A Simulated Annealing Algorithm for DNA Loci Selection

Michael A. Alcorn

Background

For my project, I developed a simulated annealing algorithm that selects loci (locations in DNA) from the output of a high-throughput DNA sequencer for use in a genetic analysis. This problem was inspired by my girlfriend, Katherine SIlliman, who is a PhD student in evolutionary biology studying oyster population genetics at The University of Chicago, though this particular problem concerns shrimp. RAD sequencing is a high-throughput DNA sequencing technique that has gained popularity among population genetics researchers. The technique uses restriction site associated DNA markers (hence, RAD) to cut the DNA of an organism into small pieces, which can then be used for genome assembly and/or statistical analyses. Because the RAD enzyme is looking for a specific short sequence of DNA to make a cut, different individuals may have different cut site locations due to genetic mutations. As a result of this phenomenon, different loci can be found for different individuals.

When performing genetic analyses, researchers attempt to (***A***) include as many loci as possible and (***B***) ensure that each locus includes as many of the study individuals as possible in order to maximize the study's statistical power. With these goals in mind, we can restate the loci selection problem as an optimization problem where each state represents a different subset of the available loci. Because there are on the order of 10^5 different loci for a given high-throughput sequencing output, there are on the order of $2^{10,000}$ different possible subsets of loci for study. Clearly, exhaustively searching the entire search space would be impossible, so other global optimization methods must be considered. Stacks (http://creskolab.uoregon.edu/stacks/), a popular genetic analysis software, uses simple thresholding to select loci (e.g., by selecting all loci with at least *x* species and at least *y*% of individuals) and, as a result, may be omitting more informative loci from analyses.

Simulated annealing is a global optimization algorithm that uses a “cooling schedule” to probabilistically explore a search space. The algorithm has been successfully applied in a number of different settings, from biology, to physics, to finance [Ingber, 1993] and possesses properties that make it an attractive candidate for the loci selection problem. Here, I present a simulated annealing algorithm for loci selection that features a tunable scoring function. My hope is that, by running this algorithm on the output of a high-throughput sequencer, scientists will be able to choose those loci that will result in the best science possible.

Simulated Annealing

Simulated annealing takes its inspiration from metallurgy where “annealing” is a technique that uses controlled temperature changes to reduce the number of defects in a given material. In the case of simulated annealing, changes in “temperature” are used to discover good solutions to a problem. The goal of any optimization problem is to find a solution that maximizes (or minimizes) some function. Because search spaces are typically non-convex, they often contain many different local optima (**Figure 1**). These local optima inhibit the ability of greedy search algorithms to find the global optimum of any reasonably complex problem. In fact, the local optimum found by greedy search algorithms will often be a rather poor solution.

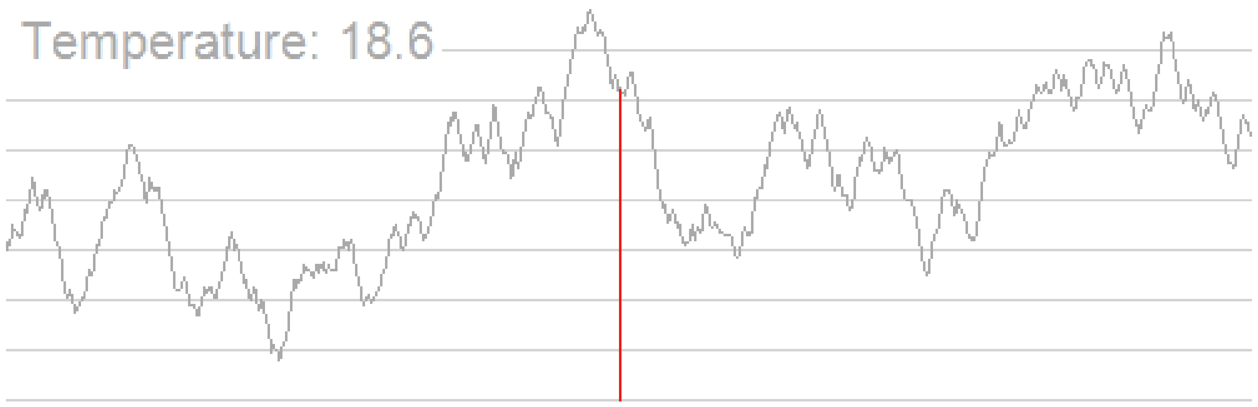

Figure 1. A search space where the x-axis represents different states and the y-axis represents the score for a given state. Notice how there are many local optima, but only a single global optimum. Greedy search methods generally fail to find the global optimum

because they will only climb their local hills. Simulated annealing allows “downhill” movements in the early stages of the algorithm, which allows it to explore a larger search space. This figure was taken from the Wikipedia article on simulated annealing (http://en.wikipedia.org/wiki/Simulated_annealing).

In simulated annealing, the algorithm begins at a “high temperature”, which allows the algorithm to move to states with lower scores in the early stages. As the temperature cools, the algorithm becomes more conservative in its exploration, increasingly preferring only those candidate states that increase the score being optimized.

Pseudocode for the simulated annealing algorithm can be found in **Figure 2**. Here s_0 is the initial state, s is the current state, and s_{best} is the best state discovered. In the context of the loci selection problem, each state represents a specific subset of loci being considered for analysis. iters is the maximum number of iterations to run the algorithm. $temp_{max}$ is the initial temperature of the algorithm and cool is the cooling rate (a number between 0 and 1).

The algorithm begins by initializing the best state and the best score. In each iteration of the loop, a candidate state is selected from the neighborhood of the current state. The score (described in the next section) is then calculated for the candidate state and the temperature is reduced by the cooling rate. The difference between the candidate’s score and the current state’s score is calculated and this difference is used to calculate the acceptance probability for the candidate state. If the candidate’s score is greater than the current state’s score, the acceptance probability is set to 1.0 (i.e., candidate states that are better than the current state are always accepted). If the candidate’s score is less than the current state’s score, than the candidate is accepted with probability $e^{diff\ /\ temp}$. As you can see, hotter temperatures (larger denominators) effectively reduce the algorithm to a random walk while cooler temperatures (smaller denominators) cause the algorithm to resemble a greedy approach. If the candidate’s score is greater than the best score discovered so far, the best score and the best state are updated accordingly.

```
SimulatedAnnealing($s_0$, iters, $temp_{max}$, cool)
    1. s ← $s_0$
    2. $s_{best}$ ← s
    3. score ← getScore(s)
    4. $score_{best}$ ← score
    5. temp ← $temp_{max}$
    6. for i ← 1 to iters
        a. $s_{cand}$ ← neighbor(s)
        b. $score_{cand}$ ← getScore($s_{cand}$)
        c. temp ← cool x temp
        d. diff ← $score_{cand}$ - score
        e. accept ← min(1.0, $e^{diff / temp}$)
        f. if random(0, 1) < accept
            i. s ← $s_{cand}$
            ii. score ← $score_{cand}$
            iii. if score > $score_{best}$
                1. $score_{best}$ ← score
                2. $s_{best}$ ← s
    7. return $s_{best}$
```
Figure 2. Pseudocode for the simulated annealing algorithm.

Scoring Function

All optimization algorithms require meaningful scoring functions in order to find valuable solutions. Here, I define a scoring function that can be tuned by the user to favor different properties of a data set being prepared for analysis. There are three separate components to consider when assessing the quality of the loci to be included in an analysis: ***(1)*** the total number of loci, ***(2)*** the amount of missing data, and ***(3)*** the number of species represented at each locus. As a result, we can define the score function as:

$$score = w_1 \times loci + w_2 \times missing + w_3 \times species$$

where

$$loci = \frac{number\ of\ loci\ in\ subset}{total\ loci}$$

$$missing = 1 - \frac{missing\ data\ points\ in\ subset}{(number\ of\ loci\ in\ subset) \times (number\ of\ individuals)}$$

$$species = \frac{\sum_{locus} \sum_{species} I_{present}(locus, species)}{(number\ of\ loci\ in\ subset) \times (total\ species)}$$

$$1 = w_1 + w_2 + w_3 \text{ and } w_i \geq 0$$

Each of the component scores can range from 0 to 1 with 0 representing the worst-case scenario for a data set and 1 representing the best-case scenario. That is to say, when holding the other component scores constant, more loci is better than fewer loci, less missing data is better than more missing data, and more species representation is better than less species representation. Because the weights must sum to one, the final score will also range from 0 to 1, with 0 being the worst score possible and 1 being the best score possible.

Experiments

I tested the algorithm on a data set of shrimp loci provided by Katherine SIlliman. The raw data included 159,751 samples representing 19,420 loci and 24 different individuals from six different species. The data set was subsequently reduced to 1,242 loci based on the requirement that each locus include at least four species. The simulated annealing algorithm was run 21 times with different parameter settings for each experiment **Table 1**. init refers to the proportion of loci randomly included in the initial state. For example, an init of 0.50 would result in the initial state including a random subset of approximately 621 loci. lociW, dataW, and speciesW represent the weights for the loci, missing data, and species score components, respectively. The initial temperature and cooling rate were 0.1 and 0.999, respectively, for all experiments, and a run was terminated when the algorithm reached a temperature of 0.00001. The score, proportion of possible loci included, proportion of missing data, and average proportion of species represented per locus were recorded for the best state discovered in each experiment (**Table 1**). In all, the experiments took 40.3 minutes to run.

Table 1. The results of 21 different experimental runs of the simulated annealing algorithm.

init	lociW	dataW	speciesW	score	loci	data	species
0.50	0.33	0.33	0.33	0.72	0.96	0.47	0.71
0.50	0.80	0.10	0.10	0.90	0.97	0.47	0.71
0.50	0.10	0.80	0.10	0.60	0.31	0.62	0.79
0.50	0.10	0.10	0.80	0.74	0.30	0.57	0.82
0.50	0.40	0.40	0.20	0.72	0.97	0.47	0.71
0.50	0.40	0.20	0.40	0.77	0.97	0.47	0.71
0.50	0.20	0.40	0.40	0.66	0.92	0.47	0.72
0.25	0.33	0.33	0.33	0.71	0.95	0.47	0.72
0.25	0.80	0.10	0.10	0.91	0.99	0.47	0.71
0.25	0.10	0.80	0.10	0.60	0.33	0.61	0.78
0.25	0.10	0.10	0.80	0.73	0.33	0.56	0.80
0.25	0.40	0.40	0.20	0.72	0.97	0.47	0.71
0.25	0.40	0.20	0.40	0.76	0.96	0.47	0.71
0.25	0.20	0.40	0.40	0.66	0.92	0.47	0.72
0.75	0.33	0.33	0.33	0.71	0.95	0.47	0.72
0.75	0.80	0.10	0.10	0.91	0.99	0.47	0.71
0.75	0.10	0.80	0.10	0.60	0.35	0.61	0.78
0.75	0.10	0.10	0.80	0.74	0.31	0.56	0.82
0.75	0.40	0.40	0.20	0.72	0.96	0.47	0.72
0.75	0.40	0.20	0.40	0.77	0.97	0.47	0.71
0.75	0.20	0.40	0.40	0.66	0.91	0.47	0.72

As you can see, in 15 out of the 21 experiments, the algorithm pushed to include all the available loci in the analysis. This result is a byproduct of specific properties of the data set. In general, drastically different subsets of loci produced only relatively small changes to either the missing data or species score components. As a result, the algorithm focused on optimizing the number of included loci whenever the weighting scheme gave considerable weight to the loci score component. Weighting schemes that gave a high relative weight to the data score component seemed to produce the most interesting results as neither the loci score component nor the species score component seemed to particularly suffer as the data score component improved.

Conclusion

In conclusion, simulated annealing appears to be a viable technique for addressing the loci selection problem. Further work should focus on tuning the parameters to various data sets and different desired properties of the resultant data.