Nucleic Acid Folding - Implementing a MIP Approach

Quill Healey
ISYE 3133 - Engineering Optimization - Final Project

April 25, 2022

1 Overview

This report will focus on the implementation and subsequent analysis of the nucleic acid folding problem. However, while the mathematical formulation did include a "most-realistic model," so to speak, it is important due to both the nature of the problem, as well as the nature of LPs, to implement "submodels" of this best approximation. To be precise, we will implement the following models (written in order of increasing biological complexity): a crude model, a simple enhanced model, a model incorporating quartets, the three prior models all allowing for ten crosses, and finally a "full model," which includes the simple enhancements as well as quartet considerations and a crossing allowance of ten hydrogen bonds. After implementing these seven models using Python and the optimization software Gurobi, we will analyze each by comparing not only their solutions (or Objective Bounds/MIP gaps), but also their total size and runtime complexity.

2 Solutions

2.1 High Level

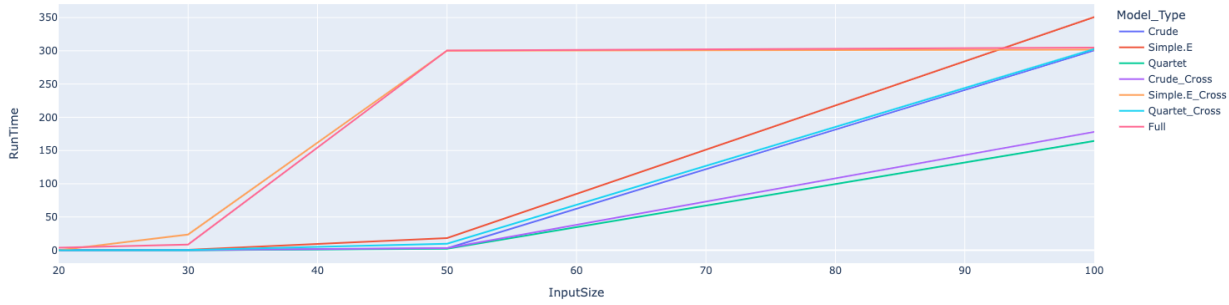
We begin our analysis by examining the solutions we obtain to the varying models (*See Appendix for output*). With a maximum run time of five minutes, only the nucleotide sequences containing twenty, thirty, and fifty characters could be solved by every model. Starting with the nucleotide sequence of length one-hundred we had to settle for objective bounds and MIP gaps for all models except for the quartet model (based on the crude model, allowing no crossings, and taking into account quartet weightings and position). The fact that we could find a solution for this particular biologically enhanced model is actually very interesting and speaks to the complexity of the varying constraints, which we will dive into further below.
Finally, I must note that I was unable to compute any of the models for the nucleotide sequence of length two-hundred. After running the code for over five hours my jupyter notebook kernel force quit the program.

2.2 Low Level

Before continuing with a bird's eye analysis of the general folding problem, let's briefly pause and convince ourselves that our models are accurate. The easiest way to make this reality check is to run the least and most complicated models on an input string that the human eye can make sense of. Thus, constructing the aforementioned models with the sequence $S = CGUCUUCACUACAGCAUCGG$, we find that the Crude Model returns the following eight pairs: $S[1, 20], S[2, 15], S[4, 14], S[5, 13], S[6, 8], S[10, 11], S[16, 17], S[18, 19]$. This is indeed a pleasing result as, upon drawing the prescribed representation (which I suggest the reader do), we see that no hydrogen bonds cross, no base acid pairs with itself, and all pairs are complementary. Now, moving on to the full model, we get the following nine pairs: $S[1, 14], S[2, 9], S[3, 8], S[4, 7], S[5, 11], S[6, 10], S[12, 19], S[13, 18], S[15, 20]$. Once again, if the reader is to draw this representation, it becomes apparent that this model does indeed follow its theoretical basis, incentivizing quartet stacking and allowing (in this case) exactly ten crosses. (*The reader should also examine the code output in case I mistranslated the pairs above transferring the indices from Python's zero-based numbering*).

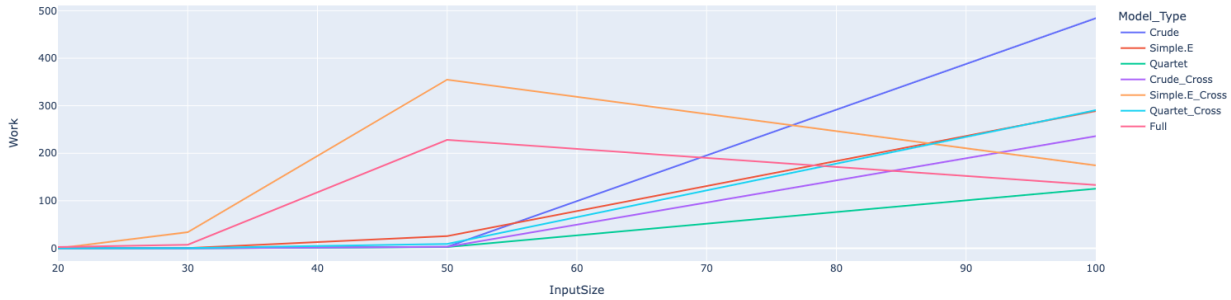
3 Graphs

As we are solving this nucleic acid folding problem with a computational technique, one of the statistics we care most about is run time complexity. That is, how much longer does it take for our model to run as we increase the length of the input nucleic acid sequence. We can approximate this run time analysis by observing the following graph, keeping in mind that no model was allowed to run for longer than five minutes.



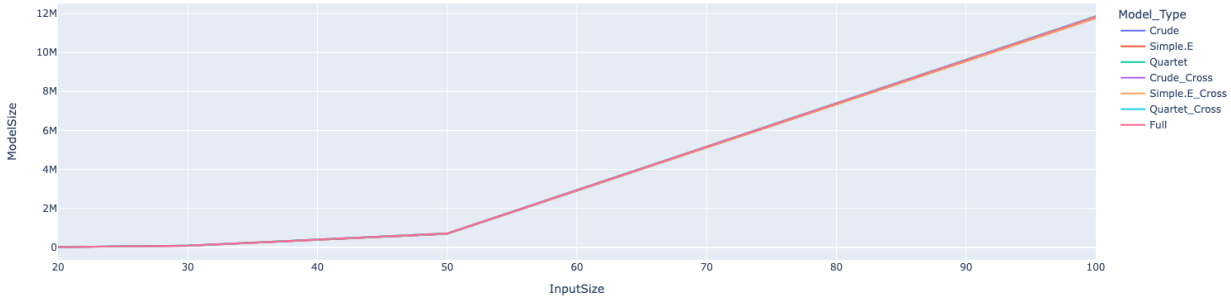
We can draw two important conclusions from this plot. Firstly, the simply enhanced model (both with and without a crossing allowance) appears to be the most challenging model to solve as they are the first to be constrained by the five minute run time constraint. Secondly, run time across all models does in fact increase as the size of the model increases (albeit at varying rates).

While this input size vs run time graph is informative, we can somewhat standardize these results by using the Gurobi model's built in "work" attribute, which is defined as the "[w]ork spent on the most recent optimization," where work is deterministic with arbitrary units that roughly follow: one work unit is roughly equivalent to "one second of runtime on a single thread."



Examining this plot with the run time plot above leads to some interesting insights - namely that while the simply enhanced model does appear to take the longest time to solve, the *crude* model appears to be increasing in work units! Additionally, the quartet model, one of the more biologically complex representations of the nucleic acid folding problem, appears relatively easy to solve! We will address all of these observations in the following section.

The final graph that we will use in our analysis plots the model size, defined as the number of decision variables plus the number of constraints, versus the number of characters in the nucleic acid sequence.



This result is rather fascinating, and I once again refer the reader to the appendix to observe the raw output that comprises this plot. Obviously, and as one would expect, the size of the model increases as the size of the input sequence increases. Surprisingly, however, it appears that all models are of the same size! I am somewhat skeptical of this result, however, and have an alternative hypothesis. I believe that the sizes that Gurobi returns for the models are the *pre-solved* sizes. Or rather, the size of the model after eliminating the unnecessary decision variables and constraints. I'd even go as far as to claim that the remaining features of the model are the basic feasible solutions. Of course this alternative hypothesis does still give an interesting insight into our seven models.

4 Constraint Analysis

Now, a nimble-minded reader at this point should be champing at the bit to make sense of the somewhat-convoluted plots. It is indeed confusing as to why the simply enhanced model seems to be the most challenging to solve, and there's also the crude model with it's

largest number of work units for the one-hundred length nucleotide sequence to make sense of. Firstly, we can go ahead and dismiss this odd result involving the crude model as it is merely a consequence of Gurobi going down a rabbit hole attempting to make the crude model hundredths of a unit better, while it was still stuck with larger computations for the more complicated models (one only needs to examine the MIP gap in the output data to confirm this).
The more interesting dissection, of course, lies in why the simply enhanced model appears to be more difficult to solve than the full model, which is exactly the simple model *plus* considerations for quartets. Therefore, why is the former apparently more difficult to solve than the latter? The answer to this question reveals itself when we incldue one more model in our analysis: the quartet model (one should make sure they recall the difference between this model and the full model). By examining the quartet model, which is the easiest to solve, we can takeaway this final result: the quartet constraints themselves must be easily computed by the simplex method, allowing for an easily found solution. On the other hand, the simple enhancements, while less biologically significant than the quartet considerations, are harder to solve (these enhancements being the minimum distance between two bases in two arbitrary pairs and allowing for non-complementary bases). Consequently, the difficulty of solving the full model, a combination of the other two models, falls between said models. As a final note, the allowing for crosses appears to have mixed, and somewhat random affects on each type of model.

5 Deviations from the Mathematical Formulation

While I speak more to the computational implementation below, I will point out now that this program follows the MIP formulation as proposed by the TAs. Meaning, I did not define the decision variables (DVs) according to the relation I constructed in my own formulation, which was close to being a Cartesian product of all the basis, but instead implemented the much simpler DVs which only allow one hydrogen bond from (i, j). With that being said, I did deviate from the TA model slightly with respect to the quartet and quartet position DVs, as their respective constraints referenced decision variables that did not exist. Therefore, I went ahead and created the quartet and quartet position DVs with indices corresponding to the Cartesian product of all base positions, and then added additional constraints to ensure the excess DVs were not utilized by the model.

6 Appendix

6.1 Raw Output

Model_Type	ObjVal	ObjBound	MIPGap	InputSize	ModelSize	RunTime	Work
Crude	8.00	8.00	0.000000	20	14888	0.009272	0.009541
Simple.E	13.00	13.00	0.000000	20	14786	0.030666	0.025634
Quartet	123.00	123.00	0.000000	20	17949	0.017882	0.011058
Crude_Cross	8.00	8.00	0.000000	20	14888	0.018417	0.015989
Simple.E_Cross	20.00	20.00	0.000000	20	14786	0.121969	0.133317
Quartet_Cross	124.00	124.00	0.000000	20	17949	0.029326	0.023406
Full	112.05	112.05	0.000000	20	17847	3.893781	2.627289
Crude	12.00	12.00	0.000000	30	83006	0.088095	0.078481
Simple.E	23.00	23.00	0.000000	30	82741	0.529652	0.497914
Quartet	236.00	236.00	0.000000	30	89997	0.100903	0.083980
Crude_Cross	13.00	13.00	0.000000	30	83006	0.173608	0.138569
Simple.E_Cross	32.00	32.00	0.000000	30	82741	23.674621	33.699167
Quartet_Cross	261.00	261.00	0.000000	30	89997	0.406573	0.376760
Full	280.10	280.10	0.000000	30	89732	8.619098	7.586907
Crude	21.00	21.00	0.000000	50	693063	2.892717	3.232931
Simple.E	45.15	45.15	0.000000	50	692276	18.229978	25.407161
Quartet	596.00	596.00	0.000000	50	712714	2.204767	2.398372
Crude_Cross	22.00	22.00	0.000000	50	693063	3.037931	3.203672
Simple.E_Cross	53.10	58.10	0.094162	50	692276	300.089571	355.036257
Quartet_Cross	664.00	664.00	0.000000	50	712714	9.898969	9.005543
Full	545.10	648.60	0.189873	50	711927	300.531339	228.285889
Crude	44.00	46.00	0.045455	100	11772434	300.599831	484.184758
Simple.E	76.05	115.00	0.512163	100	11768926	350.722601	288.966640
Quartet	1039.00	1039.00	0.000000	100	11851735	164.320815	125.219391
Crude_Cross	46.00	46.00	0.000000	100	11772434	178.026844	236.192178
Simple.E_Cross	61.30	115.00	0.876020	100	11768926	301.720668	174.465313
Quartet_Cross	1081.00	1468.00	0.358002	100	11851735	302.655902	290.554296
Full	207.30	34744.85	166.606609	100	11848227	304.856509	133.002959

6.2 Overview of Program

A reader who wishes to understand the computation of these results should have a high level understanding of the program that I wrote. Note that as the more biologically complex models do build on the more simple models, I take advantage of Python being an object oriented language. In particular, I heavily utilize inheritance. The program immediately declares an over-arching parent class, "FoldingModel" (FM), which establishes the constraints

common to each model (the crossing constraint and minimum distance constraint, to name a couple). I then declare four children classes to FM, each with their own intricacies that create a biologically unique nucleic acid folding model.