ISyE 3133 Project (Final Report)

Sydney Bice, Johanna Lumbantobing

April 19, 2024

1 Introduction

This project involves formulating multiple integer linear programming models to address the Nucleic Acid Folding Problem. The Nucleic Acid Folding Problem involves predicting the secondary structure of a nucleic acid molecule, represented as a circular string of nucleotides, by identifying the most likely pairings of these nucleotides. Additionally, each nucleotide can be in at most one pair. The objective of this project is to model various scenarios to predict the pairing of a circular string of nucleotides given different assumptions and constraints for each model.

2 Model Changes from First Report

Model 1: We corrected our Single Pairing Constraint

$$\sum_{j=i+1}^{n-1} x_{ij} \leq 1 \quad \forall i = 0, ..., n-1 \tag{1}$$

In our code, we handled the circular nature of the nucleotide sequence by having the variable l wrap around to the beginning of the sequence if it exceeds n-1.

Model 2: In our code, we specified which pairings were allowed in the model. As for our indices in this model, as well as models 3 and 4, we changed them to specify that i begins at 0 and the summation continues until n - 1. Lastly, we corrected our distance constraint.

Model 3: Changed size of formulation. In our code, we used the corrected code for Model 2 and ensured the Stacked Quartets Constraint was correct.

Model 4: Previously in Model 4, the constraints were all taken from Model 3 with the addition of limiting the crossing pair amount and setting the c_{ijkl} variable to 1 where needed. However, the non-crossing pairing constraint and the existence of crossing pairs conflict with each other. The non-crossing pairing constraint was removed.

3 Solutions

Model 1 Solution

- ACGUGCCACGAU (String 0): (0,11), (1,2), (3,7), (4,5), (8,9)
- CUUGGCUGGAAACGUAAGUA (String 1): (0,17), (1,16), (2,10), (4,5), (6,9), (12,13), (14,15), (18,19)
- GCAUAUGGUCGACGCCUUCAAUAACGAUAC (String 2): (0,29), (2,3), (5,26), (6,15), (7,12), (8,11), (9,10), (13,14), (16,20), (17,19), (21,22), (24,25), (27,28)
- AGGUACGCCGCUAGAGCGAACCGGGCACCAGCUACGCCGU (String 3): (0,39), (1,34), (2,5), (3,4), (6,7), (8,9), (10,13), (11,12), (15,27), (16,23), (17,20), (21,22), (24,25), (28,30), (32,33), (35,36), (37,38)

String	Time (s)	Objective	Number of AU pairs	Number of GC pairs
1	0.02785	5	2	3
2	0.03933	8	5	3
3	0.06013	13	7	6
4	0.14429	17	4	13

Table 1: String Solutions for Model 1

Model 2 Solution

- ACGUGCCACGAU: (0,11), (1,9), (2,8), (3,7)
- CUUGGCUGGAAACGUAAGUA: (0,13), (1,10), (2,9), (5,8), (14,19), (15,18)
- GCAUAUGGUCGACGCCUUCAAUAACGAUAC: (0,29), (3,28), (4,27), (5,26), (6,14), (7,12), (8,11), (15,25), (16,23), (17,20)
- AGGUACGCCGCUAGAGCGAACCGGGCACCAGCUACGCCGU: (0,3), (4,39), (5,38), (6,37), (7,23), (8,22), (9,16), (10,15), (11,14), (17,20), (24,36), (25,35), (26,32), (27,30)

String	Time (s)	Objective	AU Pairs	GC Pairs	GU Pairs	AC Pairs
1	0.0387570858	10	2	2	0	0
2	0.05630707741	14	4	2	0	0
3	0.3443894386	24	6	4	0	0
4	6.667886019	38	4	10	0	0

Table 2: String Solutions for Model 2

Model 3 Solution

- ACGUGCCACGAU: (0,11), (1,9), (2,8), (3,7)
- CUUGGCUGGAAACGUAAGUA: (0,17), (1,16), (2,15), (3,14), (4,12), (5,11), (6,10)
- GCAUAUGGUCGACGCCUUCAAUAACGAUAC: (0,29), (3,28), (4,27), (5,26), (6,14), (7,12), (8,11), (15,25), (16,23), (17,22)
- AGGUACGCCGCUAGAGCGAACCGGGCACCAGCUACGCCGU: (0,39), (1,37), (2,36), (5,35), (6,34), (7,33), (8,17), (9,16), (10,15), (11,14), (18,32), (19,31), (20,30), (21,29), (22,28), (23,27)

String	Time (s)	Objective	AU Pairs	GC Pairs	GU Pairs	AC Pairs	Stacked Quartets
1	0.02338	12	2	2	0	0	2
2	0.04861	17.15	3	2	1	1	5
3	0.3822	28	6	4	0	0	4
4	2.198	47.15	3	10	0	3	11

Table 3: String Solutions for Model 3

Model 4 Solution

- ACGUGCCACGAU: (0,11), (1,9), (2,8), (3,7)
- CUUGGCUGGAAACGUAAGUA: (0,17), (1,16), (2,15), (3,14), (4,12), (5,11), (6,10)
- GCAUAUGGUCGACGCCUUCAAUAACGAUAC: (0,29), (3,28), (4,27), (5,26), (6,14), (7,12), (8,11), (15,25), (16,23), (17,22)
- AGGUACGCCGCUAGAGCGAACCGGGCACCAGCUACGCCGU: (0,39), (1,37), (2,36), (5,35), (6,34), (7,33), (8,17), (9,16), (10,15), (11,14), (18,32), (19,31), (20,30), (21,29), (22,28), (23,27)

String	Time (s)	Objective	AU Pairs	GC Pairs	GU Pairs	AC Pairs	Stacked Quartets	Crossing Pairs
1	0.04234	15.05	2	3	0	1	2	6
2	0.1620	21	5	3	0	0	2	10
3	9.594	35.1	6	6	0	2	5	9
4	82.93	54.05	4	13	0	1	7	10

Table 4: String Solutions for Model 4

4 Running Time Analysis

4.1 Model 1

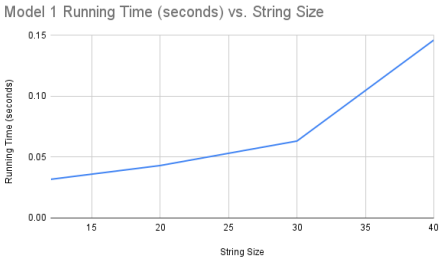

Figure 1: Runtime (s) vs String Size

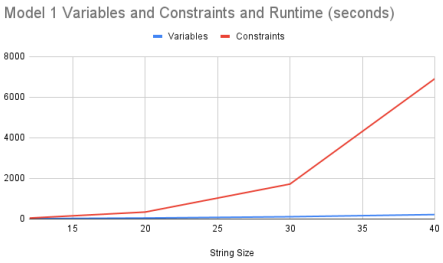

Figure 2: Runtime (s) vs Variables and Constraints

4.2 Model 2

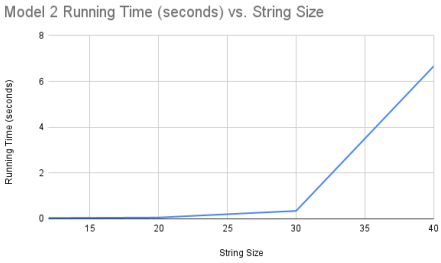

Figure 3: Runtime (s) vs String Size

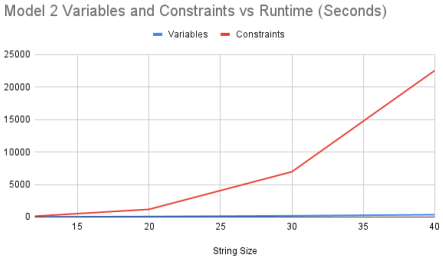

Figure 4: Runtime (s) vs Variables and Constraints

4.3 Model 3

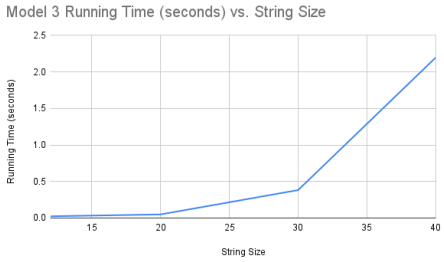

Figure 5: Runtime (s) vs String Size

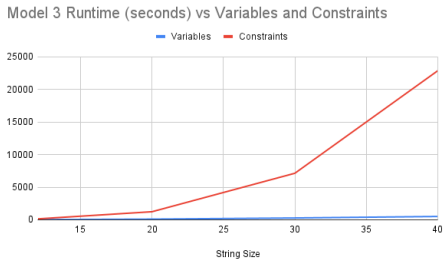

Figure 6: Runtime (s) vs Variables and Constraints

4.4 Model 4

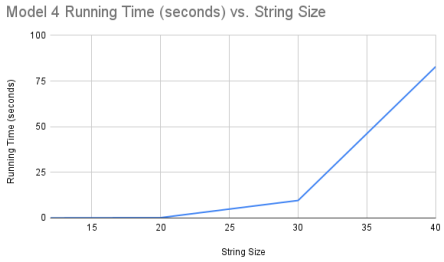

Figure 7: Runtime (s) vs String Size

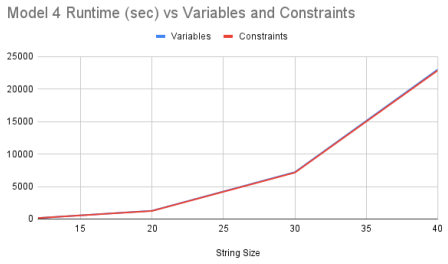

Figure 8: Runtime (s) vs Variables and Constraints

5 Conclusion

For the constraints, the required distance of 3 was difficult to solve. We attempted to solve this problem by creating a new constraint in our code, but we found it easier to limit the X variables created by this constraint. Another roadblock that we encountered was implementing the y_{ij} variable and the constraints that went along with it. Similar to before, it was simpler to restrict which Y variables were created based on the X variable and the possible pairing in the sequence.

There were a few occasions where we updated our model whilst creating the code. As mentioned before, some constraints were handled better earlier during the variable creation. At first, we attempted to create the constrains themselves, but we were not getting our desired results. Otherwise, our process of fixing our mistakes on paper first, and then starting to code it worked in our favor.