Optimized Social Network Representation Using Simulated Annealing Algorithm

Dr. C. Muthu
Associate Professor & Head
Department of Computer Science
Loyola College, Chennai
&
M. Arun Rajesh
Data Analyst, Shalom InfoTech,
Bharathidasan University Technology Park,
Tiruchirappalli

Abstract

Several business problems have many possible solutions across many variables. Their outcomes can vary drastically depending on the combinations of these variables. Stochastic Optimization Algorithms, such as the Simulated Annealing Algorithm, are often used for solving such business problems. In this paper, the determination of the Optimized Social Network Representation by using the Simulated Annealing Algorithm is discussed.

Keywords

Social Network Optimized Representation, Simulated Annealing Algorithm

Introduction

Business Organizations have recently started using several Machine Learning Algorithms for getting great business insights.[1] The successful implementation of advanced Machine Learning Algorithms is now achieved by the Business Organizations through the usage of Hadoop Ecosystem.[2] The KNN Algorithm is widely used at present for building Price Prediction Models.[3] The Hierarchical Clustering Algorithm is now used for getting useful insights regarding the customer preferences.[4] In this paper, the determination of the Optimized Social Network Representation by using the Simulated Annealing Algorithm is discussed. In a social network like MySpace, Face book, or LinkedIn, people are connected because they are friends or have a professional relationship. Each member of a social network chooses to whom they are connected, and collectively this creates a network of people. It is interesting to visualize such social networks to determine their structure, perhaps in order to find the Connectors, who know a lot of people or who serve as a link between otherwise self-contained social groups.

Optimized Representation of a Social Network

The optimized representation of a Social Network diagram shall be determined by using the Simulated Annealing Algorithm. A full-fledged Python code file is

gradually developed in this paper, which can take a list of facts about who is friendly with whom and arrive at an optimized representation of the resultant social network.

The determination of such optimal representation is usually done by using a **mass-and-spring** algorithm. When such an algorithm is used, the different nodes exert a push on each other and try to move apart, while the links try to pull connected nodes closer together. Thus, the social network slowly assumes a layout where unconnected nodes are pushed apart and connected nodes are pulled close together - but not too close together.

Unfortunately, the mass-and-spring algorithm does not stop a line in a social network diagram from crossing another line in that diagram. In a social network that has a great number of links, this makes it difficult to view which nodes are connected because visually tracking the lines as they cross each other can be tricky at times. In order to arrive at a social network diagram with minimum numbers of crossed lines, the simulated annealing optimization algorithm shall be used, which is based on an appropriate **cost function**. In this study, a cost function shall be formed so as to minimize the number of lines that cross each other.

Cost Function

Every node in the social network representation has ***x*** and ***y*** coordinates. The coordinates for all the nodes in the social network diagram can be represented as a long list, as shown below:

```
soln = [110, 190, 240, 115, ... ... ...]
```

In the above solution, the first person is placed at (110, 190), the second person is placed at (240, 115), and so on. The cost function that is used in this study will count the number of lines that cross each other. It is necessary to calculate the fraction of the line where each line is crossed. If this fraction is between 0 (one end of the line) and 1 (the other end), for both lines, then they cross each other. If the fraction is not between 0 and 1, then the lines do not cross.

The following **doCrossCount( )** function loops through every pair of links and uses the current coordinates of their end points to determine whether they cross. If they cross, the function adds 1 to the total score. A new Python program file called **enhanceSocialNetwork.py** is to be created and the following **doCrossCount( )** function is to be added to it.

```
def doCrossCount(v):
    # The number list is converted into a dictionary of person: (x,y)
      loc = dict ([(people [i], (v[i*2], v[i*2+1]))
      for i in range (0, len (people))])
      total = 0
```

```
# Every pair of links is looped through
  for i in range (len (links)):
  for j in range (i+1, len(links)):
# The locations are determined
  (x1, y1), (x2, y2) = loc[links[i][0], loc[links[i][1]]
  (x3, y3), (x4, y4) = loc[links[j][0], loc[links[j][1]]
  den = (y4–y3) * (x2–x1) – (x4–x3) * (y2–y1)
# den is equal to 0 if the lines are parallel
  if den == 0 : continue
# In other cases, ua and ub are the freaction of the lines
# where they cross each other
  ua = ((x4–x3) * (y1–y3) – (y4–y3) * (x1–x3)) / den
  ub = ((x2–x1) * (y1–y3) – (y2–y1) * (x1–x3)) / den
# In case the fraction is between 0 and 1 for both lines
# then they cross each other.
  if ua > 0 and ua < 1 and ub > 0 and ub < 1 :
  total +=1
  return total
```

The domain for the optimum solution search is the range for each coordinate. In this paper, it is assumed that the social network diagram will be laid out in a 400×400 size image. But, the domain will be a little less than that in order to allow for a slight margin around the diagram. So, the following line of code is to be added to the end of enhanceSocialNetwork.py :

```
domain = [(10, 370)] * (len(people) * 2)
```

Simulated Annealing Optimization Algorithm

The Simulated Annealing Optimization Algorithm begins with a random solution to our optimized social network representation problem. There is a variable representing the willingness to accept a worse solution, which starts very high and gradually gets lower. In each iteration, one of the numbers in the solution is randomly chosen and changed in a certain direction. It the new cost is lower; the new solution becomes the current solution. However, if the cost is higher, the new solution can still become the current solution with a certain probability. This is done so as to avoid the local minimum problem.

In some cases, it is necessary to move to a worse solution before we can get to a better one. Simulated Annealing Algorithm works well because it will always accept a move for the better, and because it is willing to accept a worse solution near the beginning of the process. As the process goes on, the algorithm becomes less and less likely to accept a worse solution, until at the end it will only accept

a better solution. The probability of a higher-cost solution being accepted is given by the following formula:

$$p = e^{((-highcost-locost)/willingness\ to\ accept\ a\ worse\ solution)}$$

Since the willingness to accept a worse solution starts very high, the exponent will always be close to 0, so the probability will almost be 1. As the willingness to accept a worse solution decreases, the difference between the high cost and the low cost becomes more important. As a bigger difference leads to a lower probability, the algorithm will favour only slightly worse solutions over much worse ones.

The following function **doAnnealingOptimize()** is to be added to **enhanceSocialNetwork.py**:

```
def doAnnealingOptimize (domain, costf, T=10000.0, coolingRate = 0.95,
                         step = 1):
  # The values are initialized randomly
    vector1 = [float (random.randint (domain[i] [0], domain[i][1]))
    for i in range (len(domain))]
    while T > 1.0 :
  # One of the indices is chosen
    i = random.randint (0, len(domain) – 1)
  # A direction to change the index is chosen
    direction1 = random.randint (–step, step)
  # A new list with one of the values changed is created
    vectorb = vector1[:]
    vectorb [i] += direction1
    if vectorb [i] < domain[i][0] : vectorb[i] = domain[i][0]
    elif vectorb[i] > domain [i][1] : vectorb[i] = domain [i][1]
  # The current cost and the new cost are calculated
    ea = costf (vector1)
    eb = costf (vectorb)
    p = pow (math.e, (–eb–ea) / T)
  # Check whether it is better, or it makes the probability cutoff
    if (eb < ea or random.random( ) < p) :
    vector1 = vectorb
  # The willingness to accept a worse solution is decreased
    T = T * coolingRate
    return vector1
```

To perform annealing, the above function first creates a random solution of the right length with all the values in the range specified by the domain parameter.

The *willingness to accept a worse solution* and the *coolingRate* are optional parameters. In each iteration, *i* is set to a random index of the solution and *direction1* is set to a random number between *–step* and *step*. It calculates the current function cost if it changes the value at *i* by *direction1*.

The line of code *p = pow (math.e, (–eb–ea) / T)* shows the probability calculation, which gets lower as T gets lower. If a random float between 0 and 1 is less than this value, or if the new solution is better, the function accepts the new solution. The function loops until the willingness to accept a worse solution has almost reached 0, each time multiplying it by the cooling rate.

The optimized social network's nodes' coordinates shall now be obtained by using the **Simulated Annealing Technique** in the following Python session:

```
>>> import enhanceSocialNetwork
>>> soln=enhanceSocialNetwork.doAnnealingOptimize
    (enhanceSocialNetwork.domain,
    enhanceSocialNetwork.doCrossCount,
    step = 50, coolingRate = 0.99)
>>> enhanceSocialNetwork.doCrossCount (soln)
1
>>> soln
[314, 180, 231, 319, 288, 227, 107, 171, 78, 96, 46, 10, 286, 360, 11, 302]
```

Here, the Simulated Annealing Technique has provided an optimum solution in which only one pair of lines will cross each other. As per the above optimum solution, the first person is placed at the position (314, 180), Augustus at (231, 319), and so on.

References

1. Jacques Bughin, "Big Data, Big Bang?", *Journal of Big Data*, 2016,Vol.3, Iss. 2, pp. 1-14.
2. Muthu, C. and Prakash, M.C., "Impact of Hadoop Ecosystem on Big Data Analytics", *International Journal of Exclusive Management Research*, Special Issue, 2015, Vol. 1, Iss. 1, pp. 88-90.
3. Muthu, C. and Prakash, M.C.,"Building a Price Predictor for an Auctioning Website", *ReTeLL*, 2015, Vol. 15, Iss. 1, pp. 135-137.
4. Muthu, C. and Prakash, M.C., "Hierarchical Clustering of Users' Preferences", *ReTeLL*, 2016, Vol. 16, Iss. 1, pp. 135-136.
5. Muthu, C. and Prakash, M.C., "Matching the users of a Website using SVM Technique", *ReTeLL*, 2017, Vol. 17, Iss. 1, pp. 53-56.
6. Muthu, C. and Prakash, M.C., "Using Bayesian Classifier for Email Sorting", *ReTeLL*, 2017, Vol. 17, Iss. 1, pp. 57-60.

———