains a difficult task. One promising avenue of modern research lies in the technique of Hi-C sequencing, which produces contact maps as part of the sequencing process. By analyzing which regions of the genome are often found together, we can build up a map of the larger genome. Trieu and Cheng recently proposed a new method based on an objective function that promises to be able to build a model using gradient descent from the Hi-C data. We decided to put their results to the test, as well as experiment with solving the objective function using stochastic (MCMC and SA) methods.

2 Overview

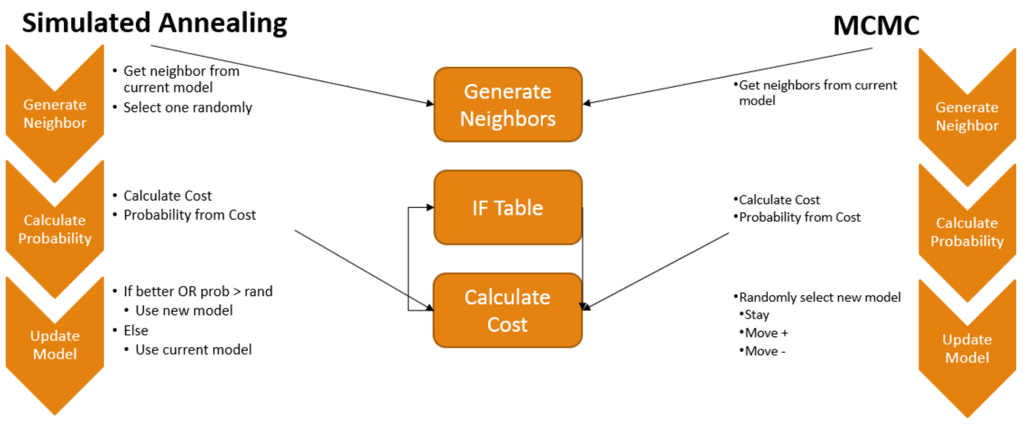

Figure 1: Architecture Summary

We do not have a complicated pipeline for this project. Essentially, we take input data, clean it up slightly, and apply one of three different methods: CG, SA, and MCMC.

3 Dataset

We obtained a megabase resolution fragment database (157x157, dropping first/last residues from Chromosome 7) from Trieu to work with. We set all entries less than 0.66 to 0, as per the same treatment in the original paper. We also made a list of the constants they used, so that our results would (hopefully) match.

4 Objective function

Simply put, the objective function advanced by Trieu and Cheng can be expressed as follows:

$$TotalModelScore(m) = ContactScore(m) + NonContactScore(m) + PairSmoothing(m) \tag{1}$$

The first operation minimizes the distances of contacts with affinity to keep them in contact (but keeps their distance above a minimum threshold):

$$ContactScore(m) = \sum_{i=1}^{n}\sum_{j=1}^{n}(|i-j|>1)*(W_1*\tanh(d_c^2-d_{ij}^2)*N_{ij}+W_2*\tanh(d_{ij}^2-d_{min}^2)) \tag{2}$$

The second operation maximizes the distances of contacts without affinity to keep them away from contact (but keeps their distance below a maximum threshold):

$$NonContactScore(m) = \sum_{i=1}^{n}\sum_{j=1}^{n}(|i-j| > 1) * (W_3 * \tanh(d_{max}^2 - d_{ij}^2)/TotalIF + W_4 * \tanh(d_{ij}^2 - d_c^2)/TotalIF) \quad (3)$$

The third operation tweaks the scores of consecutive contacts slightly so that moving them is favored over adjusting more distant relationships:

$$PairSmoothing(m) = \sum_{i=1}^{n}\sum_{j=1}^{n}(|i-j| = 1) * (W_1 * IF_{max}/TotalIF * \tanh(da_{max}^2 - d_{ij}^2) + W_2 * \tanh(d_{ij}^2 - d_{min}^2)/TotalIF) \quad (4)$$

We also used the following table of constants from Trieu and Cheng's paper. They are experimentally derived values.

Constants

d_{min}	$\sqrt{0.2}$	W1	1.0
d_{max}	4.5	W2	1.5
d_c	$\sqrt{7.0}$	W3	1.5
da_{max}	$\sqrt{1.8}$	W4	1.5

We also make use of the following:

$$TotalIF = \sum_{i=1}^{n}\sum_{j=1}^{n} IF(i,j) \quad (5)$$

$$N(i,j) = IF(i,j)/TotalIF \quad (6)$$

$$IF_{max} = \max(IF(i,j), IF(j,i)) \quad (7)$$

Combining the above equations, we end up with a combined objective function of approximately 50000 subproblems, as detailed above. Our problem now is simply to maximize the result given a 471 variable input vector (157 contacts * 3 coordinates (x, y, z)) across this space. Towards this end, we present one deterministic and two stochastic methods of optimization. For each version, our initial solution subspace is randomly chosen coordinates in the domain $-0.5 \leq (x, y, z) \leq 0.5$.

5 Conjugate Gradient Descent

Our first method we applied is standard gradient descent. We utilized a solver based upon the Powell conjugate method. This allowed us to avoid calculating a derivative function. Instead, it maintains a list of search vectors and attempts a bidirectional search along each one. Each iteration, the highest performing vector is replaced by the best new search vector and then the process repeats. This method is easy to implement, but takes some time to run.

6 Simulated Annealing

For our first test of stochastic methods, we built a simulated annealer capable of solving our objective function. SA is a well known non-deterministic algorithm to sample a state space quickly. Essentially, we start from a random model, then generate a neighboring model (our original model with a modification). We score the two models. If the new one is better, we accept it, otherwise we only replace our existing model based on a gradually decreasing schedule (temperature function). As such, the two most important parts of implementing simulated annealing are the temperature function and how we generate neighbor models.

We tested the following two temperature functions, one sigmoid and one linear:

$$T_s(e,t,k) = -t * 2/(1 + \exp(-5 * k/t)) + t * 2 \tag{8}$$

$$T_l(e,t,k) = (-t/e) * k + t \tag{9}$$

For a neighborhood function, we select a contact at random. Next, we generate a random offset, scaled using the known minimum contact distance: $d_c/2*(x, y, z)$. Then, we score the two models using the method described previously. For our simulated annealing schedule, we ended up using one million simulations (epochs) over a temperature range of 250 degrees.

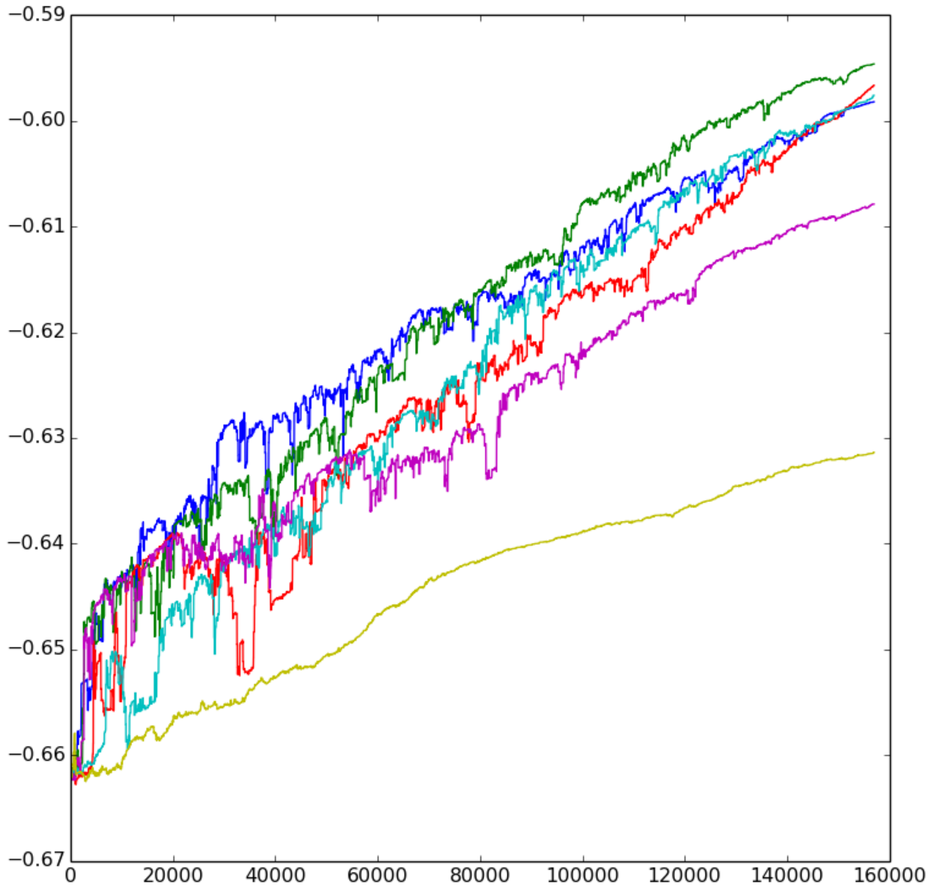

Figure 2: A graph of score versus number of simulations for SA: -n6-e157000-t250

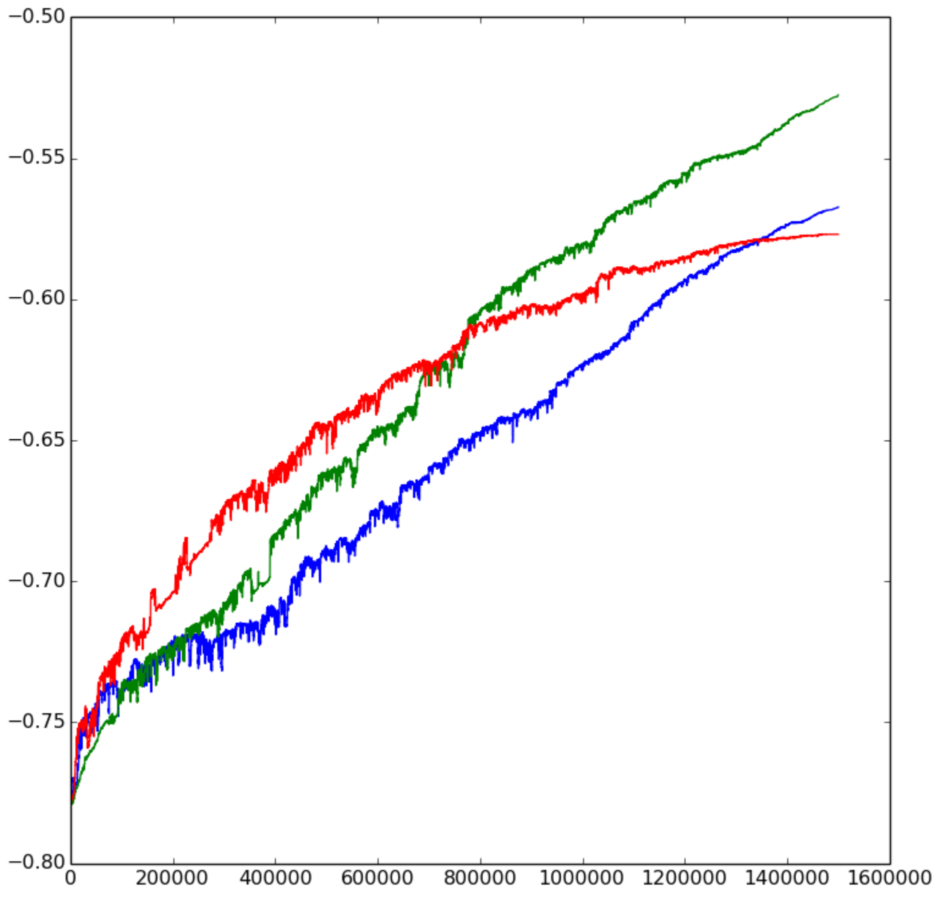

Figure 3: A graph of score versus number of simulations for SA: -n3-e1500000-t250

7 Markov-Chain Monte-Carlo

For our second test of stochastic methods, we built a Markov-Chain Monte Carlo sampler. Markov chains are an important technique in computer science from statistics. First, we build up a graph model of discreet events with a corresponding transition matrix. Next, we seed the transition matrix using real-world data (generally derived from posterior probability distributions). Then, we can sample the Markov model repeatedly (Monte Carlo) to try and converge upon a solution space that is generally an excellent model of our problem domain.

For our project, we utilized MCMC to optimize our state/input vector (coordinate data) to most closely approximate our posteior distribution (our objective function). Our methods converged well, but required a large number of simulations to mix (over one million simulations for the larger residues). We utilized a different neighborhood function than we used in Simulated Annealing. First, we select a random region from which to generate a neighborhood. The neighborhood is then created by increasing/decreasing the location in each of the three dimensions: (x, y, z). In this way we generate 8 neighbors and a default position which are scored and given a probability of selection based on the score range, with higher scoring conformations more likely to be sampled. Once a new neighbor conformation is selected the process repeats until all iterations have completed.

We tested two different versions of MCMC: a greedy and non-greedy selector:

$$T_{greedy} = (score - minScore)/scoreRange \tag{10}$$

$$T_{non-greedy} = (score - minScore)/(scoreSum - minScore * neighborCount) \tag{11}$$

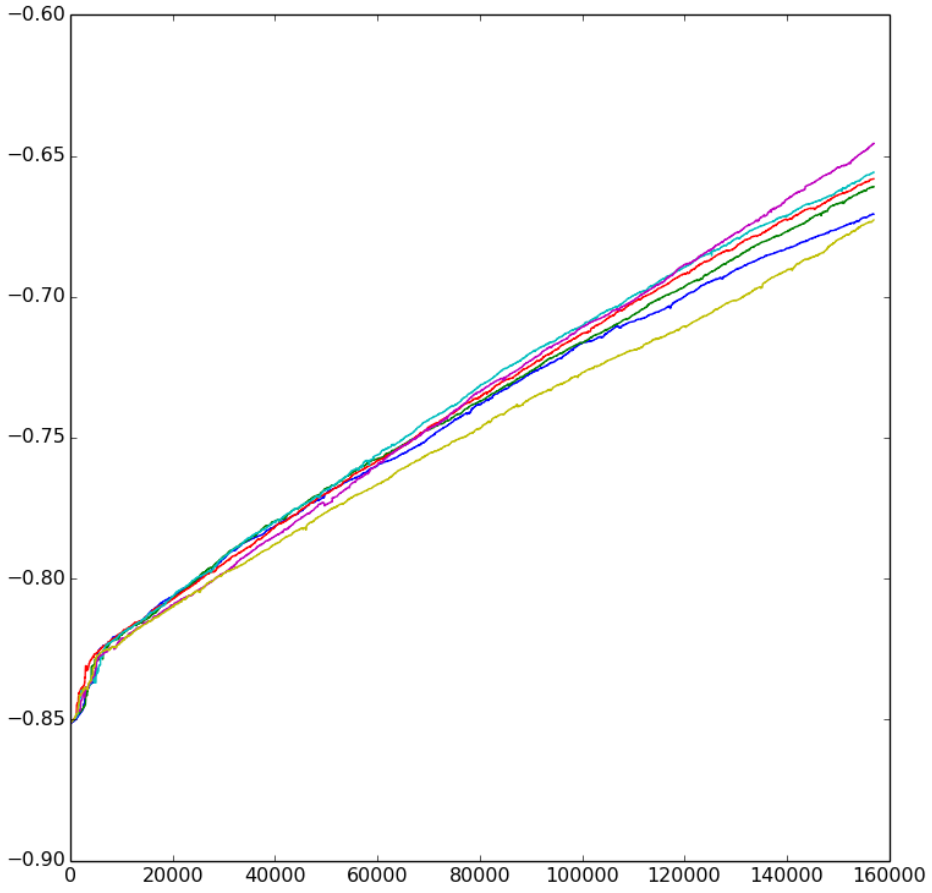

Figure 4: A graph of mixing versus number of simulations for MCMC: -n6-e157000

8 Results

For testing purposes, we wrote tools to enable us to work on a subset of the final data (ten resiudes, for example). This in turn allowed us to make sure that all of our methods converged to the same results and provided an objective benchmark.

Generally speaking, we found that:

1. Our conjugate gradient solver was not very fast, but provided accurate results. Our usage of the Powell method, while easy to implement, was not very fair to this technique. Since we know the objective function, we should have been able to build a derivative function by hand and in turn make this method much faster.

2. When utilizing an external solver, simulated annealing converged quickly and rapidly to a solution (approx. 5 minutes for a 157 residue chain). Our custom SA solver was slower (approx. 600k-1.5 million simulations required). We attribute this to that our neighborhood function did not change the state space very much and as a result the entire process required a lot more simulation time. We ended up just using the linear temperature model and did not find that the sigmoid model offered much improvement.

3. We found that MCMC mixed (and found a solution) on our problem for smaller residue tests, but for the larger problem performance was too limited for us to recommend this technique going forward (we estimate >1 million simulations are needed for a full chain). The table below is not entirely fair to MCMC, because our neighborhood function generates eight neighbors for each cycle. In other words, the two methods are doing roughly the same amount of math, the problem is simply that MCMC does not converge as quickly as would be desired.

Iterations/minute

SA	10,000
MCMC: greedy	1750
MCMC: non-greedy	1700

9 Citations

We thank the following tools and papers:

Tuan Trieu and Jianlin Cheng. Large-scale reconstruction of 3D structures of human chromosomes from chromosomal contact data. Nucl. Acids Res. first published online January 24, 2014. doi:10.1093/nar/gkt1411