STRING MATCHING AND QUANTUM OPTIMIZATION TO INVESTIGATE THE ROLE OF LEAF ORIENTATION ON PHOTOSYNTHESIS

Ayşe Bilgin[1,*], Elif Sümeyye Cirit[2], Rüştü Erciyes Karakaya[3], Zeynep Özay[4], and Yalçın Tür[5]

[!]Authors are listed alphabetically by surname and the indexes are used only to separate the authors from each other when entering their information below and do not indicate any difference (First author, second author, etc.,) as the contributor to the project. All these authors contributed equally to this work.
!

[1]Boğaziçi University, Department of Physics, Istanbul, Turkey
[2]Yeditepe University, Department of Materials Science and Nanotechnology Engineering, Istanbul, Turkey
[3]Sabanci University, Department of Electronics Engineering with minor in Mathematics, Istanbul, Turkey
[4]Hayreddin Karaman Anatolian High School, Istanbul, Turkey
[5]Koç High School, Istanbul, Turkey
[*]ayse.bilgin@boun.edu.tr

ABSTRACT

In this study, we investigated the leaf orientation required for a selected plant species to benefit from photosynthesis optimally. For this, we selected a main line in Python and generated random needles with different lengths and different angle values (angle between needle and main branch). We modeled the sequence that provides the most optimum benefit with mathematical and computational structures and investigated how suitable the existing biological systems are for these models by using "Quantum String Matching" and "Quantum Optimization" methods. At the end of our study, we observed that this similarity gives a result lower than 50% in a classical computer, but gives a better result when analyzed in a quantum structured system.

Introduction and Aim

Biological life forms maintain their vitality by consuming various forms of energy. Photosynthesis is one of these forms of energy consumption which has long been known and it is of great importance for plants to benefit from the sun in the maximum amount. Therefore, plants have developed various adaptations in order to obtain this maximum benefit. In this study, we examine whether these forms, which plants have obtained as a result of many years of adaptations, reach the optimum perfection predicted by our mathematical and computational models. This questioning brings with it another process of questioning: Does nature evolve to approach that ideal that our models can predict, given enough time, or will biological systems not reach this ideal form no matter how much time passes and how much they change?

Sampling and Biological Modeling

To make the control mode, the most convenient plant species must be determined. A conifer tree species was preferred rather than the broad leaf plant species because each needle (leaf) can be represented by only one vector and this can help to make a simpler version of the Nature. Spruce is one of the most appropriate candidate among those. There are some reasons for this; i) the needles of this species attached individually to the branches and ii) the needles are sharply pointed.[1] After deciding a particular specie, a new question emerges: “What is the orientation of the needles of spruce in nature?”. Kimes and Smith (1990) have made a study to determine the inclination distributions and size measurements of two different conifer species. As a result of this study, the distributions of the zenith angle of the normal to the needle surface θ_z for the hemlock and spruce have been determined and accordingly the angle was determined to be between 10° and 20° with the highest degree of probability as shown in the Figure-2. Exploiting this information, the distributions of the angle between needle and branch i were estimated. The needle was assumed to be perfectly straight, thus θ_i became equal to $90 - \theta_z$ and θ_i should be between 70° and 80° with the highest degree of probability. In addition, the mean length of the Spruce needles has been found as 1.00 cm (standard deviation = 0.117, n = 31).[2]

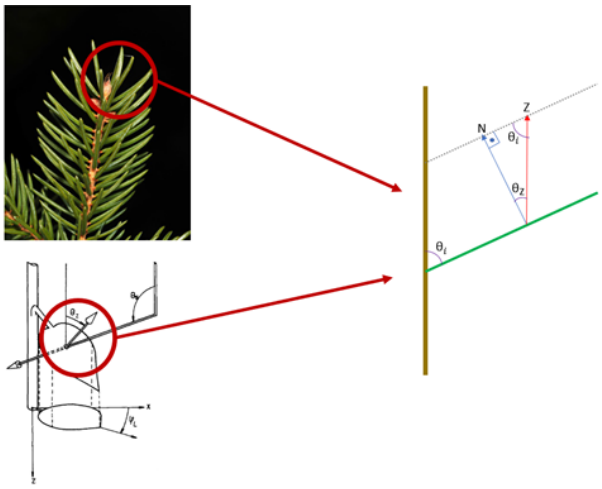


Figure 1. Schematic representation of obtaining the angle between needle and branch θ_i from the zenith angle of the normal to the needle surface θ_z.

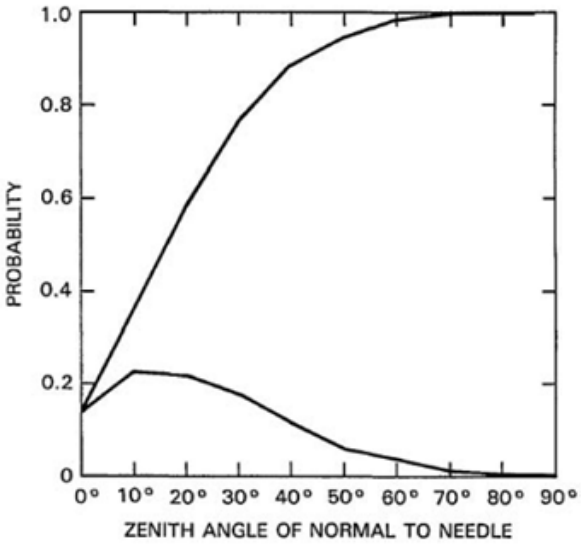


Figure 2. Discrete and cumulative distributions of the zenith angle of the normal to the needle surface for red spruce. Ten degree intervals of angle were used.[2]

The next step to develop the model was to determine the optimum angle of incidence of incoming radiation. Kriedeman et. al. (1964) have conducted an experiment about the photosynthetic behaviour of single attached leaves for four different species and at 0° to 90° angle of incidence. As a result of this experiment, It has been observed that the photosynthetic rate of a leaf is proportional to the cosine of the angle of incidence of incoming light beam. As the light entering the leaf decreases, on the other words, as the angle of incidence of light decreases, the cos increases and the efficiency of photosynthesis increases as the intensity of illumination in the leaf decreases. Therefore, photosynthesis is more efficient for a small angle of incidence light or high value of $\cos\theta$. This relationship is clearly shown in the Figure-3 .[3]

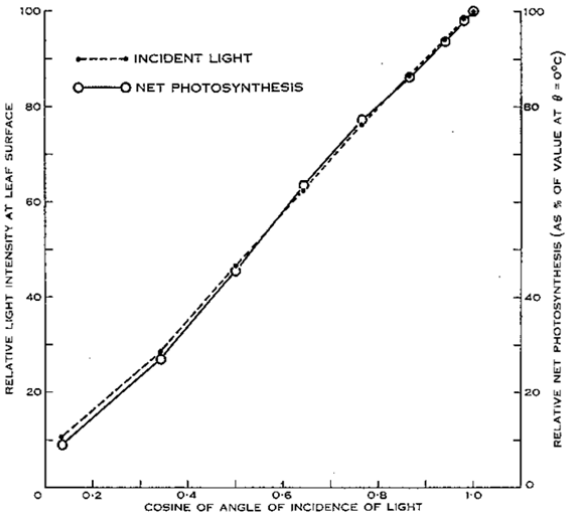


Figure 3. Net photosynthesis (mean for four leaves), and light intensity at the leaf surface, in relation to the cosine of the angle of incidence of incoming radiation.[3]

Quantum Mechanical String Matching Approach

Step-I: Modeling leaves on bough in python

Firstly, the model is considered at 2-D Cartesian system to facilitate simulation. The bough can be considered as a main line and needles can be modeled as vectors. These vectors are initialized from random generated point on main line. Initial and final points of main line are obtained by input and then the algorithm generates points on main line to generate vectors around this point for modeling of leaves ("Leaves" and "Needles" are going to be used interchangeable throughout the paper.). The leaves are, again, constructed by inputs which are the angle and the length of leaf. The angle points out here is the degree between main line and leaf. For example if we put alpha as angle, this starts to generate leaves around point by iteration until as seen at the Figure-4.

```
i=1
while i * angle_converted_radian < math.pi:
  x_new = rand_x + length_of_little_leaf* math.cos(angle_converted_radian*i + np.arctan(slope_of_the_bough))
  y_new = rand_y + length_of_little_leaf* math.sin(angle_converted_radian*i + np.arctan(slope_of_the_bough))
  mydict_for_points[x_new]=y_new

  #symmetric process
  k=(2*length_of_little_leaf* math.sin(angle_converted_radian*i))/ math.sqrt(1+slope_of_the_bough**2)
  x_new_symmetric= x_new + (k*slope_of_the_bough)
  y_new_symmetric= y_new - k
  mydict_for_symmetric_points[x_new_symmetric]=y_new_symmetric

  i=i+1

list_for_all_points.append(mydict_for_points)
list_for_symmetric_ones.append(mydict_for_symmetric_points)
```

Figure 4

All combinations around one point which was determined in randomness from main line is packed and ordered respectively in a form of dictionaries and then it is put in lists. Each dictionary in list begin with coordinate pair of the random point and continue with final coordinate pairs of vectors around this main point accordingly given angle and length of leaf as mentioned above.

The symmetric process in Figure-4 is about generating leaves around main point again, however, this generation continue until π; that's why it has to be symmetrized to obtain all combinations of positioning of leaves. Symmetric process produces the same operation which is done before for one side by the constant, k.

Step-II: Collecting all point-pairs in three dictionaries

All (x,y) pairs is classified into three lists: One of them is for random generated points on main line, another one is final point of vectors and the last one is symmetrical version of final points of leaves around main point. List structure is explained above, which starts with main point and final point of vectors around it.

Step-III: Picking randomly coordinates from dictionaries to construct form of leaves on bough

This step is constructing the sample. All vectors have main point on main line and which vector and which symmetrical one will construct the form of leaves on bough is a problem. That's why this structure can be generated in a randomness to sketch various combinations as sample. It can be resembled to picking sample from the nature to compare ideal one.

```
#skecth some combinations
list_of_string_shapes =[]
j=0
while j<5:
  k=0
  string_for_shape=""
  while k<len(list_for_rand_x_from_main_line):
    main_point_x=list_for_rand_x_from_main_line[k]
    main_point_y=list_for_all_points[k].get(main_point_x)
    random_x=random.choice(list(list_for_all_points[k].keys()))
    random_y=list_for_all_points[k].get(random_x)
    while random_x==main_point_x:
      random_x=random.choice(list(list_for_all_points[k].keys()))
      random_y=list_for_all_points[k].get(random_x)
      if random_x!=main_point_x:
        break
    print("coordinates of leaf: ",random_x,"--",random_y)
    random_x_for_symmetric=random.choice(list(list_for_symmetric_ones[k].keys()))
    random_y_for_symmetric=list_for_symmetric_ones[k].get(random_x_for_symmetric)
    while random_x_for_symmetric==main_point_x:
      random_x_for_symmetric=random.choice(list(list_for_symmetric_ones[k].keys()))
      random_y=list_for_symmetric_ones[k].get(random_x)
      if random_x_for_symmetric!=main_point_x:
        break
```

Figure 5. for Step-III: Picking randomly coordinates from dictionaries to construct form of leaves on bough

Step-IV: Converting all shapes in Cartesian-Plane to string

There are random shapes in a randomness and at this step, they will be converted into sub-string form to use string matching. We have coordinates which are numbers and they have to be converted into character form via ASCII table before the string, after that it can be summed and obtained sub-string. As known, ASCII table is on the range of 1-127 as decimal, and to obtain meaningful characters for string, we picked up the range of 47-127. Namely, the bijection between coordinates and this range must be constructed. The coordinate pairs cannot be integer although ASCII works with integers. That's why this bijection must also generate integers since domain is proceed continuously while range works on discrete values if it is considered functions. Mathematicians use ceiling and floor function for that reason, therefore, we also used these functions to construct bijection as seen in Fig-6. In ceiling and floor functions, we used max length of leaf, which was get as input, to consider maximum deviation of vectors from the main point, in a matter of course, we don't want to go beyond the 127.

```
string_for_shape = string_for_shape + chr(math.floor(47+main_point_x+max_length_of_leaf-random_x))+ chr(math.floor(47+main_point_y+max_length_of_leaf-random_y))
x_values_for_leaf_1 = [main_point_x,random_x]
y_values_for_leaf_1 = [main_point_y,random_y]
x_values_for_leaf_2 = [main_point_x,random_x_for_symmetric]
y_values_for_leaf_2 = [main_point_y,random_y_for_symmetric]
string_for_shape = string_for_shape + chr(math.ceil(47-main_point_x-max_length_of_leaf+random_x_for_symmetric))+ chr(math.ceil(47-main_point_y-max_length_of_leaf+random_y_for_symmetri
```

Figure 6

Step-V: Sketching ideal one, then converting to string and string matching

As done before, all shapes are converted to string with coordinates of the leaves and this time, ideal one is also converted to string with main points on coordinates of the bough and the leaves. In this way, the string of ideal one is longer than the other string forms, in another words, string forms from our samples can be sub-string of string form of ideal one. Thus, the main aim of this algorithm, namely string matching, can be applied. Same process is used as Figure-6 above in the converting into string form. However, the choosing ideal one and sketching it differ from what we did before in some aspects. There is some rules for sketching ideal one, which are mainly related to the angle of incidence of sun; if this angle is parallel to leaf $(cos(0))$, photosynthesis reaches maximum productivity in terms of this parameter. Here is that algorithm that scales this cosines angle and checks the most parallel case, if it is possible with available coordinate pairs. To do that, it ramifies the cases according to where the sun is.

```
list_for_ideal_shape=[]
counter_to_identify_process=0
print((y_for_sun+slope_of_the_bough + x_for_sun - slope_of_the_bough+constant) / (slope_of_the_bough**2 +1))
if x1 <= (y_for_sun+slope_of_the_bough + x_for_sun - slope_of_the_bough+constant) / (slope_of_the_bough**2 +1) <= x2: #it have to be revised
  counter=0
  if y_for_sun > (slope_of_the_bough)*x_for_sun + constant:
    for i in list_for_all_points :
      mydict={}
      mydict[next(iter(i))] = i.get(next(iter(i)))
      if next(iter(i)) > x_for_sun:
        #get second key-value from dict of symmetric ones
        x_for_ideal_shape=list(i.keys())[len(i)-1]
        y_for_ideal_shape=i.get(x_for_ideal_shape)
        mydict[x_for_ideal_shape] = y_for_ideal_shape
        x2_for_ideal_shape=list(list_for_symmetric_ones[counter].keys())[len(list_for_symmetric_ones[counter])-1]
        y2_for_ideal_shape=list_for_symmetric_ones[counter].get(x2_for_ideal_shape)
        mydict[x2_for_ideal_shape] = y2_for_ideal_shape
      else:
        x_for_ideal_shape=list(i.keys())[1]
        y_for_ideal_shape=i.get(x_for_ideal_shape)
        mydict[x_for_ideal_shape] = y_for_ideal_shape
        x2_for_ideal_shape=list(list_for_symmetric_ones[counter].keys())[1]
        y2_for_ideal_shape=list_for_symmetric_ones[counter].get(x2_for_ideal_shape)
        mydict[x2_for_ideal_shape] = y2_for_ideal_shape

      list_for_ideal_shape.append(mydict)
      counter=counter+1
      counter_to_identify_process =1

  if y_for_sun < (slope_of_the_bough)*x_for_sun + constant:
    #same process like above
    for i in list_for_symmetric_ones :
      mydict={}
      mydict[next(iter(i))] = i.get(next(iter(i)))
      if next(iter(i)) > x_for_sun:
        x_for_ideal_shape=list(i.keys())[len(i)-1]
        y_for_ideal_shape=i.get(x_for_ideal_shape)
        mydict[x_for_ideal_shape] = y_for_ideal_shape
        x2_for_ideal_shape=list(list_for_all_points[counter].keys())[len(list_for_symmetric_ones[counter])-1]
        y2_for_ideal_shape=list_for_all_points[counter].get(x2_for_ideal_shape)
        mydict[x2_for_ideal_shape] = y2_for_ideal_shape
      else:
        x_for_ideal_shape=list(i.keys())[1]
        y_for_ideal_shape=i.get(x_for_ideal_shape)
        mydict[x_for_ideal_shape] = y_for_ideal_shape
        x2_for_ideal_shape=list(list_for_all_points[counter].keys())[1]
        y2_for_ideal_shape=list_for_all_points[counter].get(x2_for_ideal_shape)
        mydict[x2_for_ideal_shape] = y2_for_ideal_shape

      list_for_ideal_shape.append(mydict)
      counter=counter+1
```

Figure 7. sketching ideal one accordingly positioning of the sun

Implementation of String Matching Approach into the Quantum Computation

If we consider the sub-strings and main string in binary string form, after this converting; main string, T (:text), and sub-string, P (:pattern), can be encoded as seen in Figure-8.

$$|\mathcal{T}\rangle = |t_0 t_1 \dots t_{N-1}\rangle = \bigotimes_{i=0}^{N-1} |t_i\rangle,$$

$$|\mathcal{P}\rangle = |p_0 p_1 \dots p_{M-1}\rangle = \bigotimes_{j=0}^{M-1} |p_j\rangle,$$

Figure 8. Niroula, P., Nam, Y. A quantum algorithm for string matching. npj Quantum Inf 7, 37 (2021). https://doi.org/10.1038/s41534-021-00369-3

In this part N and M symbolize the length of T and P respectively, in a matter of course, M < N. Quantum registration is required to implement our problem with quantum concepts, there are two quantum register as T and P in this case. The main reason of registration is also about the domain of the states, they have to be lie on Hilbert Space to make calculations over quantum computing. Thus, initial state can be constructed over T and P.

$$|\psi\rangle = |0\rangle^{\otimes n} \left[\bigotimes_{i=0}^{N-1} |t_i\rangle\right] \left[\bigotimes_{j=0}^{M-1} |p_j\rangle\right],$$

Figure 9. Niroula, P., Nam, Y. A quantum algorithm for string matching. npj Quantum Inf 7, 37 (2021). https://doi.org/10.1038/s41534-021-00369-3

The third step is proceed by Hadamard Transform which provides superpositions in the binary base. In this way, Hadamard matrix can be constructed and required transformation can be also applied, as seen in Figure-10, we can obtain configurations which means combination of all possibilities for the binary string of N-length string.

$$|\mathcal{T}\rangle = |t_0 t_1 \ldots t_{N-1}\rangle = \bigotimes_{i=0}^{N-1} |t_i\rangle,$$

$$|\mathcal{P}\rangle = |p_0 p_1 \ldots p_{M-1}\rangle = \bigotimes_{j=0}^{M-1} |p_j\rangle,$$

Figure 10. Niroula, P., Nam, Y. A quantum algorithm for string matching. npj Quantum Inf 7, 37 (2021). https://doi.org/10.1038/s41534-021-00369-3

Finally, cyclic shift operator and XOR is applied. Cyclic-shift is needed when there is a limited and defined length of a string or an array. Compared to the linear shift which shifts every entry to one side (left or right) by the same number of positions, the cyclic-shift moves final entry to the first position, the other entries to the next position, or the first entry to the last position. Thus, a circular shift between entries can be realized. It can be demonstrated on a diagram as follows,

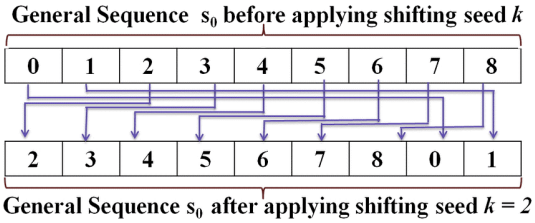


Figure 11

In order to implement a circular shift, the swaps are performed. The N/2 Swap operation in parallel deduces the number of swaps needed by half, each time it is applied. Thus, by using parallel Swap operations, N number of qubits can be permuted in $O(log(n))$ time steps. The cyclic shifted part is the main string, which is symbolized as T, to control matching between patterns, which were sub-strings, in the accordance with various configurations. As seen in Figure-12, these configurations is provided by the leftmost structure. The control for matching is occurred at this point by XOR gates. If XOR table is remembered, corresponding cases are just the case for 0,0 and 1,1; can be seen in the XOR table. Namely, The CNOT implementation into the all structure is control whether matching is occurred or not table for us in the logic of XOR.

$$\begin{aligned} &\left(H^{\otimes n}|0\rangle^{\otimes n}\right)\left[\bigotimes_{i=0}^{N-1}|t_i\rangle\right]\left[\bigotimes_{j=0}^{M-1}|p_j\rangle\right] \\ &= \left(\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}|k\rangle\right)\left[\bigotimes_{i=0}^{N-1}|t_i\rangle\right]\left[\bigotimes_{j=0}^{M-1}|p_j\rangle\right]. \end{aligned}$$

Figure 12. Niroula, P., Nam, Y. A quantum algorithm for string matching. npj Quantum Inf 7, 37 (2021). https://doi.org/10.1038/s41534-021-00369-3

Time Complexity

The time complexity of algorithm is the integral part of the algorithm for computer scientists. The total complexity as mentioned in the article is the following:

$$O(\sqrt{N}((\log(N))^2 + \log(M))).$$

Figure 13. Niroula, P., Nam, Y. A quantum algorithm for string matching. npj Quantum Inf 7, 37 (2021). https://doi.org/10.1038/s41534-021-00369-3

Our string matching algorithm is occurred in the time complexity of quadratic $O(n^2)$ because of nested loop, which can be resembled Bubble sorting in terms of that. However, If quantum computational approach is investigated, time complexity is almost reduced to logarithmic time; of course, there is a factor in the form of square root of n; however, $log(n)$ is upper bound of that.

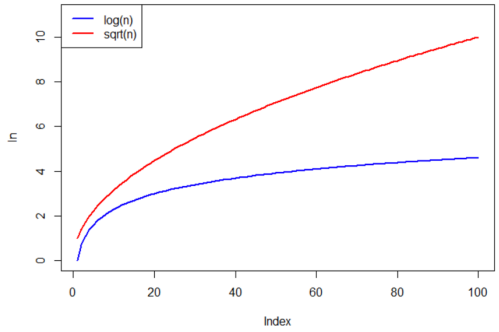


Figure 14. comparison between $O(n^{1/2})$ and $O(log(n))$

```
max_matching=0
optimal_shape=""
for i in list_of_string_shapes:
  j=0
  counter=0
  while j < len(i):
    if i[j] == string_for_ideal_shape[j]:
      counter=counter+1
    j=j+1
  if max_matching < counter:
    max_matching=counter
    optimal_shape=i

print(optimal_shape)
```

Figure 15. Basic String matching algorithm in Python with nested loop which constitutes quadratic complexity, $O(n^2)$.

Obviously, decreasing of time complexity can be observed in this case of string matching. Beside that, the configuration space provide more sample for sub-strings to check in main string which is one of the virtue of our project. More sample put more realistic model for our simulation, thus, statistical inferences from our model and simulation can give more realistic outcomes when it convergences the Nature by expanding sample space. Sample space here is sub-strings; our model encode the wide combination of leaves on bough in the string form. That's why if the configuration space or combinations are expanded, we can compare more model in the form of sub-string with main string which was ideal form of leaves on bough in terms of photosynthesis productivity. That's the convergence of real and choosing optimal one among them properly. In all these implementations, our set of sub-strings and main string which are derived from the model of leaves on bough can be converted to binary string to adapt quantum computational approach above.

Result

As a result of our study, among a sample with 5 different combinations on a classical computer, the one closest to the ideal showed $5/12$ similarity. This means that 2D modeling of a plant species currently existing in nature is only about 40% comparable to the perfection predicted by our mathematical models. These predictions can be slightly improved with better and faster algorithms. Therefore, we have shown that quantum algorithms can process a larger sample in less time, since a larger sample is required for a more accurate prediction.

An Additional Approach to Investigate the Role of Leaf Orientation on Photosynthesis: Quantum Optimization

To examine the effects of leaf orientation on photosynthesis, quantum optimization algorithms are brilliant candidates after some improvements. One may model and demonstrate the endpoints of needles with nodes using NetworkX. By choosing a considerable amount of random points on the main branch and then placing 1000 nodes around every center node, the model of a tree section could be achieved. A simplified version is exemplified below (Figure-15) with 1 center node and weight values of the extended needles.

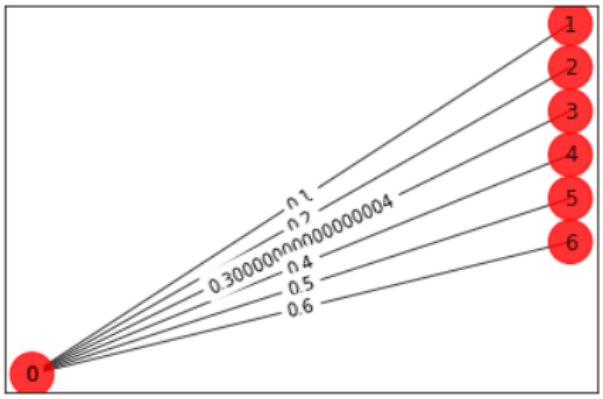


Figure 16

We chose 3 properties while calculating the overall probability, however, one may choose any number of parameters. Parameters which we chose are length, the angle between the needle and the main branch, finally the zenith angle of the normal to the needle surface. In the modeling process, the following steps are implemented. After applying the random module, samples are created. Simultaneously, the length of the needles extending from the main branch and the angle they make should be recorded in a list. Certain coefficients are assigned to these parameters according to the efficiency of their role in photosynthesis. Then, a weight matrix is composed by adding these parameters up to examine which composition has which activity coefficient by investigating the entries of the matrix.

```
w = np.zeros([n,n])
for i in range(n):
    for j in range(n):
        temp = G.get_edge_data(i,j,default=0)
        if temp != 0:
            w[i,j] = temp['weight']
```

Figure 17

```
for ll in range(0,n):
    weigth_list.append(angle_list[ll]*1.27 + length_list[ll]*1.05)


elist = []
for k in range(1,n-9): # 0 is the center node here and the others are extended from the node 0
    elist.append((0,k,weigth_list[k]))
for p in range(n-8,n): # n-9 is the center node here and the others are extended from the node n-9
    elist.append((n-9,p,weigth_list[p]))
```

Figure 18

After the stages above are implemented, one needs to apply optimization to the weight matrix to find which orientation has the minimum cost.

```
min_otl = the_list[0][0] + the_list[0][1]
for entry in range(0,10):
    sum_otl = the_list[entry][0] + the_list[entry][1]
    print(sum_otl)
    if sum_otl <= min_otl:
        gb = [the_list[entry][0],the_list[entry][1]]
        min_otl = sum_otl


print(min_otl,gb)
gamma = gb[0]
beta = gb[1]
print("gamma",gamma)
print("beta",beta)
```

Figure 19

In order to run the system using qubits, a quantum circuit is created and a Hadamard gate is applied to every qubit aiming to create superpositions. Subsequently, Ising gates and rotations are applied and the code is run in the local simulator.

```
# run on local simulator
backend      = Aer.get_backend("qasm_simulator")
shots        = 100

simulate     = execute(QAOA, backend=backend, shots=shots)
QAOA_results = simulate.result()

plot_histogram(QAOA_results.get_counts(),figsize = (8,6),bar_labels = False)
```

Figure 20

Future Directions

Our work can be used to answer deeper questions with a few modifications and extensions to it. By examining the suitability of plant species belonging to the same family or plants from different families with this algorithm, we can reach the orientation of the similarity ratio. If the similarity rate is higher in relatively old species, it can be claimed that evolution can reach the mathematical optimum given enough time, and that every species will eventually reach this optimum after a long period of time. If the ratios are the same, it may mean that biological systems have not evolved to progress to the optimum we predicted and are not increasing their efficiency.

References

1. https://www.extension.iastate.edu/news/yard-and-garden-selection-and-care-christmas-trees

2. Kimes, D. S., Smith, J. A. (1991). Inclination distributions and size measurements of hemlock and red spruce needles. Applied optics, 30(36), 5226-5228

3. Kriedeman, P. E., Neales, T. F., Ashton, D. H. (1964). Photosynthesis in Relation to Leaf Orientation and Light Interoeption. Australian Journal of Biological Sciences, 17(3), 591-600.

4. Niroula, P., Nam, Y. A quantum algorithm for string matching. npj Quantum Inf 7, 37 (2021). https://doi.org/10.1038/s41534-021-00369-3.

5. https://ieeexplore.ieee.org/document/8113570

Sample

```
Enter a point: 1 2
x for initial point:  1
y for initial point:  2
Enter a point: 4 7
x for final point:  4
y for final point:  7
Random x-coordinate on main line:  4
Random y-coordinate on main line:  7.0
Enter an angle(C): 30
Enter a length(m): 1.2
Random x-coordinate on main line:  1
Random y-coordinate on main line:  2.0
Enter an angle(C): 4
Enter a length(m): 0.7
Random x-coordinate on main line:  3
Random y-coordinate on main line:  5.333333333333333
Enter an angle(C): 77
Enter a length(m): 0.9
coordinates of leaf:  4.020183917779878 -- 8.199830241935523
coordinates of leaf:  5.028991510855053 -- 6.382605093486968
main:  4 - 7.0
coordinates of leaf:  0.3653975417080333 -- 2.295431413241369
coordinates of leaf:  1.1099414495118283 -- 1.3086875686932586
main:  1 - 2.0
coordinates of leaf:  2.2455067088706757 -- 4.842680835489043
coordinates of leaf:  3.8561266201964504 -- 5.055759540998748
main:  3 - 5.333333333333333
```

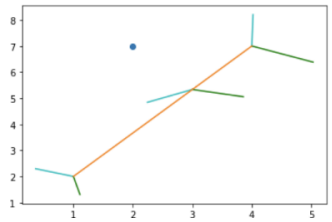
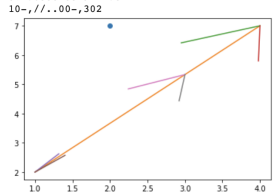


Figure 21. The Figure shows that taken inputs and one of the form of random generated leaves on bough model with coordinates above

```
['0//.0/..00/.', '1//-00/.00/.', '10.-02/.05..', '10/.00/.05/.', '07..0///00.-']
3.4705882352941178
10-,//..00-,302
```



```
0//.0/..00/.
```

Figure 22. The Figure shows that set of sub-strings and sketched form of ideal one with its string.