Structured Analysis

February 7, 2016

1 Step by step analysis of Ant Colony Simulations

1.0.1 Ant Colony Evaluation through Simulations

In the last half of the century there were created techniques to analyze behavior through random repetition. These techniques are known as Monte Carlo Methods, this is name came from the old name of the casino game Roullete. Papers were written showing the power of simulating an experiment over thousand and thousand times.

So with this idea, I created an script that runs the Ant Colony System to solve the graph of 14 Cities in Burma. And based on this script and the Analysis of the results I also created an Step by Step Analysis to later confirm the results found to this instance in other graphs.

Ant Colony System is based on 6 Parameters: Alpha, Beta, Rho, Elite, Ant and Limit

1) Alpha is the parameter related to the pheromone amount in an edge
2) Beta is the parameter related to the shortness of an edge
3) Rho is the parameter related to the pheromone decrease rate
4) Elite is the parameter related to the pheromone boost to the best ant of one generation
5) Ant is the parameter related to the amount of Ants per generation
6) Limit is the parameter related to the number of generations the algorithm is going to run

To each of this parameter were chosen a sample of values to iterate over, and since the algorithm relies on randomness every group of the parameters were repeated 30 times, this way ensuring that a reliable solution may come appear.

- Initialize with the first value
- For 30 Times do
- Run ACS with chosen parameters
- Save sequence and results
- Change one parameter
- Repeat 2 until all parameters are tested

This Process generated a dataset with over a Million data points and with this Dataset I going to go step by step in finding the best values to find a good solution.

1.0.2 First Step

to read and clean the data turn it in a DataFrame

```
In [ ]: #importing needed libraries
        # DataFrame library
        import pandas as pd
        # numerical library
        import numpy as np
        # Scientifical library
        import scipy as sp
        # Statistical Library with R notation for Linear Models
        import statsmodels.formula.api as smf
        # Ploting Library
        import matplotlib.pyplot as plt
        %matplotlib inline

        #Creating the pandas DataFrame from csv file
        df = pd.read_csv("burma14extend.csv")
        #cleaning cells with Not Available data, or erased cell
        clean = df.dropna()
        #Creating a column with exceeding distance from optimal result
        clean.loc[:,'delta'] = clean.loc[:,'distance'] - 3323
        #Creating column with exceeding percentage from optimal result
        clean.loc[:,'%delta'] = clean.loc[:,'delta']/3323
```

1.0.3 Second Step

create hexagram of the columns

Answering the question: *How many times each value of a parameter got to optimal result*

With a hexagram we answer the general question: "How many times each value y was found by each value x" and this is displayed by intensity of color, a colorbar is displayed by side to help analyze it

```
In [ ]: #iterate through each parameter column
        for name in clean.columns:
            # since we are interested in the relation between distance and another parameter, those rel
            # are excluded from analysis
            if name in ['distance','ID','delta','%delta'] :
                continue
            # create figure
            fig, ax = plt.subplots(figsize=(8,6))
            # create plot type and change setup
            d = ax.hexbin(clean[name],clean['distance'], cmap=plt.cm.Blues, alpha=0.8)
            plt.xlabel(name,fontsize=14)
            plt.ylabel('Distance in Km',fontsize=14)
            plt.title('Hexbin of times distance X was reached with parameter '+name,fontsize=14)
            cb = fig.colorbar(d)
            cb.set_label('number of times',fontsize=14)
            fig.savefig('img/plot_hexbin_'+name+'.png')
```

Refining Repeat hexagrams

After analizing the result, we can see a lot of good results already, but the final answer is still obscure since the results are similar to a few values.

To reduce it we are increasing the **minimum** value, this way only after the count reach this minimum we display it

```
In [ ]: for name in clean.columns:
            if name in ['distance','ID','delta','%delta'] :
                continue
```

```
fig, ax = plt.subplots(figsize=(8,6))

# change point
# before we only plotted the hexagram, without tuning
# this time we set the minimal count parameter, so we only display values after it
# we set it different based on the results
if name == 'alpha':
    d = ax.hexbin(clean[name],clean['distance'],mincnt=8000, cmap=plt.cm.Blues)
elif name == 'rho':
    d = ax.hexbin(clean[name],clean['distance'],mincnt=25000, cmap=plt.cm.Blues)
elif name == 'limit':
    d = ax.hexbin(clean[name],clean['distance'],mincnt=48000, cmap=plt.cm.Blues)
else:
    d = ax.hexbin(clean[name],clean['distance'],mincnt=18000, cmap=plt.cm.Blues)

plt.xlabel(name,fontsize=14)
plt.ylabel('Distance in Km',fontsize=14)
plt.title('Hexbin of times distance X was reached with parameter '+name,fontsize=14)
cb = fig.colorbar(d)
cb.set_label('number of times',fontsize=14)
fig.savefig('img/plot_hexbin_refmin_'+name+'.png')
```

Analysis Unfortunatelly the dataset is not uniformilly shapped because of initial try&error tentatives of creating a standalone script to simulate it.

Even so, every case has at least 30 simulations of it, initial cases have a few more. Because of it, we can say that the points with most repeated solution are the best ones.

From the Graphs would be:

- Ant: 10 ~ 50 (Ant 50 is not fully simulated)
- Alpha: 7
- Beta: 7
- Rho: 0.2, 0.5, 0.8
- Elite: 0.6
- Limit: 100

Next step is to verify this with statistical analysis

1.0.4 Third Step

Statistical Analysis The idea in the previous steps were to gain information about this phenomenon, what is happening, is it replicable, when does it happens, how does it happens. Those kind of question were answered, at least in part. Now remains the question, ***why***?

And now we try to answer it. With some statistical tools we try to gain knowledge about this phenomenon

1) Linear Model 1 variable Our first try is to see how every parameter answer as a LM, we seek something like

```
y = a*x + b
```

This code show as y as variable dependable of x by a scale of a with an error of b
We are intereted in 3 values:

- The parameter value it self, in our exemple the value a
- The P-value, it shows the statistical significance of our parameter, in simple words: 'Is my value random?'

- The R^2 value, it shows how much this LM can explain my data
- The Correlation between our parameter and the distance, if the parameter increase 1 point what happens to the distance?

In [ ]:
```
# auxiliar array
aux = []
# same trick to ignore ours parameters
for name in clean.columns:
    if name in ['distance','ID','delta','%delta'] :
        continue

    # formula for the linear model
    # this notation can be read as
    # distance is proportional to parameter name
    # we are interested in find this proportion
    formula = "Q('distance') ~ Q('"+name+"')"
    model = smf.ols(formula,data=clean)
    # if the Number of OBServations is lower than ha
    if model.nobs < len(clean)/2:
        continue

    results = model.fit()

    #insert R^2 values in aux array with the parameter name
    aux.append((results.rsquared, name))
    #The Real Values of the Parameters
    print(results.params)
    #The P-value between 0 and 1, 0 is better
    print(results.pvalues)
    print(' ')
    #Pearson Correlation - Correlation
    print('pearson correlation distance and '+name,clean[name].corr(clean['distance'],method='p

    # Spearman Correlation - Correlation - with more robust approach to non linearity
    print('spearman correlation distance and '+name,clean[name].corr(clean['distance'],method='s
    print(' ')


#rank the array from greatest to lowest
aux.sort(reverse=True)
for mse,name in t:
    print(name,mse)
```

First Analysis A first look through this numbers show us that Beta is the most explanatory data, with a *85%* correlation with the decrease of distance. But our linear model only shows a R^2 of 63%

This may be related to seeing beta as an Linear parameter rather than an Non-Linear and even a Categorical one.

Looking at the graphs may helps us better understand it

In [ ]:
```
for name in clean.columns:
    if name in ['distance','ID','delta','%delta'] :
        continue

    formula = "Q('distance') ~ Q('"+name+"')"
```

```
        formula1 = "Q('%delta') ~ Q('"+name+"')"
        model = smf.ols(formula,data=clean)
        model1 = smf.ols(formulaq,data=clean)
        if model.nobs < len(clean)/2:
            continue

        results = model.fit()
        results1 = model1.fit()

        #ploting a figure with our data points and with the modelled points
        fig, ax = plt.subplots(figsize=(8,6))
        ax.plot(clean[name], clean['distance'], 'o', label="Data")
        ax.plot(clean[name], results.fittedvalues, '^r', label="Predicted")
        plt.xlabel(name,fontsize=14)
        plt.ylabel('Distance in Km',fontsize=14)
        plt.title('Variation of Final Distance by variation of '+name,fontsize=14)
        plt.grid(True)
        legend = ax.legend(loc="best")
        fig.savefig('img/plot_distance_'+name+'.png')
        #ploting a figure with our data points and with the modelled points
        fig, ax = plt.subplots(figsize=(8,6))
        ax.plot(clean[name], clean['%delta'], 'o', label="Data")
        ax.plot(clean[name], results1.fittedvalues, '^r', label="Predicted")
        plt.xlabel(name,fontsize=14)
        plt.ylabel('Delta in %',fontsize=14)
        plt.title('% of exceed distance by variation of '+name,fontsize=14)
        plt.grid(True)
        legend = ax.legend(loc="best")
        fig.savefig('img/plot_delta100_'+name+'.png')
```

Graph Analysis - Linear Model Both graphs for each parameter are essentially the same. The first one shows the Distance ~ Paramter and the second one %Delta ~ Parameter. The latter says about how much extra distance from the optimal this parameter create, or how close to optimal does it get?

We can see that because of the way this simulations were designed. With the Economics idea of Ceteris Paribus in mind, each simulation only changes one value and sees how the whole system reacts to it. This way even with a non-linear moviment as the Beta parameter, we should still treaty each possible value as category with a binary way, either on or off.

So in our refing step we change from LM to Categorical Binary Model.

In [ ]:
```
        # auxiliar array
        aux = []
        # same trick to ignore ours parameters
        for name in clean.columns:
            if name in ['distance','ID','delta','%delta'] :
                continue

            # formula for the linear model
            # this notation can be read as
            # distance is proportional to parameter name
            # we are interested in find this proportion
            formula = "Q('distance') ~ C(Q('"+name+"'))-1"
            model = smf.ols(formula,data=clean)
            # if the Number of OBServations is lower than ha
            if model.nobs < len(clean)/2:
```

```
            continue

        results = model.fit()

        #insert R^2 values in aux array with the parameter name
        aux.append((results.rsquared, name))
        #The Real Values of the Parameters
        print(results.params)
        #The P-value between 0 and 1, 0 is better
        print(results.pvalues)
        print(' ')
        #Pearson Correlation - Correlation
        print('pearson correlation distance and '+name,clean[name].corr(clean['distance'],method='p

        # Spearman Correlation - Correlation - with more robust approach to non linearity
        print('spearman correlation distance and '+name,clean[name].corr(clean['distance'],method='
        print(' ')


    #rank the array from greatest to lowest
    aux.sort(reverse=True)
    for mse,name in t:
        print(name,mse)
```

Graph Analysis - Categorical Model Now our results starts to improve, even values with statistical insignificance, given from high P-values, are better indicating their differences and also sorting which is the best of them.

Our R^2 is closer to our correlation now. This confirms that this Model better explain what our simulations intended to show, which parameter changes more our result, and from a preset of values which gives better performance.

From our Analysis until now they are:

1. Beta
2. Limit
3. Ant
4. Alpha
5. Elite - not significant
6. Rho - not significant

Our next step is to combine Beta and this other results to see what that bringes

In [ ]:
```
#Generating arrays for tuples of data
betas = np.array((0,0.25,0.5,0.75,1,1.5,2,2.5,3,3.5,4,4.5,5,6,7))
alphas = np.array((0,0.25,0.5,0.75,1,1.5,2,2.5,3,3.5,4,4.5,5,6,7))
ants = np.array((10,50))
limits = np.array((10,100))
rhos = np.array((0.2,0.4,0.5,0.6,0.8))
elites = np.array((0,0.2,0.4,0.5,0.6,0.8))

alpha = [(beta,i) for beta in betas for i in alphas]
ant = [(beta,i) for beta in betas for i in ants]
limit = [(beta,i) for beta in betas for i in limits]
rho = [(beta,i) for beta in betas for i in rhos]
elite = [(beta,i) for beta in betas for i in elites]
```

```
#Dict of tuples with the parameters values
dfs = {'alpha':alpha,'ant':ant,'limit':limit,'rho':rho,'elite':elite}

#skip trick
for name in clean.columns:
    if name in ['distance','ID','beta','delta','%delta'] :
        continue

    #formula for Two Categories
    formula = "Q('distance') ~ C(Q('beta')) + C(Q('"+name+"')) -1"
    model = smf.ols(formula,data=clean)
    results = model.fit()

    #plot data
    fig, ax = plt.subplots(figsize=(8,6))
    ax.plot(clean[name], clean['distance'], 'o', label="Data")
    #generate a data frame from the tuples
    df = pd.DataFrame(dfs[name], columns=['beta',name])
    #separate the different Betas so two show with diffrent colors
    for i in betas:
        x = df.loc[df.loc[:,'beta']==i]
        ax.plot(x[name],results.predict(x) , '^-', label="Predicted beta "+str(i))
    plt.xlabel(name,fontsize=14)
    plt.ylabel('Distance in Km',fontsize=14)
    plt.title('Variation of  Distance by variation of '+name+'with Beta separation',fontsize=14
    plt.grid(True)
    legend = ax.legend(loc="upper right",bbox_to_anchor=(1.3, 1.0))
    fig.savefig('img/plot_'+name+'_beta.png')
```

Graph Analysis - Two Categories Model Confirmation! So, the graphs confirm what we already knew. A higher Beta leads to a lower Distance, in all case-scenarios. This confirms the Spearman and Pearson correlation showing that beta alone explain most of the model. So we shall pick Beta 7. These graph also show that Rho and Elite have now real change between different cases. This way, there is no real picking, from this analysis alone.

Alpha in the other hand shows two moments of slight increases in 0 to 0.25 and 4 to 4.5, and no indication of decrease. So alpha 0 is the best result.

In contrast, Ant and Limit have great descent in the cases studied. Both of them shows that higher values lead to lower distance. From our cases, Ant 50 and Limit 100 are the choosen ones.

Remembering:

- Alpha: 0
- Beta: 7 (Higher)
- Rho: Anyone
- Elite: Anyone
- Ant: 50 (Higher)
- Limit: 100 (Higher)

Multi Category Model Analysis One last is to test to do is a stacked test. Change one item at a time we going to construct Seven Models and see how including that item change the previous model or the base model.

As result from this we can see how each value possible for the parameters affects our Model

```
In [ ]: #Seven Model based on all values possible for each category without an intercept
        formula1 = 'distance ~ C(beta)-1'
        formula2 = 'distance ~ C(beta)+C(ant)-1'
        formula3 = 'distance ~ C(beta)+C(limit)-1'
        formula4 = 'distance ~ C(beta)+C(limit)+C(ant)-1'
        formula5 = 'distance ~ C(beta)+C(limit)+C(ant)+C(alpha)-1'
        formula6 = 'distance ~ C(beta)+C(limit)+C(ant)+C(rho)-1'
        formula7 = 'distance ~ C(beta)+C(limit)+C(ant)+C(elite)-1'

        formulas = [formula1,formula2,formula3,formula4,formula5,formula6,formula7]
        t = []
        models = {}

        for formula in formulas:
            model = smf.ols(formula,data=clean)
            results = model.fit()

            t.append((results.rsquared, formula))

        t.sort(reverse=True)
        for mse,formula in t:
            #difference between this model and the Base model
            diff = mse - t[6][0]

            print(formula,mse*100,diff*100)
```

Categories Analysis Our latest test shows that Parameter Beta explain most of the Model, but Ant and Limit also contribute signifcantly. In opposite direction, Rho, Limit and Alpha impact very little with an explanation of 0,003% more than the other had already explained.

The values shown for the parameters are also in the way we expected, except for *alpha*. Even with low influence in the overall Model, the formula result indicates that Alpha 4.5 as the resposible for most impact in lowering the distance.

1.0.5 Conjugate Analysis and going Further

Both our analysis shows similarities in Beta and Limit. The Parameter Ant tends to higher values, but no concrete answer since it was not fully simulated by the end of this notebook with the repeated process, but least stick with this idea.

There were also some differences. Alpha in the stats should be 0, but there were more lower values in with alpha 7. Rho and Elite also have preferentiable values with the repeated process, but stats say there is no significance in them, so anyone is a good choice.

To sort this out, I ran a new simulation with values:

- Limit: 100, 500, 1000
- Ant: 50, 100, 200
- Alpha: 0, 4.5, 7
- Beta: 7, 8, 9
- Rho: 0.8
- Elite: 0.6

This new simulation is going to test the differences we found and our new beliefs:

- Which Alpha is the best?
- The Higher value for Limit, Ant and Beta are really better?