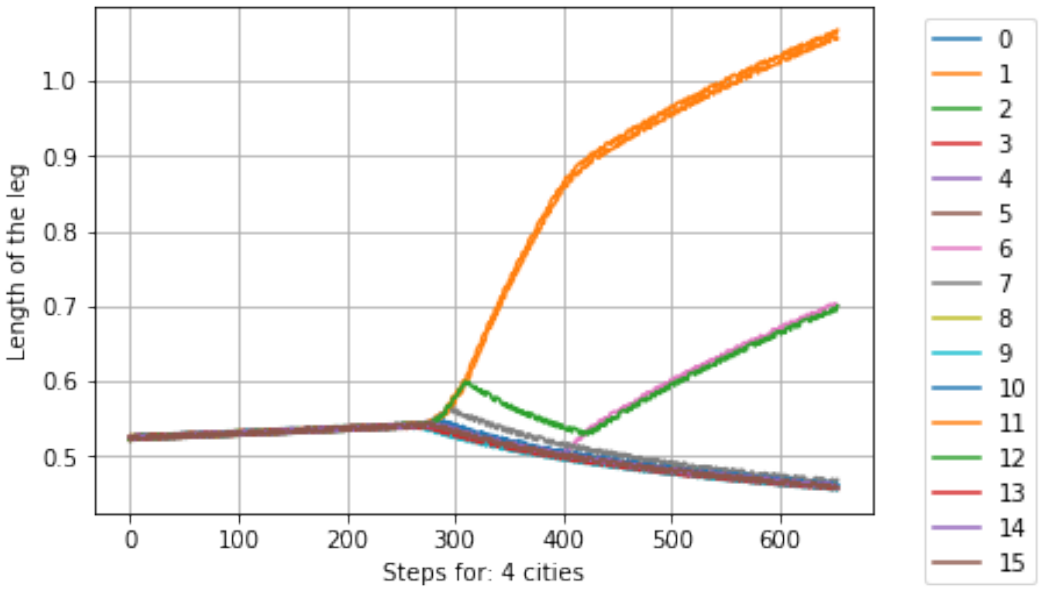
最終レポート　　環境三年西村郷

今学期、私はアメーバ型巡回セールスマン問題アルゴリズムを Python で実装することに取り組んだ。青野さんの多大なる支援のお陰で、課題はまだ多いもののなんとか過去の論文で出されていた結果をある程度再現できるプログラムが出来上がった。これまで上手くいっていなかった原因として、どのパラメータをどのタイミングで更新するか、という点について無頓着だったことが大きい。送ってくださった論文とプログラムを照らし合わせ、更新の順番を変えながら小さなバグを直すということを繰り返すことで、ようやく思い通りに動くものが出来上がった。これからはどのパラメータをどのタイミングで更新するか、という点に気をつけながらきちんと論文を読み進めるということに注意を向けていきたい。

仕組みをこの場でわざわざ解説するのは釈迦に説法であり、私自身もデバッグを通して仕組みが理解できたので、ここではプログラムを動かした結果を簡単にまとめることとする。

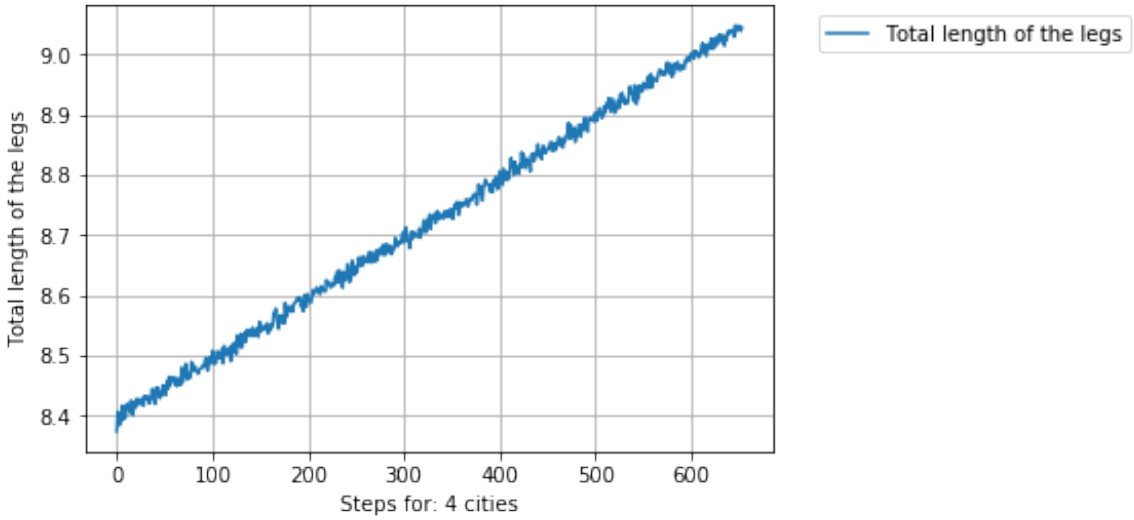
まずは一試行分、四都市から八都市の巡回セールスマン問題を解かせた時の振る舞いを見てみたい。Δin と Δout は 0.001、揺らぎの振幅は 0.003 である。

下の図は、ロジスティック写像を揺らぎとして四都市の問題を解かせた時の、各ステップにおける各足の振る舞いを示している。縦軸が足の長さ、横軸がその時のステップである。到達解は１、６、１１、１２（都市１２３０）であり、



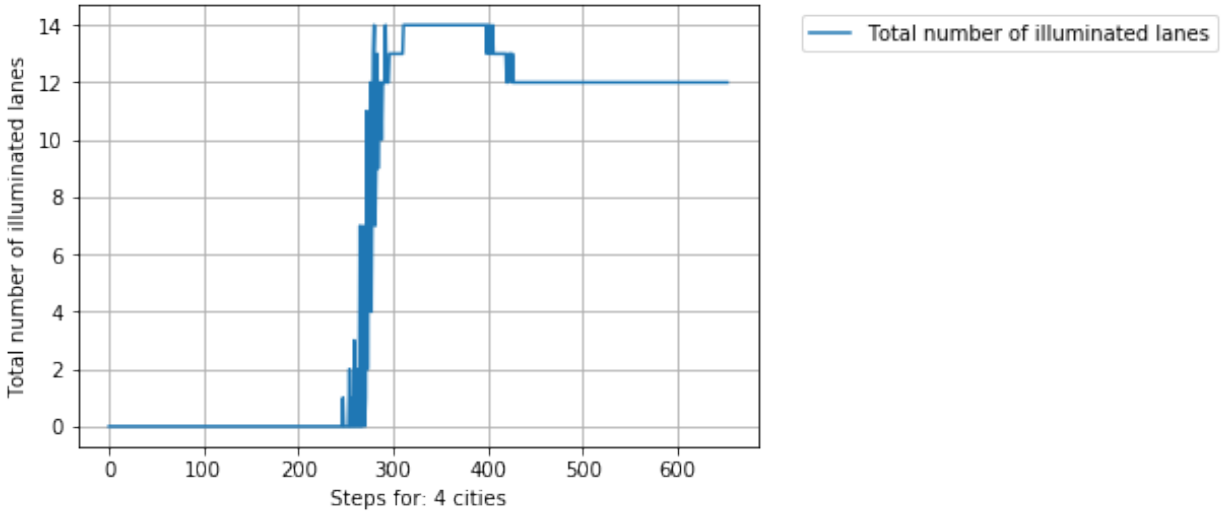
ルート長の精度が約 90%で最適解と思われる。青野さんのシミュレーションのものに対して、序盤で全ての足が同時に成長する期間が長いこと、中盤以降の足の成長速度が速いことが違いとして挙げられる。光の照射から足の出し入れ辺りで間違いを犯しているのかもしれないが、詳しい原因は不明である。

次の図は足の長さの総計である。



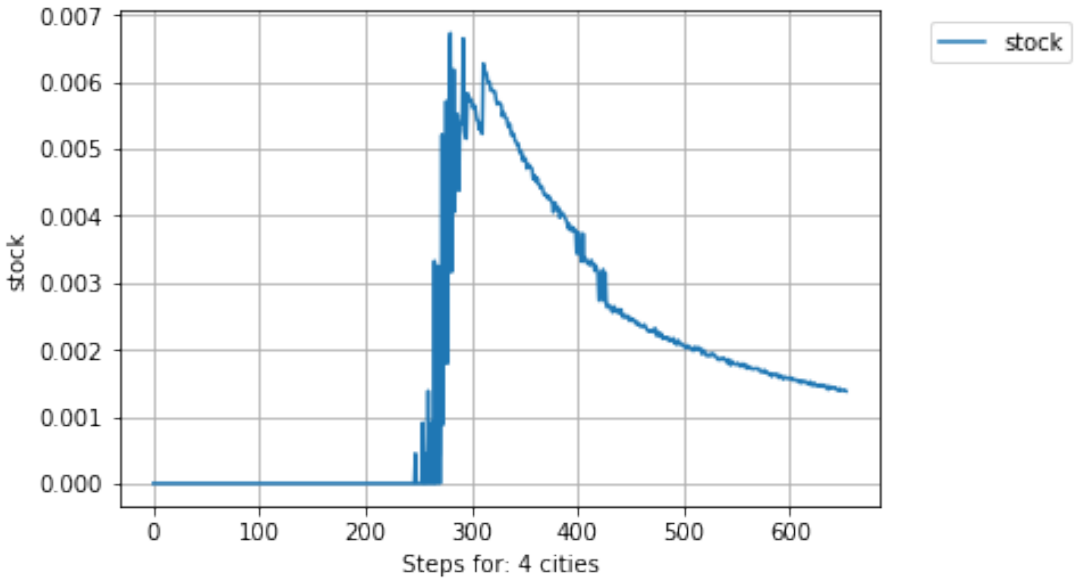
確かに線形に成長しているが、揺らぎの振幅を 0.003 にしたにしてはあまり揺らいでいない。振幅を小さくしたところ、ギザギザが直線になった。振幅を大きくするとギザギザの振幅は大きくなるものの、青野さんのシミュレーションのような全体的な揺らぎが観測されなかった。影響しているのが揺らぎでもなければストックでもないとしたら、一体何がこの違いを生んでいるのだろうか。

次の図は照射された足の本数である。



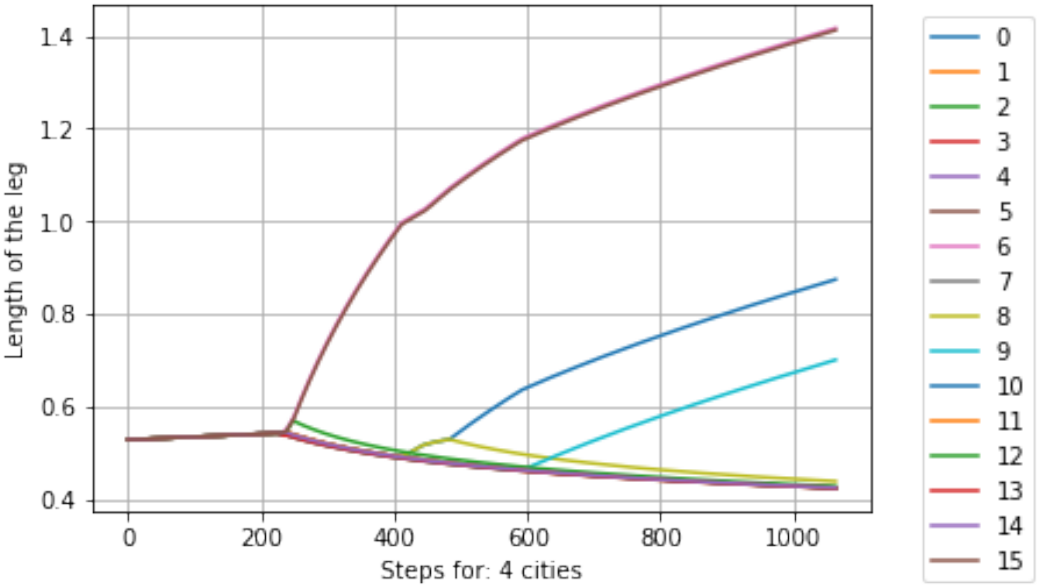
200 ステップ以降の照射本数があまりに乱高下している。これも揺らぎの影響を小さくすることで回避できるが、青野さんのシミュレーションと同じパラメータなのにここまで違いが出るのはやはり何かがおかしいのだろう。

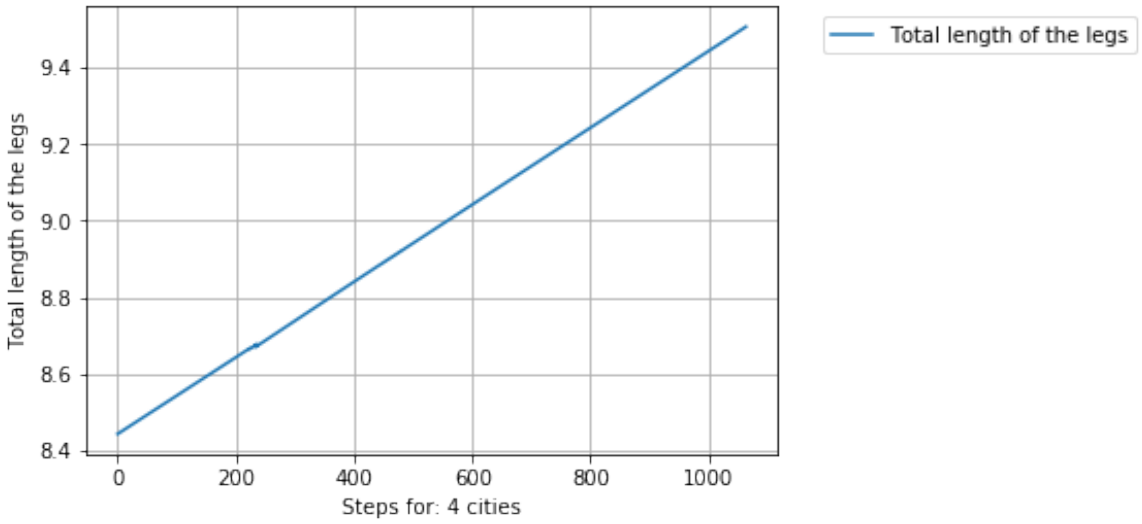
次の図はストックの変遷である。

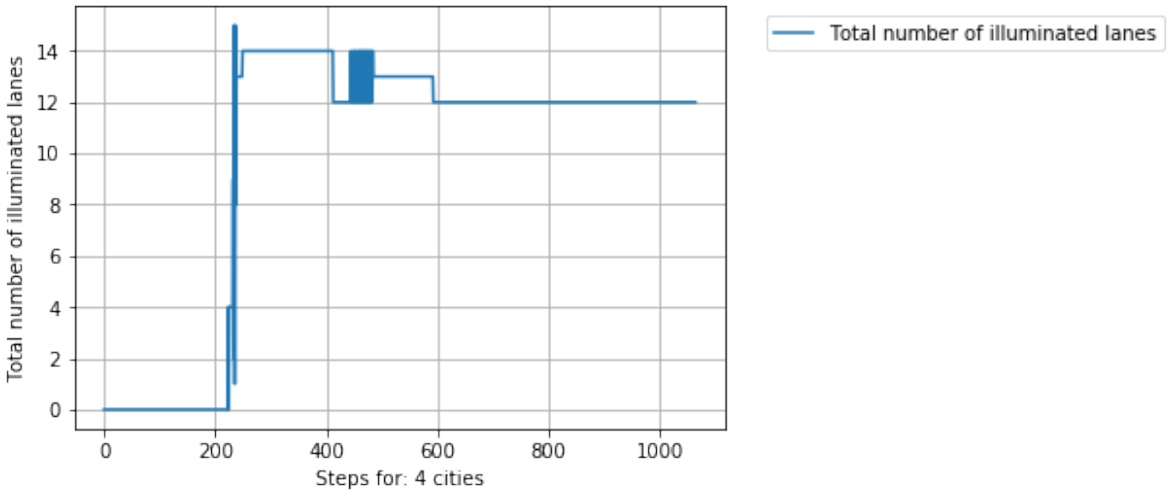


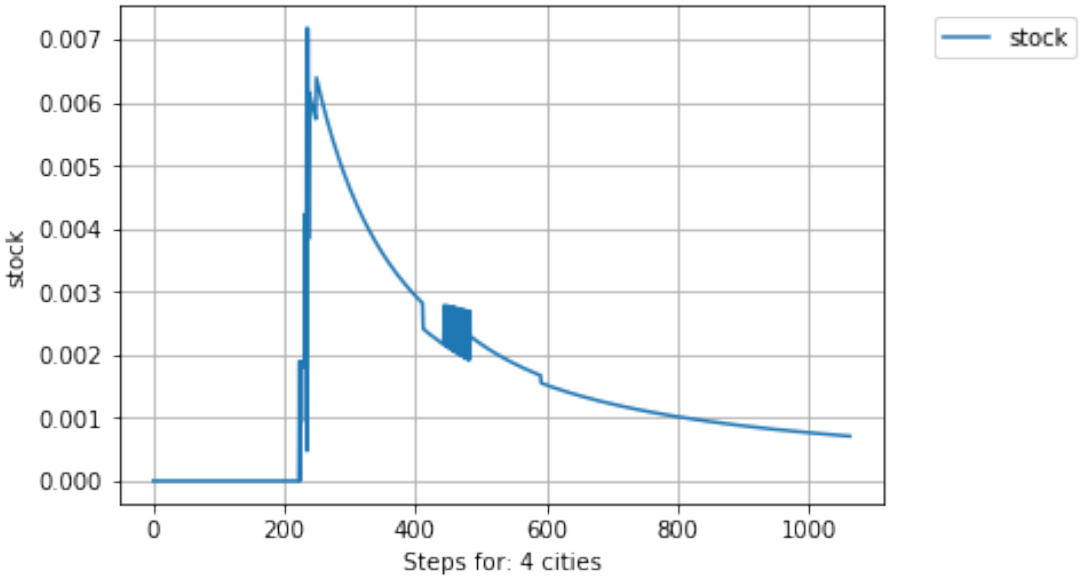
こちらもやはり 200 ステップ以降の乱高下が凄まじい。山を迎えた後に頂上付近で大きめの振幅を見せるのも気になる。

ランダムノイズでやってものも、基本的な傾向に変わりはなかった。サインノイズを入れたものは少し違ったので一応紹介する。




Total length of the legs
Total length of the legs
9.4
9.2
9.0
8.8
8.6
8.4
0
200
400
600
800
1000
Steps for: 4 cities

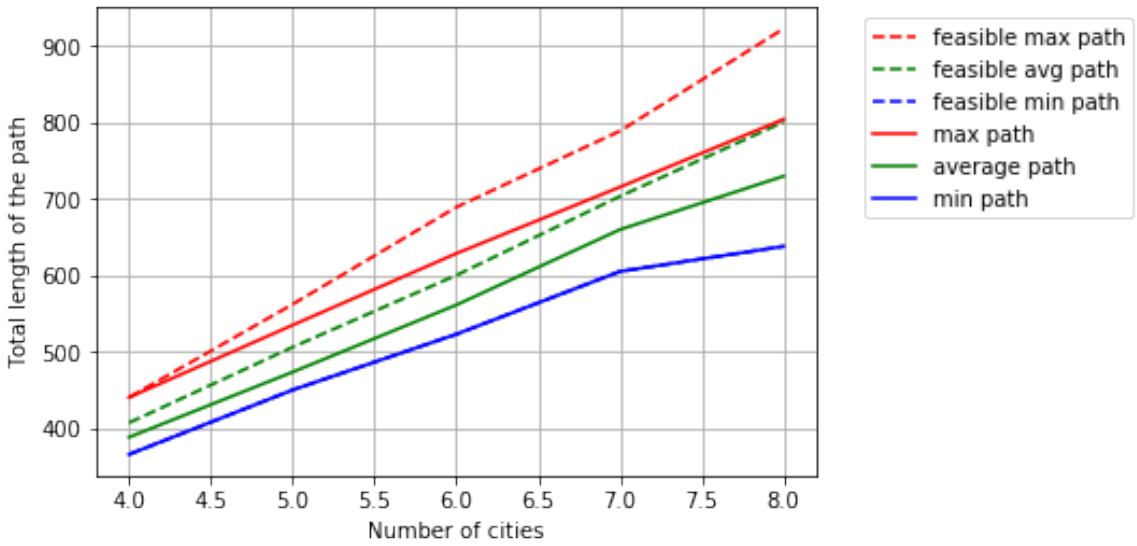

Total number of illuminated lanes
Total number of illuminated lanes
14
12
10
8
6
4
2
0
0
200
400
600
800
1000
Steps for: 4 cities


stock
stock
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
200
400
600
800
1000
Steps for: 4 cities

サインノイズは、他の二つのアルゴリズムに比べて値の乱高下が少なかった。これを防ぐ手立ては揺らぎの振幅を小さくすることや、ノイズの更新を止めてしまうことでも達成されるが、仮にそうしたところで乱高下が完全に防げる訳ではない。現状のプログラムの中にまだ不具合があるのかもしれないが、今のところ特に心辺りがない。

次に、100 回の試行による到達解のルート長、精度、ステップ数と経過時間を見ていく。まず、百回の試行をすると何度か許容解に到達しない場合が出てきた。今回は、都市数４のとき１４回、５で０、６で３、７で７、８で０回であった。問題の複雑さに応じて到達しなくなる、という訳ではなさそうなので、上で見られた値の乱高下（デッドロック？）がこれに影響しているのかもしれない。解に到達しなかった場合の最終的な足の状態は、都市数に満たない数の足が突出して足を伸ばし、他の足がほとんど足を縮めている、という状態だった。溜まりに溜まったストックが一気にそれら突出した足に流れ込むというより、長い時間をかけてそれらの足だけが光の当たらない状況が作られるということが起きていることを確認した。一応 Wvkul の設定の仕方を見直したが、ここには問題を見出せなかった。

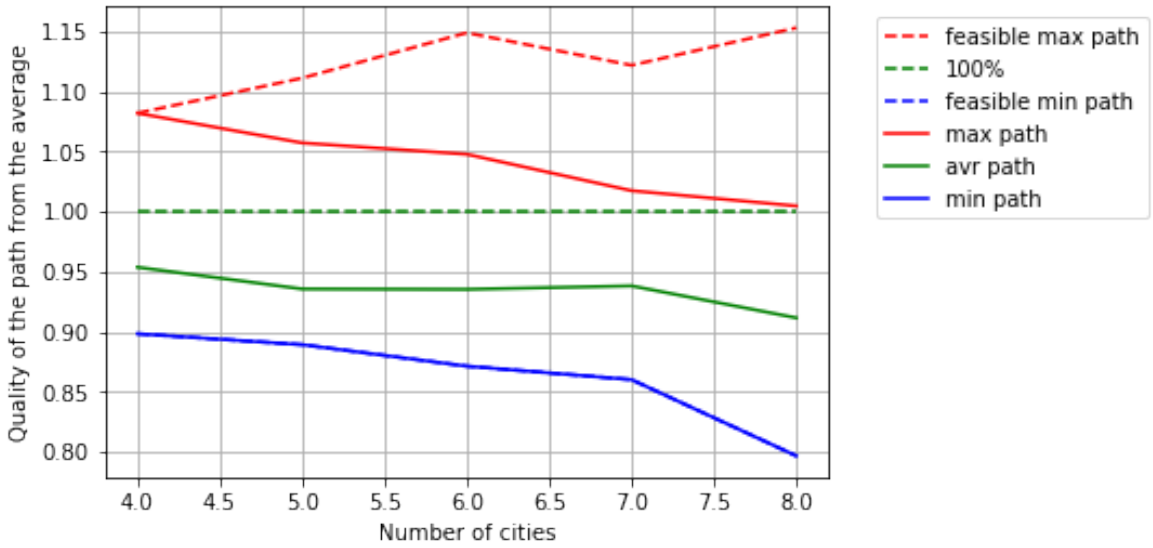
さて、ロジスティック写像の時の統計を見てみよう。まずはルート長である。



見ればわかるように、全ての解の平均ルート長は、問題の全ルート長の平均を下回って推移しており、単なるランダムチョイスよりも良い結果を出せることが分かる。また、最悪の到達解は、都市数が増えるごとに全ルート長の平均値に向

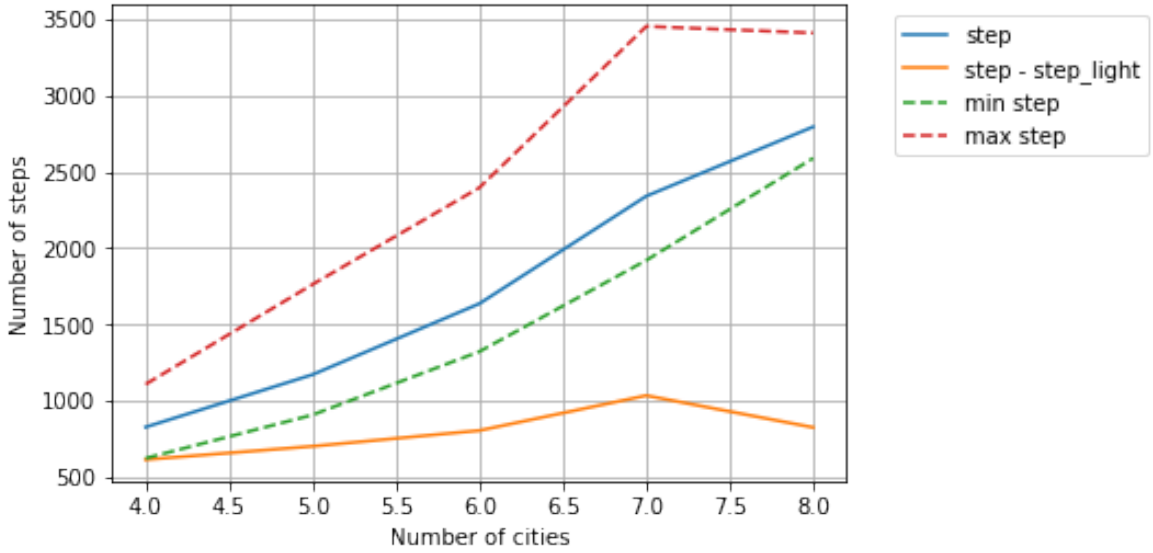
けて下がっていくことが分かる。もっと都市数を増やせば、これを下回ることが可能かもしれない。全ての問題において、百回の試行の中で必ず最適解に到達している。都市数 7 から 8 にかけては最適解がぐっと下がっているが、これもきちんと見つけ出している。

次は精度を見てみよう。



到達解のルート長はおおよそ 95%の精度を持つことが分かる。都市数 8 以降ではさらにこの精度が改善できるような動きも見せているが、最悪の到達解との距離も少しずつ狭まっており、予断を許さない状況ではある。

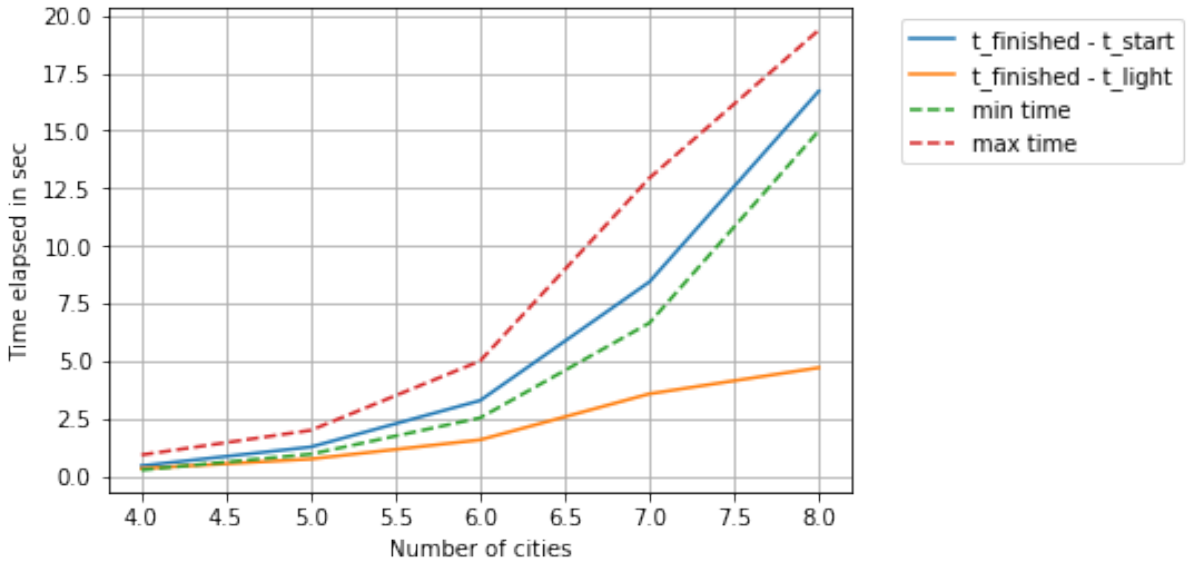
今度はステップ数である。



平均ステップ数はなんとなく線形な成長を見せている。照明が点いた以降のス

テップ数に関しては、７都市から８都市にかけて大きく増加しているのが特徴的だが、おそらく非線形になるだろう。最小ステップ数も非線形のきらいがあり、最大ステップ数もそんな感じがしてくる。やはり７都市から８都市にかけて何が起こったかの解釈がポイントで、生成する地図を変えたら全体的に非線形に舞い戻るかもわからない。

最後に経過時間である。



こちらははっきりと非線形な成長を遂げている。ステップ数と同様、ここでも照明が点灯してからの時間が急拡大しているのが気になるところではある。

ほとんど同様の結果が他の揺らぎでも見受けられた。ランダムノイズにおける未到達解は、４都市で１２、５と６で０、７で８、８で０となった。サインノイズの場合、４都市で１１、５と６で０、７で１０、８で０となった。こうした結果を見ていると、解への到達率はプログラムもそうだが地図の形状にも大きく依存するのかもしれない。

都市数に違いはあれど、統計に関してもやはり青野さんのシミュレーションとは違いがあるようである。特に精度が違っていて、青野さんのシミュレーションでは、精度が９割前後で、ばらつきもほとんどないのに対し、こちらのものは精度が五分程度高く、ばらつきも大きい。効率的に良い解を見つけられていない証拠であり、これについても今後究明がなされるべきだろう。

来学期としては、上述のバグを直すことを最優先としながらも、弄れる部分は積極的に弄ってみたい。例えば、活性化関数をシグモイド関数を使う代わりに、単純な線形関数などにしてみても面白いかもしれない。また、揺らぎについても他のロジスティック写像や、結合振動子のようなものを使ってみるべきだろう。ストックの配分の仕方にも揺らぎを入れてみたり、結合したグループのようにしてみても良いかもしれない。本物の粘菌コンピュータのように、足の伸びに限界を設けたり、解に到達した時に足の長さが 1 か 0 になるようにもしてみたいものである。個人的には実際に粘菌に問題を解かせた時に、縮んだ足の向こう側にごく微量の粘菌の欠片がレーンの中に残っているのが面白いと思っており、あれも何らかの形で実装してみたい気がする。最終的には経過時間そのものも線形の成長で抑えられるように頑張ってみたい（結局は問題サイズの影響が大きいのかもしれないが)。