Single Particle Tracking in E. Coli Cells

Authors: Stella Stylianidou, Phil Snyder

Names and Roles

Stella – Wrote most of the finished product, including this report! Merging, splitting, integration with the simulated annealing code, other miscellaneous helper functions. She understands this exact implementation the best.

Phil – Mostly debugged and wrote a lot of functions that didn’t make it into the final commit. Wrote functions for generating and testing random time series that resemble real-world data. Also wrote a nearest neighbor implementation (but wasn’t simulated annealing, so we ended up not using it).

What the program is supposed to do

Biology has been seeing great advances in the last few years, with being able to tag proteins in live cells and observe them during the lifecycle of a cell. We can take pictures of those proteins and observe where they are found in a every frame. A main problem is to able to link those different particles to figure out the tracks they took during the duration of the experiment. There are several difficulties with this: if there are a lot of proteins it is hard to find from which one they came from in the previous frame, the protein may split into two, or a protein may merge with another protein, the proteins may disappear and reappear in a frame.

Our program receives real data of the positions of proteins in E.coli cells in each time frame. It starts with random tracks and then modifies the tracks of the proteins in each frame with proteins observed in the next frame attempting to create the track. It then attempts to add splits and merges to the tracks to all the possible places were that can happen (ends and starts of tracks), and it decides whether that is a good choice or not according to the cost.

Technique: Simulated Anneal

Our program uses simulated anneal to identify whether a neighbor state should be accepted or not. The way this works is the following

1) It starts with some random connections for the each protein in each time frame with another protein in the next time frame. This random connections are just the way the data were inserted in the program.

2) It then calculates neighbor states.

a) For the first half of the iterations the program atempts to optimize the paths. It applies the following operator to the current state to find a neighbor state: the positions of jumps are found (with distance more than the MAX\_JUMP variable). and one of those big jumps and the tracks at which it occurs is selected randomly. Another track that has a big jump at the same time is selected. The new state is created by switching the connections of the two tracks at the time where a jump is found.

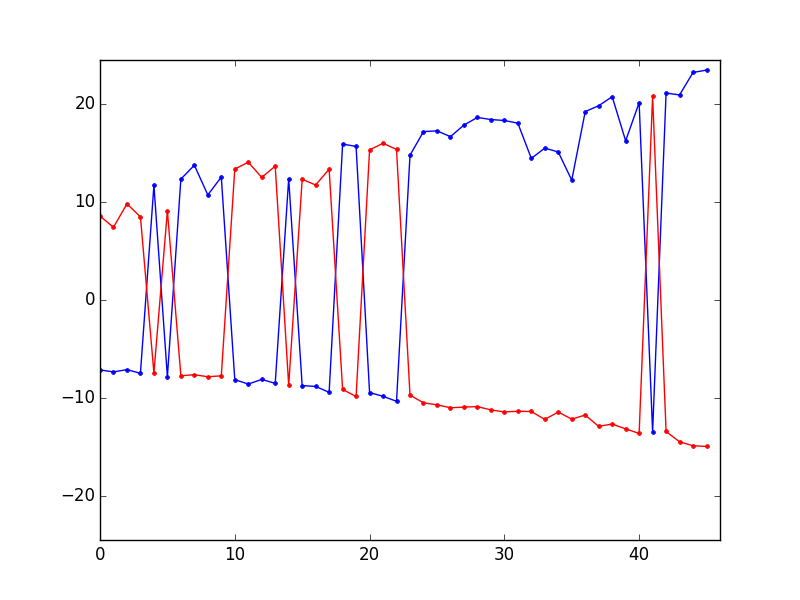
b) For the second part it finds tracks that start after 0 or end before the lifetime of the cell. It randomly selects one of those tracks and it then either attempts to connect the start to a random other tracks (create a split) or merge the ends to a random other track that is alive at that time.

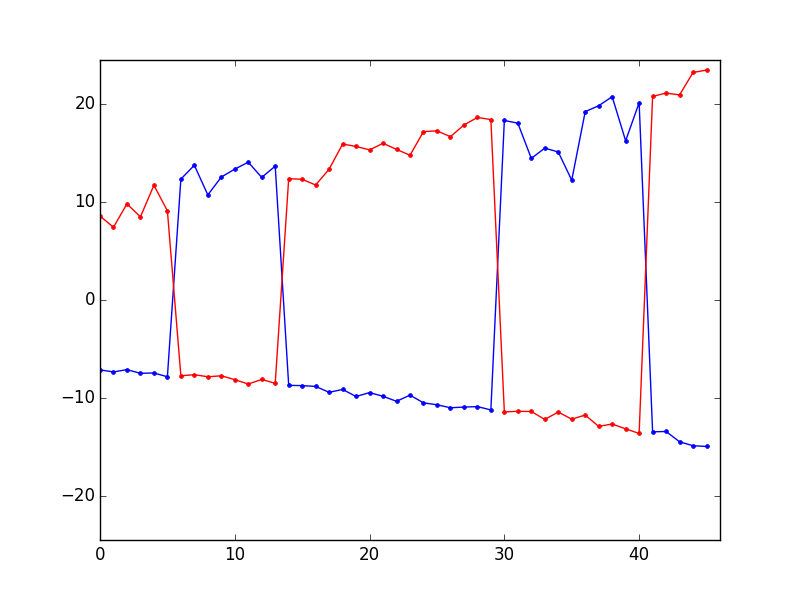
4) If the new state is better than the old state accept the new state

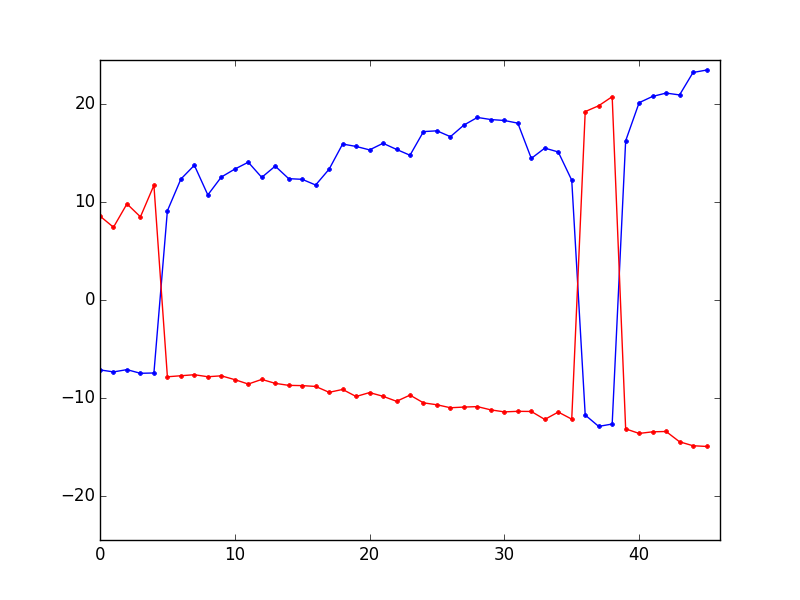
5) If the new state is worse accept it with probability exp (cost(old\_state) – cost(new\_state)/T). A higher temperature makes it more likely to accept an inferior state.

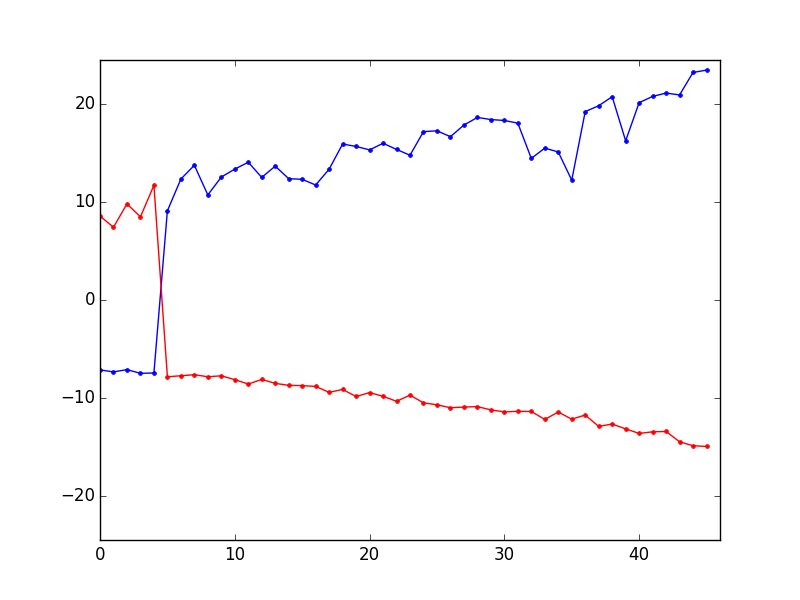
6) Returns to step 2 and lowers the temperature at every iteration

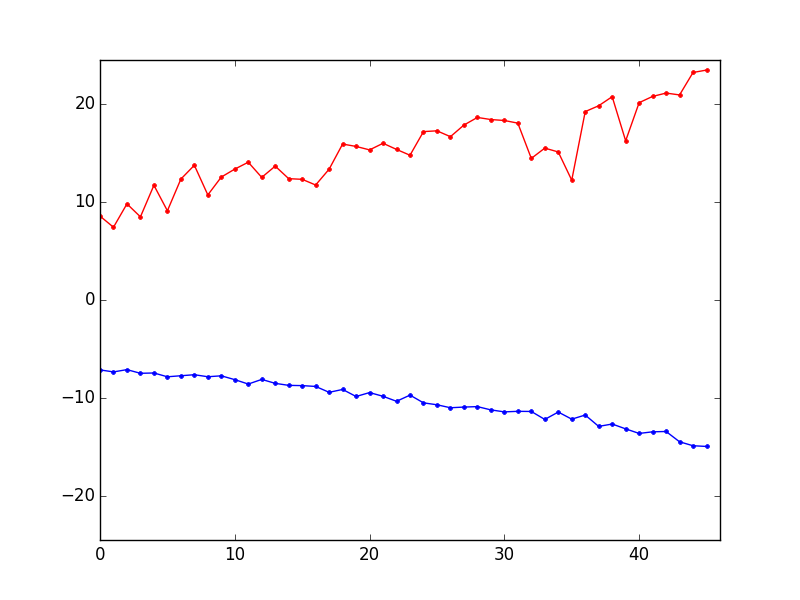
Interesting sample session











Demo instructions

1. use run (filename) where filename is one of the sample matlab files provided (see sample call at bottom of tracking.py)

2. the program should just start running and showing the process of the simulated anneal as it runs

3. to optimize you can modify the variables, and also modify the temperature and annealing schedule:

MIN\_SCORE : the minimum intensity of proteins that will be included in the calculations  
ALLOW\_SPLITS : to allow splitsBIRTH\_PENALTY : cost penalty for having birth of tracks after time 0  
DEATH\_PENALTY : cost penalty for having death of tracks before end of lifetime  
ALLOW\_MERGES : to allow mergesMAX\_TIME\_WINDOW ?  
MAX\_JUMP : the distance at which it will be considered a jump and thus considered to be changed in the neighbor algorithm

Code Excerpt

*# simulated anneal function  
# main simulated annealing algorithm***def** sim\_anneal(state,splits,merges):  
 old\_cost = cost(state,splits,merges)  
 T = 10.0  
 T\_min = 0.01  
 alpha = 0.97  
 iterations = 500  
 old\_cost\_plot = []  
 new\_cost\_plot = []  
 **while** T > T\_min:  
 i = 1  
 **while** i <= iterations:  
 **if** i < iterations/2 :  
 [new\_state,new\_splits,new\_merges] = neighbor\_switch\_jumps(state,splits,merges)  
 **else**:  
 new\_state,new\_splits,new\_merges = neighbor\_merge\_split(state,splits,merges)  
 new\_cost = cost(new\_state,new\_splits,new\_merges)  
 ap = acceptance\_probability(old\_cost, new\_cost, T)  
 print(**'new cost: '** +str(new\_cost) +**'vs old cost: '**+ str(old\_cost))  
 **if** ap > random.random():  
 print(**'accepted'**)  
 state = deepcopy(new\_state)  
 splits=new\_splits  
 merges=new\_merges  
 plot(state,splits,merges)  
 old\_cost = new\_cost  
 i += 1  
 T = T \* alpha  
  
 **return** state,splits,merges,old\_cost

Brief Description of what we learnt

Phil : Before working on this project I didn’t appreciate how generally applicable simulated annealing is. Whatever gradient descent can do, simulated annealing can probably do it better by avoiding getting stuck at local maxima/minima or requiring multiple trials with different initial configurations to give us a greater certainty that what the algorithm is finding is indeed a global maximum. I’m interested what other kind of optimization techniques could be used within this application or how a greater knowledge of stochastic processes could have helped us create an even better model/heuristic for our time series data.

Stella : I initially thought that this problem was very similar to the travelling salesman problem but it ended up being quite different. I found that there we quite a few complications on writing a good cost function and that even a great cost function may not work if the way that the neighbors are found is completely random. The current program seems to work enough but doesn’t seem to be much better or less computationally expensive than what my lab currently uses, which is just a greedy algorithm that connects the closest particles. Possibly with more work on it, it could be better than the current program.

What would you add to your program if you had more time:

The program does not work perfectly for the total range of data that we may observe experimentally. The disappearance and appearance of particles still causes problems. If we had more time we would add some more neighbor operators that would try and tackle the disappearance and appearance of particles and further improve the cost function for such cases.

Citations for references

Jaqaman, Khuloud, et al. "Robust single-particle tracking in live-cell time-lapse sequences." *Nature methods* 5.8 (2008): 695-702.

This is one of the current algorithms on how to create tracks for single particles. We used the idea from here that the merges and splits need to be addressed after the connections are satisfying.

<http://katrinaeg.com/simulated-annealing.html>

We used this website for the description of simulate annealing