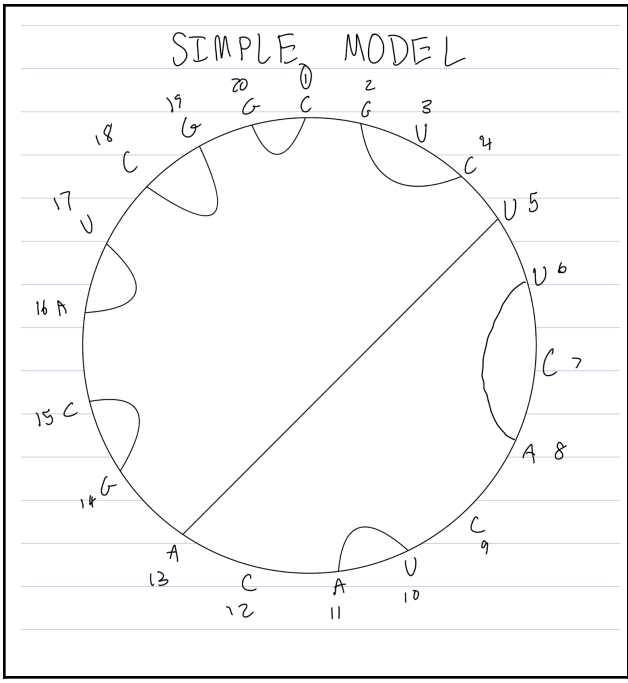
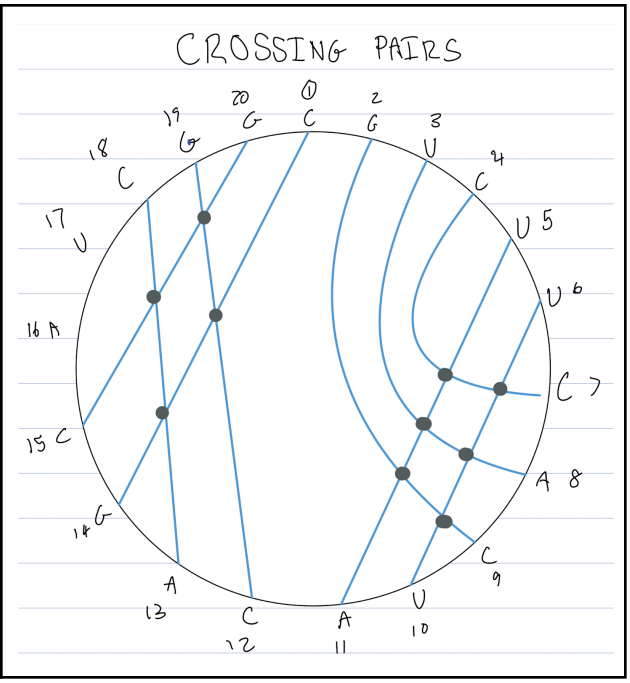
Lokranjan Lakshmikanthan
Nikita Jakkam
Sunny Patel
4/25/2022
ISYE 3133

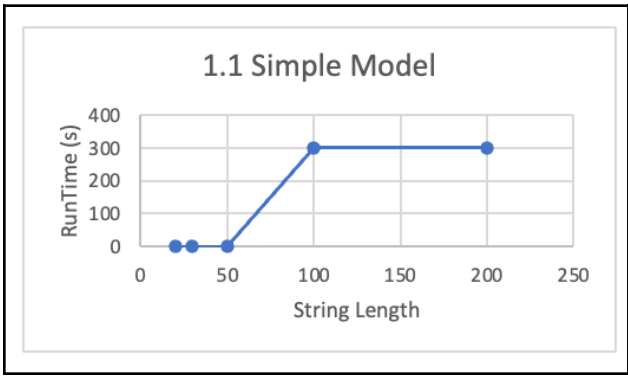
Final Report

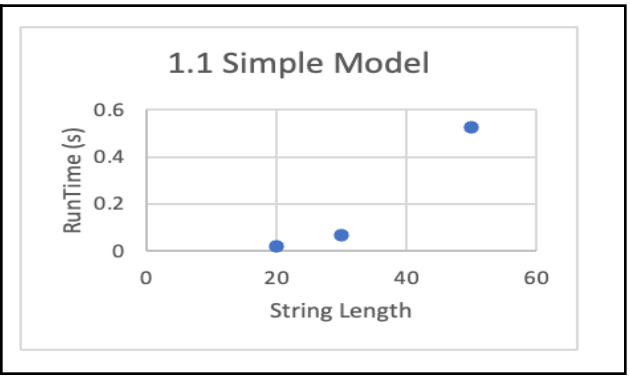
Our group decided to implement all of the models. We have six files where the simple_model represents the model in 1.1, simple_enhancements represents the model in 1.2, base_stacking represents the model in 1.3.1, weighted_stacked is the model in 1.3.2, position_quartet is the model in 1.3.3, and crossing_10 is the model in 1.3.4. We ran our code on Jupyter Notebook so the .ipynb and .py files are attached. Below are pictures of the optimal solution for test case 1 with the simple model 1.1 and the crossing pairs model 1.3.4.

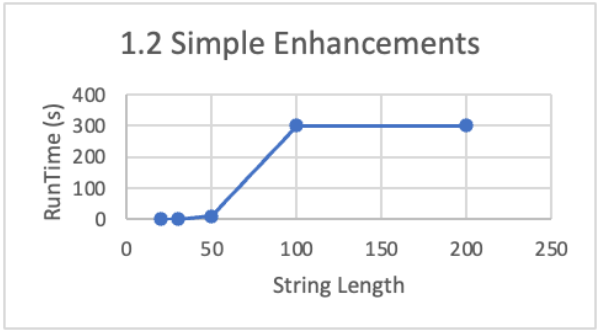


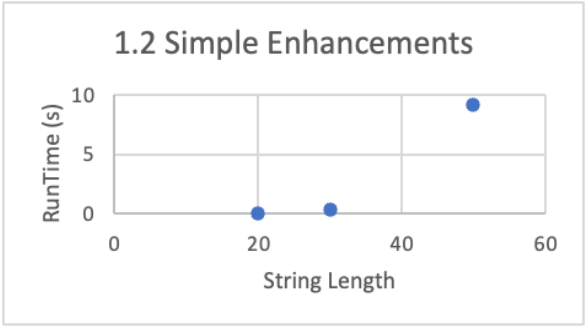


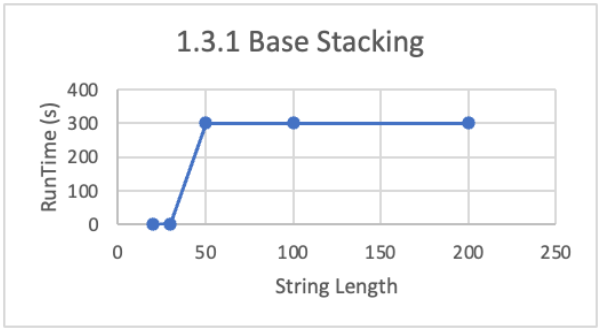
Size vs Runtime

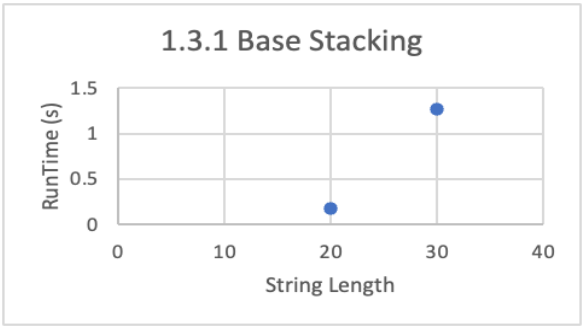


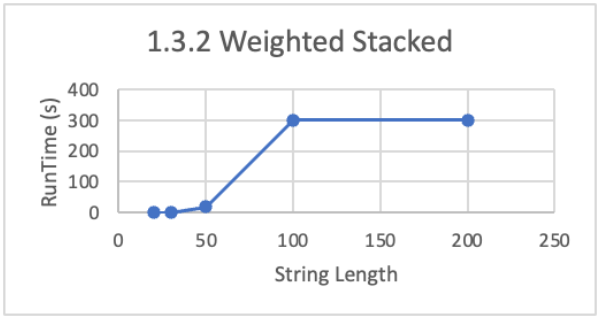


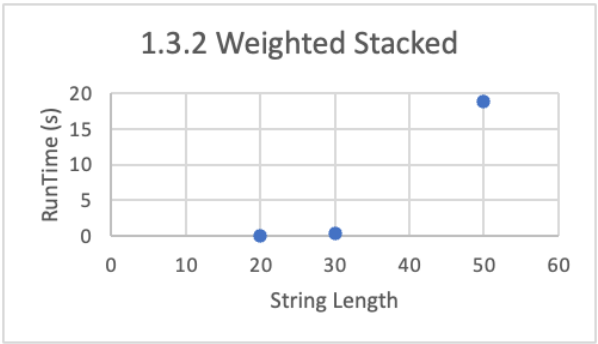

1.2 Simple Enhancements
RunTime (s)
400
300
200
100
0
0
50
100
150
200
250
String Length

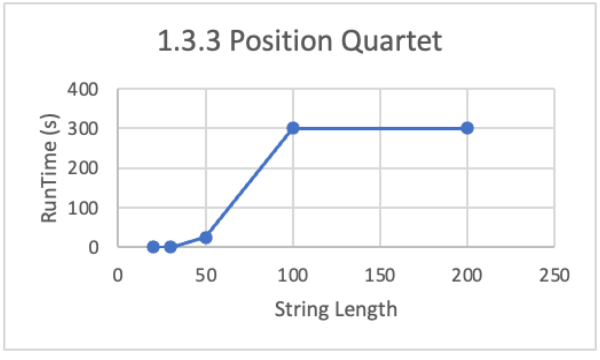

1.2 Simple Enhancements
RunTime (s)
10
5
0
0
20
40
60
String Length

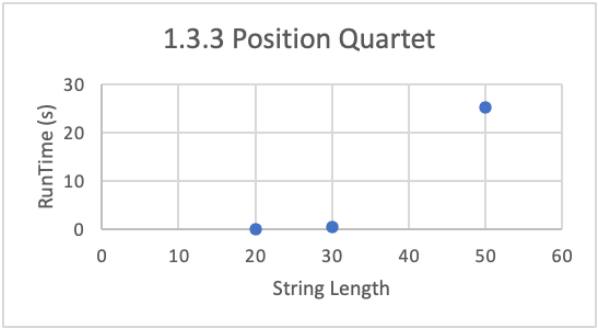

1.3.1 Base Stacking
RunTime (s)
400
300
200
100
0
0
50
100
150
200
250
String Length

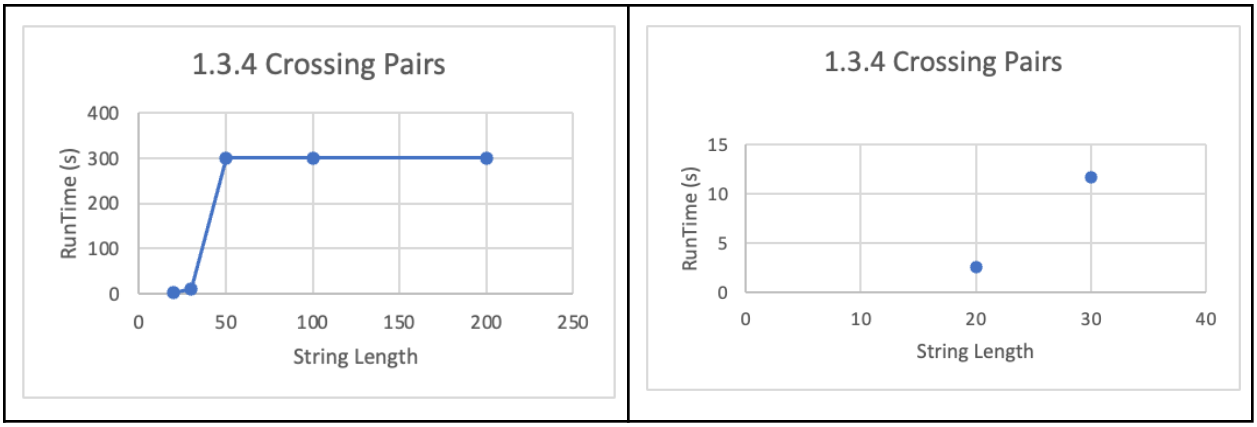

1.3.1 Base Stacking
RunTime (s)
1.5
1
0.5
0
0
10
20
30
40
String Length


1.3.2 Weighted Stacked
RunTime (s)
400
300
200
100
0
0
50
100
150
200
250
String Length


1.3.2 Weighted Stacked
RunTime (s)
20
15
10
5
0
0
10
20
30
40
50
60
String Length


1.3.3 Position Quartet
RunTime (s)
400
300
200
100
0
0
50
100
150
200
250
String Length


1.3.3 Position Quartet
RunTime (s)
30
20
10
0
0
10
20
30
40
50
60
String Length



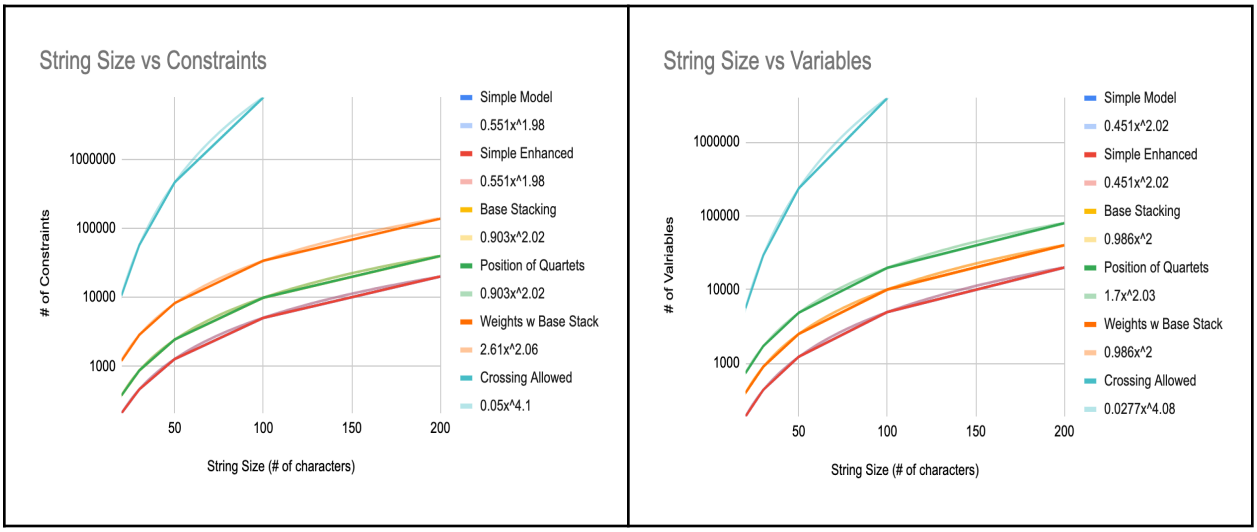
The plots above show the running time (in seconds) as the size of the input increases for each model. The plots on the left show the running time for strings with lengths 20, 30, 50, 100, and 200. We set the timeout to 5 minutes and noticed that most strings with 50 characters or more take more than 5 minutes to run. The plots on the right show strings that take less than five minutes to run for each model. The runtime for the model increases quickly with respect to string length at a seemingly large nonlinear rate.

Objective Values Across Models and Test Cases

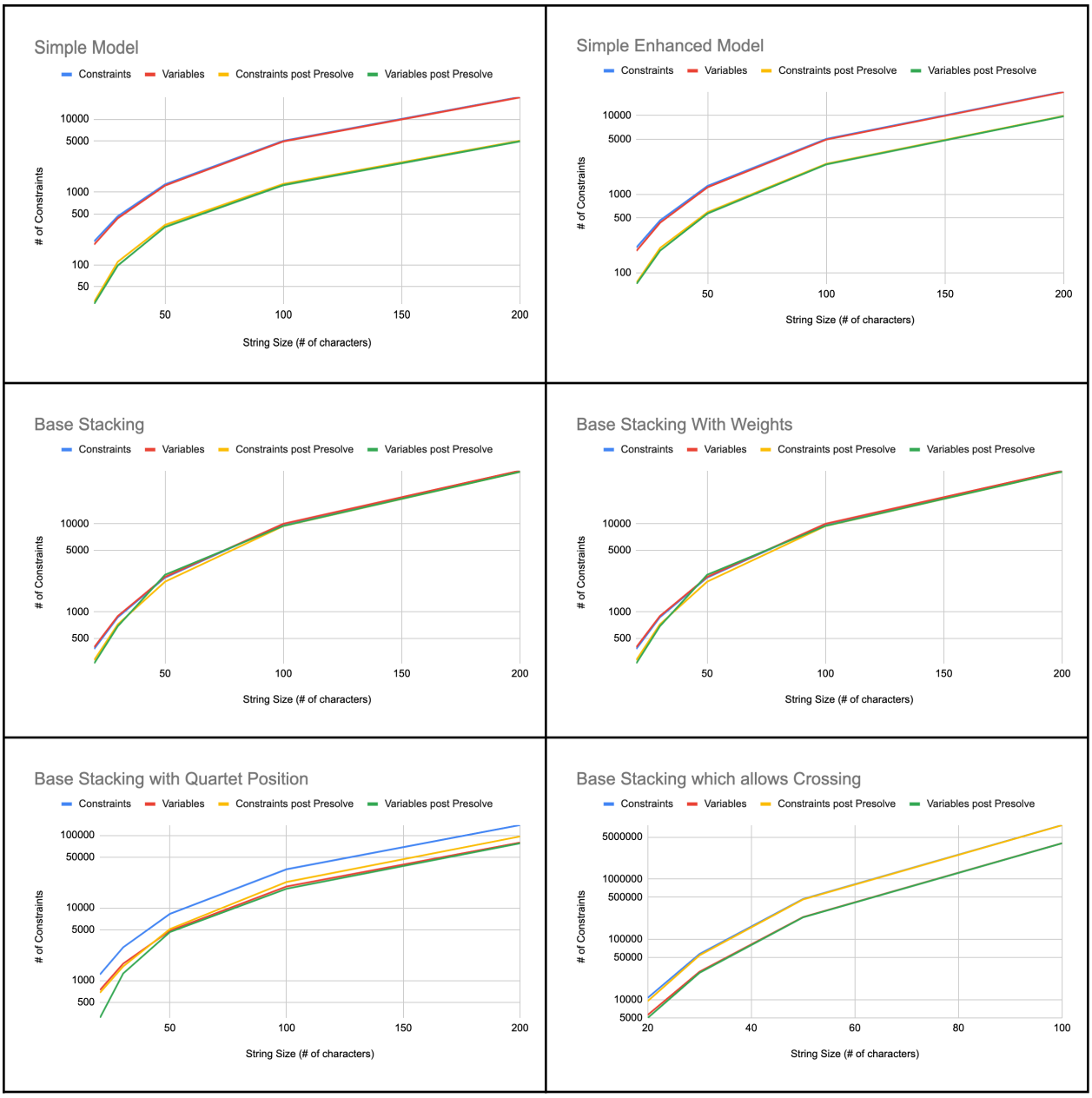
	Test Case 1 (n=20)	Test Case 2 (n=30)	Test Case 3 (n=50)	Test Case 4 (n=100)	Test Case 5 (n=200)
Model 1.1	8	12	21	44	83
Model 1.2	13	23	45.15	86	62.75
Model 1.3.1	17	29.15	57.25	89.2	65.75
Model 1.3.2	60	137.1	291.1	547	65.75
Model 1.3.3	107	250.1	532.1	822.15	68.75
Model 1.3.4	112.05	280.1	504.1	207.3	n/a

The table above shows the optimal solution we found for each model and each test case. We noticed that for each model the optimal solution increases as the string length increases except for the last test case where the string length is 200. The string length for the last test case was a lot larger than the other test cases. We think the optimal solution for the last test case is not accurate because it did not have enough time to run. Once the string lengths started increasing to 100 and 200, the model would take a lot longer to process the string and reach the maximum time of 300 seconds. For Model 1.3.4, we were unable to find an optimal solution due to the length of the string and the size of the formulation.
We also noticed that the optimal value increases as we increase the complexity of the model.
This is a relatively trivial insight as the objective function for each model is much more likely to increase in comparison to a simpler model simply because the objective is a sum of more positive values for each consecutive value.
Model Size and Scaling

We wanted to take a data-based and graphical approach to understanding how the size of each model was affected by the string input size. We recorded the values that describe model size and graphed them as seen below.



The graphs indicate some very interesting relationships. The number of variables and constraints created for all models are very similar in value. We found that some of the updated or more complex models had barely if any increase in size to their precursor. For example the Simple Crude model has the same number of variables and constraints as the simple enhanced model. This is because the only difference between them is the addition of new parameters that affect the objective function. Similarly the model that introduces Base Stacks has the same size as the model with weighted base stacks. However there are clear increases in the size of the model between the simple model compared to that with base stacks compared to that which includes quartet position. The largest model by far is the one that allows crossing. Graphing these values allowed us to find a simple power regression line that fits the data. The functions that represent the regression line for each model are written in the key directly below the model. We see that while most of the models are polynomials of power two with respect to string length, the model that allows crossing has a polynomial regression line of power four. Since the model size is described by a quartic polynomial, an increase in input size for the crossing model results in a much larger model when compared to the other models.



We were also curious to better understand how the model size changes for each model type in terms of actual complexity. We thought that the size of the model after presolve could lead to some interesting insights. The difference between the size of the model before and after presolve indicates to some degree the actual complexity of the model in terms of computational expense. If the size of the model after presolve is significantly smaller, this indicates that the formulation of the model contains more trivial variables or constraints. The size of the model after presolve can in some ways be interpreted as the “True” indicator of the model’s computational complexity. We can notice some interesting patterns here. These patterns could entirely be a product of the test cases chosen, but since the relationships seem to hold for all 5 test cases, that seems unlikely. The simple and simple enhanced models are significantly smaller after presolve. We see here that while the model sizes are the same prior to presolve for both models, the post presolve size of the simple with enhancements is consistently larger than the initial crude

model. Thus the simple model with enhancements is obviously slightly more computationally expensive in comparison. We find an interesting relationship with the Base Stacking and Base Stacking with Weights models where the presolve seems to barely reduce the size of the model. This indicates that there are very few if any trivial decision variables or constraints. Presolve reduces the number of constraints on the model with Quartet position included but has no significant effect on the number of variables as input size increases. Similar to all the models that include base stacking, presolve results in very minimal reduction of the model that allows crossing. This contributes to the computational expense of the crossing inclusive model. Since there is no reduction in the size of the large formulation, runtime is likely to increase at a large non-linear rate.

Complex Constraints

We believe that the most complicated constraint to solve is the non-crossing constraint and the allowing 10 crossing pairs constraint. These constraints had the most amount of for loops in order to implement them. Since there are four nested for loops, runtime complexity is becomes $O(N^4)$. Furthermore these sets of constraints contain the most constraints. Since there are more constraints these sets of constraints are the most constrictive on the possible solution space. Jupyter notebook allowed us to see how long each code block took to run and we noticed that as the length of the test cases increased, the code cells with the non-crossing constraint and allowing 10 crossing pairs constraint took the longest to run. We were unable to see the difference with the smaller test cases as they were able to run relatively quickly.

Model Updates

When we were creating the model in phase 1 of the project we had a lot of decision variables and constraints for the simple model and the simple enhancements model. Towards the end we realized that it would be best to change our model since our old approach would not be scalable to more intense constraints. Our approach at the end of phase 1 matches the approach in the solution however we still had mistakes with many of the constraints. After looking at the solution we realized that our original model would take a long time to run and we decided to go with the solution since it was easier to implement and it was more concise. The solution and our code should match as we didn't make changes to how we implemented the solution of the project.

Conclusion and Next Steps

The objective is not super insightful with respect to understanding the relative stability of a nucleic acid with each consecutive model because although a larger objective is obviously better, it does not give us a lot of information about how stable a nucleic acid is relative to its modeling size and constraints. It would be interesting to find out what the maximum possible objective value could be for a string of some fixed length S and compare the actual objective for some optimal folding of a specific nucleic acid with fixed length S. Some ratio or comparison of these values gives us some quantitative understanding of the relative stability of the specific nucleic acid at its most optimal folding arrangement. This understanding of relative stability has biochemical implications that will be useful for practical applications.