Summer 2015

BIV TAR RNA Binding Glycine Mutant Tat Peptides: An Integrated Modeling and Binding Assay Approach

Loc Tien Nguyen
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Nguyen, Loc Tien, "BIV TAR RNA Binding Glycine Mutant Tat Peptides: An Integrated Modeling and Binding Assay Approach" (2015). *Master's Theses*. 4604.
DOI: <https://doi.org/10.31979/etd.bt53-x6as>
https://scholarworks.sjsu.edu/etd_theses/4604

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

BIV TAR RNA BINDING GLYCINE MUTANT TAT PEPTIDES: AN
INTEGRATED MODELING AND BINDING ASSAY APPROACH

A Thesis

Presented to

The Faculty of the Department of Chemistry

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

by

Loc Tien Nguyen

August 2015

© August 2015

Loc Tien Nguyen

ALL RIGHTS RESERVED

The Designated Thesis Committee Approves the Thesis Titled

BIV TAR RNA BINDING GLYCINE MUTANT TAT PEPTIDES: AN
INTEGRATED MODELING AND BINDING ASSAY APPROACH

by

Loc Tien Nguyen

APPROVED FOR THE DEPARTMENT OF CHEMISTRY

SAN JOSÉ STATE UNIVERSITY

August 2015

Dr. Brooke Lustig Department of Chemistry

Dr. Elaine Collins Department of Chemistry

Dr. Lionel Cheruzel Department of Chemistry

ABSTRACT

BIV TAR RNA BINDING GLYCINE MUTANT TAT PEPTIDES: AN INTEGRATED MODELING AND BINDING ASSAY APPROACH

by Loc Tien Nguyen

Interactions between viral encoded regulatory proteins and RNA target sequences control gene expression of Lentiviruses, including human immunodeficiency virus (HIV). Bovine immunodeficiency virus (BIV) provides a simpler model of interaction between the viral trans-activating protein (Tat) and trans-activation response RNA element (TAR), using Tat peptides binding to TAR RNA fragments. The resulting characterization of the hinge region of native BIV TAR-Tat complex was confirmed by more comprehensive calculations, involving an exhaustive generation of lattice chains. This modeled 2-residues per move of the native 11-mer Tat peptide and a 28-nucleotides TAR fragment. But these sorts of coarse-grained calculations, upon substitution of Gly at key hinge region positions, are not fully sensitive to the local flexibility of amino acid side chains optimized for packing and possible interaction with relevant all-atom RNA structure. An overall binding destabilization effect is indicated for the single substitution at 78 and double substitution of Gly at positions 75 and 78. Destabilization effects were further examined, and model data showed that it included both potential and flexibility effects. Future studies require 1-residue per move approach and building all-atom models to fully examine molecular interactions of TAR-Tat complexes.

ACKNOWLEDGEMENTS

There are three main individuals whom I would like to offer my deep gratitude for helping me throughout my time at the Department of Chemistry at San Jose State University. First and foremost, I would like to thank Dr. Brooke Lustig, my research advisor, for mentoring me during my research projects and the writing of my Masters thesis. He provided me with his patience, knowledge, and expertise for the completion of this project. Most importantly, he gave me a chance to become a more scientific, logical, and critical thinker, which will definitely be helpful for my next career journey. I would also like to thank my committee members, Dr. Lionel Cheruzel and Dr. Elaine D. Collins, for taking the time to review my thesis. I especially would like to thank Dr. Cheruzel for putting up with me and giving me great advice. I would like to thank Savitha Srinivasan, a former graduate student at Dr. Lustig's lab for her help during my first time in the lab. I would also like to thank the undergraduate students, Reema Shalan, Lilley Tran, and Ashleen Kaur for assisting me throughout the project. For my family, I would like to thank my parents for their financial support. I am also extremely grateful to Hong Nguyen, my fiancée, who has been absolutely patient, encouraging, and my pillar of mental support throughout my academic years. With her unconditional love and unshakable support over the years, we manage to pull through the hard times and stay strong together. Last but not least, this thesis is dedicated to my loving and caring grandfather in heaven. To others who have been supporting me, I thank you very much.

Loc Tien Nguyen

Table of Contents

List of Figures	viii
List of Tables	xiii
Chapter 1 Introduction	1
1.1. Importance of RNA and Proteins	1
1.2. RNA-Protein Interactions	2
1.3. TAR RNA and Tat Peptide: Structure and Interaction	3
1.4. Macromolecular Structure Modeling and Predictions	5
1.5. Present Research	7
Chapter 2 Background	8
2.1. Bioinformatics and Cheminformatics	8
2.2. Programming Languages for Software Development	9
2.3. Coarse-grained and Lattice Modeling: Definition and Development	9
2.4. Flexibility of Macromolecules	11
Chapter 3 Methods	13
3.1. Resurrection, Modernizing, and Modulation of Former Coding	13
3.2. Setting Up	13
3.3. Algorithm	21
3.4. Running the Program	28
3.5. Energy Calculation	29
3.6. Mutating Peptide Sequence	34
3.7. Other Software Tools	34
3.8. Future Modifications	35
Chapter 4 Results	38

4.1.	Lattice Modeling Results	38
4.1.1.	Contact Frequency Map	38
4.1.2.	Configurations for ‘2-residues per move’ Approach	39
4.1.3.	Configurations for ‘1-residue per move’ Approach	43
4.1.4.	G11-C25 Distance Distribution	45
4.2.	Special Configurations	46
4.3.	Energy Distribution of All Configurations	49
4.4.	Energy Comparison for “2-residues per move” Special Configuration Sets	52
4.4.1.	<i>K75G</i> Tat Mutant	52
4.4.2.	<i>R78G</i> Tat Mutant	54
4.4.3.	<i>K75G-R78G</i> Tat Mutant	55
4.4.4.	Flexibility Effects	58
	Chapter 5 Discussion	61
	Chapter 6 Conclusion	65
	Chapter 7 Future Studies	66
	References	67
	Appendices	70
	Appendix A Additional Tables and Figures	70
	Appendix B FORTRAN Program Listings	80
	Appendix C Python Program Listings	123

List of Figures

- Figure 3.1. BIV TAR RNA and Tat peptide. A) Secondary structure of RNA BIV TAR fragment. B) Wild-type sequence of Tat binding peptide. Bold sequence (Arg70 to Arg80) shows binding region used in lattice modeling. 14
- Figure 3.2. Lattice space for Stem 1. There are 7 (x,y) planes along the y-axis to accommodate the 6 base pairs of stem 1 and stem 2. (A) 119 lattice points of stem 1 in 3D system: red = x, yellow = y, green = z. (B) 42 lattice points lattice space of stem 1 on xy-plane. 16
- Figure 3.3. Division of BIV TAR RNA and Tat peptide into smaller sections. (A) There are three sections on the RNA fragment: 2 fixed stem regions and 1 moveable bulge region; the bulge region is further divided into two bulges (UGU for bulge 1 and C25 for bulge 2) in the program. (B) The bold region (11 residues) was used in the program to calculate possible moves and folding of the Tat peptide. 17
- Figure 3.4. The lattice moves used in the program with 18 total possible moves. (Red) point n centered at the junction of 8 cubes. (Green) point n+1 achieved after using 1 of the 18 possible moves from point n. (A) 6 moves along the edges. (B) 12 moves along the diagonals. 18
- Figure 3.5. All 18 possible moves of to build one amino acid residue from the junction of eight cubes. 19
- Figure 3.6. Setup of boundaries for the Bulge region. Numbers represent indices on the stems: 6 and 7 for stem 1, and 1 and 12 for stem 2. This boundary limits the moves for the bulges, so that any bumping of lattice points between the stems and bulges can be eliminated. 20
- Figure 3.7. Generalized lattice conceptualization for TAR-Tat binding complex. (A) Stem space; (B) Bulge space; (C) Tat peptide space. The stem space creates a spatial boundary to prevent the bulges and peptide from reaching points too far away from the whole system. 21
- Figure 3.8. Existing Interaction Potentials (Lustig et al., 1997) of BIV TAR-Tat complex. The ATOMS 1st row represents the RNA bases: capitalized letters for major groove and small letters for non-major groove. The ATOMS 1st column shows the 20 alphabetically organized one-letter coded amino acids with corresponding

numerical order set up by the program. The columns range from 1 to 8 to accommodate the two types of grooves. When the program calculate the energy of a configuration, it will look up a pair(i, j) with i from 1 to 8 for an RNA base and j from 1 to 20 for an amino acid. The junction cell then gives the interaction potential of that specific base-amino acid pair. 30

Figure 3.9. General operation schematics. (A) Running the main program. (B) Running extra tools: (1) Examine flexibility based on ‘edge’ and ‘diagonal’ moves for special configurations; (2) Generate .pdb files. Solid arrows represent the order of operation. Dash arrows show that the processes can be connected within the code. In the ‘Input’ section, peptide sequence can be modified to calculate energies for different substitutions in the sequence. All files are modifiable accordingly to different scenarios. 37

Figure 4.1. Contact map between Tat amino acid positions and RNA bases. RNA positions are shown in both lattice positions as used in the main program and their actual sequence locations. Bold/italic RNA nucleotides are those in the major groove. 39

Figure 4.2. TAR RNA (1MNB) backbone and its Tat peptide aligned with lattice RNA and its best-fit native-like peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA. 40

Figure 4.3. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its offset peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA. 41

Figure 4.4. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its nonmajor-groove peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA. 42

- Figure 4.5. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its minor-groove peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA. 43
- Figure 4.6. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its best fit native-like peptide (1-residue per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA. 44
- Figure 4.7. Distance distribution of G11-C25 base pair for all 12,047,142 configurations (2-residues per move). 46
- Figure 4.8. Special configurations in 2D and 3D. (A1 and A2) All Edge. (B1 and B2) Diagonal after 74th position, Diag @74. (C1 and C2) Diagonals at 74th and 78th positions, Diag@74_78. (D1 and D2) Diagonals after 74th and 76th positions, Diag @74_76. (E1 and E2) All Diagonal. (F1 and F2) Diagonal after 78th position, Diag @78. (Red) Edge moves. (Green) Diagonal moves. Note that the 6 segments start from G14 and end at Arg80, arrows show peptide direction, and all odd-numbered CA are midpoints of the segments. 47
- Figure 4.9. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its best fit native-like all-edge peptide (2-residue per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA. 48
- Figure 4.10. Energy distribution for all 12,047,142 configurations in Native and Mutant peptides (Native, K75G, R78G, and K75G-R78G). X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals (see Appendix A for breakdown of configurations in major and non-major groove space). 50

Figure 4.11. Energy distributions of Native and K75G for same-configuration species. There are three sets of special configurations: All Edge, Diagonal at position 78, two diagonals at positions 74 and 76, and two diagonals at positions 74 and 78.	53
Figure 4.12. Energy distributions of Native and R78G for same-configuration species. There are three sets of special configurations: All Edge, Diagonal at position 78, two diagonals at positions 74 and 76, and two diagonals at positions 74 and 78.	55
Figure 4.13. Energy distributions of Native and K75G-R78G for same-configuration species. There are three sets of special configurations: All Edge, Diagonal at position 78, two diagonals at positions 74 and 76, and two diagonals at positions 74 and 78.	57
Figure A.1. Energy distribution for all configurations in Native and Mutant peptides (Native, K75G, R78G, and K75G-R78G) located in the Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.	74
Figure A.2. Energy distribution for all configurations in Native and Mutant peptides (Native, K75G, R78G, and K75G-R78G) in the Non-Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.	75
Figure A.3. Energy distribution for All-Edge configurations of Native and Mutant peptides (Native, K75G, R78G, and K75G-R78G) in the Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.	76
Figure A.4. Energy distribution for configurations with diagonals after the 74 th and 76 th positions of Native and Mutant peptides (Native, K75G, R78G, and K75G-R78G) in Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.	77
Figure A.5. Energy distribution for configurations with diagonals after the 74 th and 78 th positions of Native and Mutant peptides (Native, K75G,	

R78G, and K75G-R78G) in Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

78

Figure A.6. Energy distribution for All-Diagonal configurations positions of Native and Mutant peptides (Native, K75G, R78G, and K75G-R78G) in Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

79

List of Tables

Table 3.1. Conversion table between BIV TAR RNA fragment sequence and indexed regions in the output array rnapts. Only the stem regions have fixed coordinates. Bulge regions can vary based on the program's calculations of possible moves for each residue of the two bugles.	16
Table 4.1. Filtering results for the special configurations.	49
Table 4.2. Binding energy (kcal/mol) for '2-residues per move' and '1-residue per move' lattice structures. Energy values were calculated from the best-fit structures (see Section 4.1.2 and 4.1.3) in both native and mutant Tat peptide scenarios.	51
Table 4.3. Binding energy difference (kcal/mol) between Native (Nat) and mutant (K75G) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row)	53
Table 4.4. Binding energy difference (kcal/mol) between Native (Nat) and mutant (R78G) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).	54
Table 4.5. Binding energy difference (kcal/mol) between Native (Nat) and mutant (K75G-R78G) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).	56
Table 4.6. Binding energy difference (kcal/mol) between different mutant (K75G) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).	58
Table 4.7. Binding energy difference (kcal/mol) between different mutant (R78G) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).	59
Table 4.8. Binding energy difference (kcal/mol) between different mutant (K75G-R78G) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).	59

Table A.1. Energy values (kcal/mol) from energy distributions (histograms) for each type of configurations and each Tat species.	70
Table A.2. Energy values (kcal/mol) from energy output files after running the main program and/or energy calculation program for each type of configuration and each Tat species.	71
Table A.3. Binding energy difference (kcal/mol) between Native (Nat) and mutant (K75G) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).	72
Table A.4. Binding energy difference (kcal/mol) between different mutant (K75G) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).	72
Table A.5. Binding energy difference (kcal/mol) between Native (Nat) and mutant (R78G) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).	72
Table A.6. Binding energy difference (kcal/mol) between different mutant (R78G) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).	73
Table A.7. Binding energy difference (kcal/mol) between Native (Nat) and mutant (K75G-R78G) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).	73
Table A.8. Binding energy difference (kcal/mol) between different mutant (K75G-R78G) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).	73

Chapter 1

Introduction

1.1. Importance of RNA and Proteins

Ribonucleic acid, RNA, and proteins are two of the three major biological macromolecules that are essential to all life forms on Earth. Together with DNA, they form the central dogma (Crick, 1970):

However, thanks to scientific breakthroughs in molecular design, synthesis, and spectroscopic techniques, researchers have made new discoveries updating the central dogma. In this current and more extensive version, the dogma now includes nucleotide-protein interactions in both transcription and translation.

Unlike DNA, whose function involves semi-conservative self-replication and storing genetic information, RNA can have more diverse cellular functions, because it can form subunits of the ribosome also known as rRNA, and spliceosomes from small nuclear RNAs (snRNA) together with a collection of protein factors, and in addition shows some catalytic activities as ribozymes (Jones et al., 2001; Will and Luhrmann, 2011). The structure of RNA, even though mostly being single-stranded and containing ACGU, can very likely be found in short duplex helical shapes similar to those of DNA. By achieving helical shapes, they can fold into compact shapes to

facilitate the transfer of genetic information and perform many cellular activities (Fohrer et al., 2006). RNA has a dominant secondary structure of the stable and favorable A-form helix. Interestingly in regulation, RNA can be found as microRNA (miRNA), small interfering RNA (siRNA), and so on, many of which can down-regulate gene expression by acting as complimentary or anti-sense sequences to mRNAs (Erdmann et al., 2001). Overall, RNA has diverse forms and cellular functions that are highly essential to regulating life.

Proteins, equally diverse as RNA, form from basic amino acid residues with peptide bond formation. A single chain is described by a primary structure of only amino acid sequence and secondary and tertiary structures (Alberts et al., 2002). Secondary structures can be classified as helix, sheets, and turns. The tertiary assemblies result from three-dimensional (3D) folding assembly of secondary structures and also includes quaternary structures involving the assembly of multiple chains. Besides, because proteins are really large macromolecules, they typically involve hundreds of amino acid residues. Peptides, on the other hand, are amino acid chains with much smaller primary sequences usually fewer than 50 residues in length. Overall, thanks to such variation in structures, proteins possess a wide range of cellular functions such as structural support, biochemical reactions, molecule transportation, and signaling pathways.

1.2. RNA-Protein Interactions

Even though proteins are synthesized from genetic messages encoded in mRNA, they also have to interact with rRNA and tRNA. For example, many specific

tRNAs can bind aminoacyl-tRNA synthetase during the translation of codons. This and other examples indicate RNA-protein interactions are extensive and diverse, but have not been as thoroughly studied as protein-DNA complexes (Masliah et al, 2013). There is currently focus on the RNA-DNA reverse-transcription and viral transcription processes.

Over decades, according to Falanes-Belasio et al. (2010), it has been shown that there are some specific proteins and peptide sequences that can interact with viral RNA to enhance the transcription of viral gene products. This involves trans-activating transcriptional activator (Tat) proteins that play a key role in viral life cycles and as a potential target for gene therapies and pharmaceutical agents. Tat protein, which was first characterized in 1988, was shown to bind the HIV trans-activation response (TAR) RNA element responsible for trans-activation of viral promoter and for virus replication (Greenbaum, 1996). The TAR-Tat binding complex is the main focus of this thesis.

1.3. TAR RNA and Tat Peptide: Structure and Interaction

HIV and the equivalent bovine immunodeficiency virus (BIV) belong to the *Lentivirus* genus, which collectively requires long incubation periods of time. HIV-1, the most commonly known HIV strain, and BIV possess homologous TAR RNA structures (Greenbaum, 1996) that include:

- Two helical stems.
- The stems flank the two bulges: one 3-nucleotide (nt) bulge on 5' side and one 1-nt bulge on 3' side.

- One hairpin loop structure on stem 2: active in HIV-1 but not binding to Tat in BIV.

These viruses depend on the interaction between virally encoded regulatory proteins and particular viral RNA target sequences, and their critical life cycles are also modulated by other RNA-protein interactions. It has been reported that during the long proviral latency periods most RNA polymerases can disengage prematurely from transcription complexes, thus inhibiting transcriptional elongation and production of viral proteins (Greenbaum, 1996). In order to avoid the inhibition, the complexes rely on the binding of Tat protein to TAR region, a long terminal repeat region at the 5' end of the viral mRNA prematurely terminated transcripts. HIV-1 Tat recruits Positive Transcription Elongation Factor b (P-TEFb) to continue the transcriptional elongation and produce more viral proteins (Romani et al., 2010). BIV Tat, on the other hand, recruits cyclin T1 protein to carry out the same function as the HIV-1 transcription complex (Bogerd et al., 2000).

Tat proteins are gene products encoded by HIV-1 or BIV viral mRNAs to aid and enhance viral transcription. Tat peptides include the binding regions of Tat proteins and are also called cell-penetrating peptides (CPP), because the active domain usually has fewer than 40 amino acid residues and can efficiently enter surrounding cells from the extracellular matrices via some proposed possible methods, e.g. direct penetration and endocytosis mediated translocation (Futaki et al., 2001). Tat peptides belong to the nonamphipathic peptide category of CPP classification as a result of their high content of cationic amino acids (Madani et al.,

2011). This special structural feature gives them the ability to bind to phosphate groups on cell membrane lipid heads and nucleotides, and guanine bases. Tat proteins, in general, do not have specific secondary structure until they bind to nucleotides. In the TAR RNA-Tat protein complexes, the proteins achieve beta-sheet structure and can go into the major groove of TAR RNA (Puglisi et al., 1995).

Focusing on the Tat peptide model, Puglisi et al. (1995) found several special molecular events that can aid the penetration and binding. First, at the bulge region of TAR RNA, U10, unlike the non-interacting U12 in the minor groove, is well positioned in the major groove to interact with Arg70. Second, the pair G12-C25 between the bulges can cause minor distortions in the upper and lower helical stems, thus widening the major groove to allow better penetration of Tat peptide. Third, the peptide itself has multiple Gly residues, which allow the beta turns of the most interacting region (Arg70 to Arg80) to better fit in the major groove. Last, as mentioned above, other positively charged Arg residues can greatly enhance binding.

1.4. Macromolecular Structure Modeling and Predictions

Scientists have utilized two main methods to determine molecular structures for decades: X-ray crystallography and Nuclear Magnetic Resonance (NMR) spectroscopy. These two techniques have been laboratory gold standards for determining molecular structures. Macromolecular crystallography uses X-rays being diffracted due to the regularized structure in a crystal, and the resulting angles and intensities help create a three-dimensional representation of the electron density within the crystals for structural interpretation. NMR utilizes a large magnetic field

to perturb intrinsic magnetic properties of atoms, and their corresponding resonance spectrum can be determined. The resulting data can help interpret molecular structures based on functional groups and non-covalent bond interactions (Nguyen, 2012).

Even though X-ray crystallography and NMR still remain two of the most commonly used techniques for modeling molecular structures and interactions, they do not come without challenges. In X-ray crystallography, the very first step is to create and purify crystals of the molecules in question, and this process requires a meticulous and sometimes extended process that is still being improved (Dale et al, 2003). Similarly, NMR requires a significant purification process to avoid ‘noise’ in the interpretation step and is restricted to smaller proteins (Romero et al., 2001). Lastly, both can be slow and costly and may take months or years to complete (Montelione and Anderson, 1999).

A significant portion of proteins and RNA as well as their complexes have not been crystalized and analyzed. Computational methods have been utilized to augment X-ray and NMR analyses (Kitano, 2002). This can allow researchers, usually biologists and chemists with computer science (CS) backgrounds or collaborating with CS experts, to create algorithms that incorporate constraints consistent with homologous protein or partial NMR or X-ray structures. Then, the algorithms can apply many popular CS methods, e.g. Monte Carlo and Lattice Boltzmann to predict protein folding paths, alternative nucleotide secondary structures, and molecular interactions (Rizzuti and Daggett, 2013). Even when the

structures can be directly obtained experimentally from X-ray crystallography and NMR, computational chemists can apply the data for analysis of intermediates or transition states, modeling, and energy optimization (Kmieciak and Kolinski, 2007). However, analyzing very large structures also can require time and extensive hardware, usually clusters of computers, to allow complete calculation, e.g. Folding@home project at Stanford University (Pande, 2015). Overall, techniques come with both advantages and disadvantages. In the case of computational chemistry, research can expand a menu of possibilities for structural models, analyses, dynamics, and intermolecular interactions.

1.5. Present Research

In this thesis, computational algorithms and coding were utilized to model BIV TAR RNA structure and binding Tat peptide, using a form of coarse-grained model building, lattice modeling. The Method Section will include detailed paradigms, algorithms, and program documentation of computational tools. Lastly, the Future Studies Section will give some proposals and suggestions on continuing this research, including enhanced storage, and application of all-atom approaches to lattice models. The research noted here is to investigate the role of local peptide flexibility in RNA-protein binding. For BIV TAR-Tat, the implications of such research are to advance possible medical applications.

Chapter 2

Background

2.1. Bioinformatics and Cheminformatics

Bioinformatics is the study of biological systems utilizing computational tools and methods. The tools involve the use of personal computers, computer clusters in local laboratories, large-scaled supercomputer centers, and, most recently, ‘cloud’ or internet-based computer systems. One key application of this field is to manage and analyze large data sets, and aid researchers in sequencing, simulating, and modeling poorly-characterized macromolecular structures and newly-found ones (Roberts, 2000).

Similarly, computational methods have been integrated into Chemistry and Biochemistry for decades. This flourishing field was named Cheminformatics in the 1990s, and it was first originally developed in the pharmaceutical field to better create, monitor, and analyze new therapeutic molecules and cellular targets (Gillet and Leach, 2007). Although the methods can be shared between these two fields, the main difference is that in Cheminformatics, the goal is to model the interactions and calculate inherent properties, such as energies, interacting potentials, and binding effects with less focus on the cellular physiological functions. Overall, the information obtained in one area can help the other achieve its goals. Therefore, despite the overlapping features of hardware, software, and methods, both fields have become great assets to support biomedical research.

2.2. Programming Languages for Software Development

There are a variety of computational programming languages such as C, C++, Python, FORTRAN, and Java. A diversity of tools has been developed to predict and model RNA-protein interactions, and each comes with its own strengths and flaws (Puton et al., 2011). In this project, FORTRAN was utilized to create a program for exhaustively exploring RNA-peptide interactions and modeling. FORTRAN has been used as a stable platform for large-scale calculations (Wild et al., 1995). In addition, its input and output files require a highly structured format, such as those used in X-ray crystallography and 2D NMR. Even though FORTRAN is one of the oldest currently utilized programming languages, it has remained highly applicable among researchers in computational chemistry and biology.

Hardware advances have required the need to rewrite and modernize programs written in the older FORTRAN versions. Python, on the other hand, is a more recent scripting language and widely used for general purposes and even high-level programming. Many implementations of Python, such as Pymol, Biopy, and PyRosetta take advantage of code simplicity and fast execution of the scripts. A FORTRAN program can be used for large-scale modeling systems, while Python can provide additional ways to further analyze obtained data. The project in this thesis uses both FORTRAN and Python.

2.3. Coarse-grained and Lattice Modeling: Definition and Development

Computer simulations with controllable details and parameters are called granular computing (Yao et al., 2013). There are two levels of granularity: fine with

as many details as possible and *coarse* with as few degrees of freedom as possible. In biochemical systems, all-atom approaches to determine molecular structures and interactions usually require very extensive calculations of inter- and intramolecular interactions. Therefore, it takes much longer time scales to complete any necessary calculation and modeling. In order to decrease computing runtimes, coarse-grained systems approximate local details with larger components having fewer degrees of freedom. Although coarse-grained models usually cannot explore all possible details, they can exhaustively explore all the spaces occupied by key backbone components. For example, proteins and nucleotides can be reduced to only their important backbone atoms, e.g. C-alpha, to simplify their structures for a significant reduction in computing time. These backbone structures can be built on 2D planes or in 3D space. Using molecular dynamics programs, one can add and minimize the energy for all-atoms. This can include significant changes to the packing of side chains (Kolinski, 2004; Kaufmann, 2010, Dill and MacCallum 2012; Kmiecik et al., 2012).

Lattice models, a subset of coarse-grained models, use points embedded in a simple cube or a 2D Cartesian plane, to model the macromolecules. On a lattice, one can implement an algorithm, called self-avoiding walk (SAW), which allows a sequence of moves for one point that does not overlap on the chain; SAW was first introduced by the chemist Paul Flory (Flory, 1953; Madras and Slade, 1996). The first structures of such modeling were classified and had energies calculated based on hydrophobic and hydrophilic groups of side chains (Lau and Dill, 1989). Each

residue is presented as a bead. On the lattice, each bead can have a random move, as long as it follows a SAW algorithm specifically designed for hydrophobic and hydrophilic interactions and set up with appropriate lattice spacing and constraints. Then, the algorithm can be ‘1-residue per move’ or ‘2-residues per move.’ The latter was commonly used to reduce computing time, but still be able to provide sufficient details for structural investigations; structure predictions of small sequences of proteins were highly achievable (Covell and Jernigan, 1990; Dill et al., 1995; Tozzini, 2005). The lattice model can also be very useful for DNA and RNA modeling (Lustig et al., 1994; Maciejczyk et al., 2013; Cragolini et al., 2013).

2.4. Flexibility of Macromolecules

RNA possesses a diversity of shapes, sizes, and functions, suggesting it can be very flexible. Flexibility includes the ability to adopt different structural conformations when bound by proteins. Puglisi et al. (1995) found that multiple local sites on the TAR RNA fragment, especially in the bulge region, could give RNA some local flexibility by distorting the helical stems and widening major groove for better access of Tat protein to that groove. Weeks and Crothers (1993) also found that by increasing the bulge size and resulting entropies of BIV TAR RNA, the major groove could provide better binding for Tat peptide, and the peptide binding strengths correlated with these entropies with the improved conformational access. Lustig et al. (1998) suggested that RNA-protein binding may be enhanced by increasing local peptide flexibility.

Lustig, Collins and coworkers (Hsieh et al., 2002) explored flexibility of BIV TAR-Tat binding models and determined appropriate possible Gly substitutions in Tat peptide, especially at Lys75 and Arg78, to potentially enhance binding. In this study, lattice modeling was utilized to explore possible configurations and evaluate the energy of the TAR-Tat complex. Continuing this modeling has become the main research interest of this thesis. Therefore, in this project, we substitute Gly into the recommended positions. We explore a variety of lattice moves by filtering them out from all the configurations obtained through our FORTRAN program. Lastly, we also evaluate the energy of TAR-Tat binding complexes. All of the results obtained in this research can be useful in testing the role of peptide flexibility in RNA-peptide binding and hopefully provide preliminary data for developing potential antiviral treatment modalities.

Chapter 3

Methods

3.1. Resurrection, Modernizing, and Modulation of Former Coding

The main program was developed using FORTRAN 77 in late '90s (Hsieh, M.; Lustig, B. San Jose State University. Unpublished Work, 2002). However, due to the recent improvement of computer systems, new FORTRAN compilers have been modified to be more compatible with updated hardware. This creates a problem for older codes when run by new compilers. Therefore, the priority and the majority of time spent in this project involved the resurrection of the codes into FORTRAN 95 and fixed any earlier computational bugs to better parse and analyze output data.

3.2. Setting Up

In this thesis, the BIV TAR RNA and Tat peptide sequences were adapted from Puglisi, and other groups (Puglisi et al., 1995; Hsieh et al., 2002). Figure 3.1 shows the RNA and peptide sequences.

Figure 3.1. BIV TAR RNA and Tat peptide. A) Secondary structure of RNA BIV TAR fragment. B) Wild-type sequence of Tat binding peptide. Bold sequence (Arg70 to Arg80) shows binding region used in lattice modeling.

As a general rule for the program (see Appendix B), any arrays, sized 3 X (# of rows), contain 3 columns and represent the x, y, and z components of an atom in space including RNA nucleotides (nt) and peptide residues. An example array `rna_location(3,28)` can have 3 columns for xyz components and 28 rows for 28 nucleotides presented by their O3' atoms. This array also comes accompanied by another array storing the RNA quartet labels to be used in energy calculation.

From the RNA fragment, a 119-point lattice space (Lustig et al., 1994) can accommodate the fragment and its Tat peptide by using whole-integer coordinates. The use of whole-integer numbers help reducing calculating time for XYZ coordinates and rounding errors when real numbers are used. These 119 points are stored as indices for XYZ coordinates of a 3x119 input array `s1` (stem 1) in the

FORTTRAN program. From the RNA fragment, there are 4 subsections that can be built from the program: section 1 and section 2 each contain 12 base residues of stem 1 and stem 2, section 3 the first bulge (U10, G11, U12), and section 4 the second bulge (C25). In order to reduce computational complications and time, stem 1 and stem 2 are fixed within the program. The first stem's coordinates can be used from indices 1 to 6 and 23 to 28 in the input array. The coordinates of stem 2 are recalculated from stem 1's coordinates and stored in an output array s2 (stem 2) to show the helical turn and hairpin loop of stem 2 by the following spatial transformation:

$$X_{\text{Stem 2}} = (-1) * X_{\text{Stem 1}} + 6$$

$$Y_{\text{Stem 2}} = (-1) * Y_{\text{Stem 1}} + 6$$

$$Z_{\text{Stem 2}} = Z_{\text{Stem 1}} + 8$$

The O3' representations of stem 2 were rotated 180 degrees about the helical axis of stem 1 (z-axis) and translated 6 units away from the axis; they were also moved up 8 units along the z-axis. There are 24 points used to create stem 1 and stem 2, and the remaining 95 points act as spatial constraints for the two bulges and the peptide and prepare a helical lattice space for any possible stem modifications in the future. In Figure 3.2A, the lattice space for stem 1 consists of seven parallel planes along y-axis, and in Figure 3.2B, the overlay of all seven planes shows the boundary of stem 1 if points along the edges are connected. Stem 2 will have the same spatial region as stem 1 around the z-axis.

Figure 3.2. Lattice space for Stem 1. There are 7 (x,y) planes along the y-axis to accommodate the 6 base pairs of stem 1 and stem 2. (A) 119 lattice points of stem 1 in 3D system: red = x, blue = y, green = z. (B) 49 lattice points lattice space of stem 1 on xy-plane.

The whole RNA sequence is initialized as a 3x28 output array, where 28 points sequentially indexed from 1 to 28 in the FORTRAN code, with the first index point being G4 and the last one C31 (see Table 3.1).

Table 3.1. Conversion table between BIV TAR RNA fragment sequence and indexed regions in the output array rnapts. Only the stem regions have fixed coordinates. Bulge regions can vary based on the program's calculations of possible moves for each residue of the two bulges.

Indexed Region in output array	BIV TAR RNA region
1 to 6	G4 to G9 on stem 1
7 to 9	U10 to U12 on bulge 1
10 to 21	A13 to U24 on stem 2
22	C25 on bulge 2
23 to 28	C26 to G31 on stem 1

Similarly to rnapts for RNA, a 3x11 output array peppts was also set up to later contain the 11-residue peptide fragment (Arg70 to Arg80). Overall, Figure 3.4 shows all the divided sections.

Figure 3.3. Division of BIV TAR RNA and Tat peptide into smaller sections. (A) There are three sections on the RNA fragment: 2 fixed stem regions and 1 moveable bulge region; the bulge region is further divided into two bulges (UGU for bulge 1 and C25 for bulge 2) in the program. (B) The bold region (11 residues) was used in the program to calculate possible moves and folding of the Tat peptide.

After the two stems are set up, a lattice set of possible moves for each residue of the two bulges and each alpha carbon (CA) of Tat peptide is created from the input file. This lattice is a modified simple cubic lattice, where one residue $n+1$ needs to move in space such that its previous neighboring residue's coordinates (point n) are put at a virtual center at the junction of eight cubes. From that virtual center, the $n+1$ point can choose one of 18 possible moves: 6 along the edges and 12 along the diagonals. Each cube has $2 \times 2 \times 2$ size, and the total length of each edge is 6.15 Å long. These considerations are consistent with proper O3' spacing of an A-form

canonical RNA fragment. Figure 3.4 illustrates the 18 possible moves from the 8-cube system.

Figure 3.4. The lattice moves used in the program with 18 total possible moves. (Red) point n centered at the junction of 8 cubes. (Green) point n+1 achieved after using 1 of the 18 possible moves from point n. (A) 6 moves along the edges. (B) 12 moves along the diagonals.

Note Figure 3.5 shows how one amino acid can determine one of 18 possible moves for the next CA representation of the amino acid to continue the Tat peptide sequence. Under certain circumstances (2-residues per move), a single move is bisected at a midpoint to represent an intervening CA position.

Figure 3.5. All 18 possible moves of to build one amino acid residue from the junction of eight cubes.

After the lattices are set up, the RNA nucleotides and peptide residues are also converted from letter-based to number-based descriptors. For RNA, the bases A, C, G, and U are numbered 1, 2, 3, and 4 according to their alphabetic order. The program similarly creates the array that contains number-based amino acids, in which the numerical order matches the alphabetical order for 1-letter coded amino acids. This can help track peptide residue of interest to a particular RNA nucleotide. The two numbers obtained from this track represent the potential energy cell with 8 columns and 20 rows (columns = RNA nucleotides in exclusively major groove and others, rows = amino acid) lookup map to calculate a configuration's energy. For example, (8,20) shows the potential energy interaction between tyrosine and uracil in the non major groove, and (4,20) the potential energy interaction between tyrosine and uracil in the major groove (see Figure 3.8). These numbers are stored in arrays rnanum and pepnum for RNA and peptide, respectively.

Another input file was also used to mark the bulges. This region contains indices of the two stems: lattice positions 6 and 7 for stem 1, and 1 and 12 for stem 2. It is important to set this up, so that the algorithm can calculate the possible moves that avoid any bumping into the stem (Figure 3.6).

Figure 3.6. Setup of boundaries for the Bulge region. Numbers represent indices on the stems: 6 and 7 for stem 1, and 1 and 12 for stem 2. This boundary limits the moves for the bulges, so that any bumping of lattice points between the stems and bulges can be eliminated.

The last two important input files were used to set up the possible base-pairing of G11-C25 between the bulges and contact locations of the RNA and peptide. The former tells the program that there is one base pair in the RNA fragment with the first base being the second point of bulge 1 (G11 or point 8 in output RNA array) and the second one the first point on bulge 2 (C25 or point 22 in output RNA array). The latter input file gives two contact points: the first one at second lattice position on the second stem (G14 or point 11 in output RNA array) and the second contact at the third index point of bulge 1 array, i.e. U10. The program can implement these conditions to eliminate configurations without any aforementioned base pair and

constrain the lattice space to build the Tat peptide. Figure 3.7 shows all the lattice spaces for TAR stems, the bulges, and Tat peptide.

Figure 3.7. Generalized lattice conceptualization for TAR-Tat binding complex. (A) Stem space; (B) Bulge space; (C) Tat peptide space. The stem space creates a spatial boundary to prevent the bulges and peptide from reaching points too far away from the whole system.

3.3. Algorithm

The program adopted the ‘2-residues per move’ SAW strategy to calculate possible moves for the bulges the peptide. This strategy has been proved to effectively reduce computational time and give efficiency in building macromolecular models. Here is how it works (note added comments and only algorithm and pseudocode are given):

```
{
  Initialize array of points for the necessary structures
  • Array = (n, n+1, n+2, n+3...)
  # Depend on programming languages:
```

- Fortran: Indices start at position 1 → even-numbered indices are midpoints
- Java, C++, Python: indices start at position 0 → odd-numbered indices are midpoints

Apply the move lattice:

- First point is always known or a fixed contact point
 - Every other point calculated as:
 - $\text{Point } n+2 = \text{Point } n + \text{move}$
 - The midpoint calculated as:
 - $\text{Point } n+1 = \text{Point } n + \text{move}/2$
 - Check for bumps:
 - If bump happens: apply another move in the lattice move
 - If no bump: move to the next point in the array
 - Store possible moves in a *record* array for future modeling
- }

This strategy applies effectively for the 11-residue Tat peptide. However, because the bulge regions has a 3-nt bulge (U10-G11-U12) and 1-nt bulge (C25), application of this strategy appears not computationally efficient. There are two reasons for this dilemma:

- For bulge 1: the second point will always be the midpoint and all points are on the straight lines.
- For bulge 2: there is only one point and the variety of configurations will be much less diverse.

Therefore, we came up with the following solutions:

- For 3-nt bulge 1: a 7-point array was created. The first point is G9, the bulge's O3' at points 3, 5, and 7, and the remaining dummy midpoints. A13 on stem 2 limits how far U12 can move.
- For bulge 2: a 3-point array was created. The first point is U24, the bulge's O3' at point 3, and the midpoint between O3' and U24 at point 2. C26 on stem 2 limits how far C25 can move.

This solution can create flexibility for the bulge region setup from input file and close potential holes between bulges' atoms to prevent peptide from intersecting with the virtual bonds between the bulges' atoms.

For the 11-residue Tat peptide, we included the first contact point of the peptide with RNA fragment, which is the second index point on stem 2 (G14), and the peptide's Pro69, which we assumed as the midpoint between G14 and Arg70. However, when energy was calculated, Pro69 was excluded due to having no contact with RNA. Using such criteria, we set up a 13-points array, in which the first point is G14 and all of the odd-numbered positions, corresponding to the odd-numbered amino acids, are midpoints.

Applying this strategy, the program divides the bulge 1 into 3 smaller sections (G9 to U10, U10 to G11, and G11 to U12) to record the possible moves for each section. Similarly, there are two sections for bulge 2: U24 to dummy midpoint and dummy midpoint to C25 to prevent the peptide from crossing U24-C25 connection. For Tat peptide, there are six sections: G14 to Arg70, Arg70 to Thr72, Thr72 to

Gly74, Gly74 to Gly76, Gly76 to Arg78, and Arg78 to Arg80. Since the lattice edge is 6.15 Å long, each section, RNA and peptide, also has the same length (or its integer multiple) and distance between two consecutive peptide points, including midpoints, at 3.075 or 4.349 Å.

After dividing the sequences for bulge 1, bulge 2, and peptide into smaller sections, we applied these classifications into making FORTRAN subroutines to calculate possible moves for the lattice structures. Note all detailed codes can be found in Appendix B.

Subroutine bulge1

```
{
# make bulge 1 array: 3 columns for XYZ coordinates and 7 rows for 7 points
INTEGER, DIMENSION(3,7) :: b1

# make record array to keep track of the moves
INTEGER, DIMENSION(3) :: histb1 # record the three sections of b1 array

# import the lattice move matrix into the program
INTEGER, DIMENSION(3,18) :: latmov = import(lattice move input file)

# assign G9 to the first index point
b1(i,1) = s1(i,6) # i is from 1 to 3 for x, y, and z

ptb(i) = s2(i,1) # similar to above. This sets up the limit point for bulge 1

rstp = 1 # represent sections of b1: rstp is from 1 to 3

move = 1 # default first move and from 1 to 18

Loop:
```

```

{
    b1(i,2*rstp+1) = b1(i,2*rstp-1) + latmov(i,move) # set up every other point
    b1(i,2*rstp) = b1(i,2*rstp-1) + latmov(i,move)/2 # set up the midpoints
    ### Code: calculate absolute values for x, y, and z distances from
    b1(i,2*rstp+1) point to the limit point. This is to check how far/close the O3'
    atom is from stem 2. The maximum distance (max) is calculated, and so is the
    difference between the sum of all distances and max. A variable reqmov,
    number of required moves, can be assigned by either max or the difference
    value, whichever greater. ###

    remain = 2*(3-rstp+1) # calculate the number of points remaining
    # Check:
    • if remain less than reqmov → increase to a bigger move because there can
      be more possible moves for the same points (see General Rules).
    • if remain greater or equal reqmov → no more possible move and the
      previously used move number is recorded into histb1; rstp is incremented
      to move the next section of b1 and restart the beginning of the loop.
    • Call subroutine check → check for any bump between the points of b1 and
      their selves, stem 1, and stem 2. If bump appears, go to check move (see
      General Rules).

    ##### After 'move' is used up and recorded and if 'rstp' is not 3, 'rstp' is
    incremented by 1 unit and 'move' reset to 1 → return to beginning of loop:

    histb1(rstp) = move

```


If 'rstp' is 3 → last section of b1 is reached:

call subroutine bulge2 # calculate bugle 2

The variable 'move,' if not equal to 18 (last possible move), is increased by 1 and the building process restarts with 'rstp' = 3 and new 'move.'

If 'move' becomes 18, 'rstp' is decreased by 1 unit, 'move' is assigned a new value corresponding to the one associated with new 'rstp' value recorded in histb1.

General Rules #####:

- a. Always check for 'move,' if it is not 18, then increase. When 'move' reaches 18, decrease 'rstp' (currently greater than 1) and use a new 'move' associated with the new 'rstp' value. Restart loop.
- b. When 'rstp' is 3, call subroutine bulge2; if 'rstp' is less than 3, check 'move' value. Comb back to rule (a).

} #### End loop

} ##### End subroutine bulge1

We applied the same algorithm for bulge 2 but with some modifications:

Subroutine bulge2

{

INTEGER, DIMENSION(3,3) :: b2 # make b2 array containing 3 points

INTEGER, DIMENSION(1) :: histb2 # there is only 1 section of bulge 2

Apply same algorithm and rules as those of subroutine bulge1.

```

##### Replace call subroutine bulge2 with call subroutine tatmov → calculate
peptide

##### rstp now only has 1 value of 1

remain = 2*(1+rstp-1) = 2

##### Check for bumps between the points of bulge 2 and their selves, bulge 1,
stem 1, and stem 2.

##### Check for base-pairing between G11 (5th position of b1) and C25 (3rd
position of b2).

} ##### End subroutine bulge2

```

Moving to the calculation of the moves for Tat peptide, we applied the same algorithm:

Subroutine tatmov

```

{
INTEGER, DIMENSION(3,13) :: t1 # make array t1 to store G14, P69, and
                                the 11 residues of the sequence used in
                                this program.

INTEGER, DIMENSION(6) :: histt1 # make array histt1 to store the moves
                                for 6 sections of t1 array.

##### Apply same algorithm and general rules as those of subroutine bulge1.

##### Replace call subroutine bulge2 with call savecfg. The called subroutine
depends on whether the new re-written/modified program or the original one.

These functions store calculated moves for the bulges and peptides, and also
calculate potential energies for each configuration.

```

```

##### Variable tstp represents the sections of t1, and it ranges from 1 to 6.

remain = 2*(6-tstp+1)

##### Check for bumps between the points of peptide and their selves in t1, bulges
1 and 2, and stems 1 and 2.

} ##### End subroutine tatmov

```

There are other subroutines to convert the recorded moves into actual coordinates for RNA and peptide atoms and calculate potential energies. These will be shown in the codes for the program in Appendix B.

3.4. Running the Program

Before being run on any computers or after modifying the code, the program has to be compiled to ensure system compatibility and check for any post-modification errors. Here is how to compile and run the program:

- Compile: `gfortran -O3 -o filename script.f95`
- Run: `./filename`

The compiler uses `-O3` optimization flag to ensure the highest level of optimization and to check for as many syntax and other code errors as possible and reduce file size (GNU Compiler Collection). The program was checked to run on the three most popular operating systems:

- Mac OSX Yosemite 10.3.3: gfortran 5.0 was installed. Compiling and running programs were done in Mac's native Terminal application.

- Windows 7 and 8.1: Cygwin was set up to install gfortran to create a more programming-friendly environment on Windows machines. It also gives a better Terminal (Command) application to run the codes.
- Ubuntu 12.04: FORTRAN can either be natively run or installed from extra depositories. Compiling and running programs were done in the native Terminal application.

In those aforementioned operating systems, there were no compiling and running errors observed. Besides those systems, the program was also uploaded onto Lustig Group/College of Science proteins1 server for storage and compilation tests. The server runs a scientific version of Linux (Carbon 6.0), and no errors were observed in the Terminal application.

After running the program to output a record file of configurations, another program, makepdb.f95, was written to convert the moves from a record file into actual coordinates, and output ‘.pdb’ files which can be viewed in Pymol, Jmol, and Swissview. Note that all program names in this chapter are temporary and final versions of these programs can have users input their preferred output filename. Appendix B lists all programs in more details.

3.5. Energy Calculation

Within the TAR-Tat complex, one amino acid can have multiple contacts with RNA bases. This condition requires an averaging method to account for important interactions. Therefore, the program weights an ensemble of contacts by distances, which are 2, 3, and 4 lattice units, because a base-amino acid interaction is defined

within 12.30 to 24.60 Å (1 lattice unit = 6.15 Å). The program calculates the potential energy for each lattice configuration as follows:

- All recorded moves for the bulges and peptide were utilized to create arrays of coordinates for these structures.
- All RNA base-amino acid interaction potentials were adopted from logarithmic values of the relative frequencies or relative energies (Lustig et al., 1997). Figure 3.8 summarizes these interaction potentials as a map.

Pair = (Column, Row)		Columns	1	2	3	4	5	6	7	8
Rows	ATOMS	A	C	G	U	a	c	g	u	
1	A	0	0	0	0	0	0.566	1.51	1.017	
2	C	0	0	0	0	0	0	0	0	
3	D	0	0	0	0	0	1.003	0	0.037	
4	E	0	0	0	0	0	0	1.576	0.282	
5	F	0	0	0	0	0	0	0	0	
6	G	0	0	0	0	0	0	0	0	
7	H	0	0	0	0	0	0	0	0	
8	I	0	0	0	0	0	0	0	0	
9	K	0	0	0	0	0	1.038	0	0.632	
10	L	0	0	0	0	0	0	0	0	
11	M	0	0	0	0	0	0	0	0	
12	N	0.01	0	0	0.2	0	0	0.481	0	
13	P	0	0	0	0	0	0	0	0	
14	Q	0	0	0	0	0	0.536	0	0.836	
15	R	0.484	0	1.36	0	0	0	0.237	1.419	
16	S	0	0	0	0	1.383	0	0	0.753	
17	T	0	0	0	0	1.383	0	0	0.753	
18	V	0	0	0	0	0	0	0	0	
19	W	0	0	0	0	0	0	0	0	
20	Y	0	0	0	0	0	1.531	0	0	

Figure 3.8. Existing Interaction Potentials (Lustig et al., 1997) of BIV TAR-Tat complex. The ATOMS 1st row represents the RNA bases: capitalized letters for major groove and small letters for non-major groove. The ATOMS 1st column shows the 20 alphabetically organized one-letter coded amino acids with corresponding numerical order set up by the program. The columns range from 1 to 8 to accommodate the two types of grooves. When the program calculate the energy of a configuration, it will look up a pair(i, j) with i from 1 to 8 for an RNA base and j from 1 to 20 for an amino acid. The junction cell then gives the interaction potential of that specific base-amino acid pair.

- A contact between a base and an amino acid was defined as a distance within a range of 12.30 and 24.60Å, or 2 and 4 lattice units (1 lattice unit = 6.15Å). The program needs to check if the contact is exclusively in the major groove or other grooves before calculating the energy.
- Algorithm:


```

{
  # Variables:

  REAL prsum # Energy

  INTEGER, DIMENSION(28) :: rnanum # array stores the RNA
  sequence in number-based system (see Section 3.2)

  INTEGER, DIMENSION(11) :: pepnum # similar to rnanum, but
  for Tat peptide (see Section 3.2)

  INTEGER, DIMENSION(3,11) :: peppts # array stores whole-
  integer coordinates of the 11 residue of Tat peptide (see Section 3.2)

  INTEGER, DIMENSION(3,28) :: rnapts # array stores whole-
  integer coordinates of the BIV TAR RNA 28 bases (see Section 3.2)

  loop j = 1 to pepcnt # pepcnt = number of amino acids = 11

  loop i = 1 to rnaCnt # rnaCnt = number of bases = 28

  # Within loop(j) that goes through all amino acids, loop(i) makes sure
  to check all possible contacts between each amino acid and each base.

```

The program checks the XYZ coordinates of each amino acid to determine if it belongs to a minor groove. Non-exclusively-major grooves are defined by coarse planes $P = (x,y,z)$ as follows:

- $y < 2$ and $z < 4$
- $x > 4$ and $4 \leq z \leq 10$
- $y > 4$ and $z > 10$

These numbers are defined as lattice units (1 lattice unit = 6.15Å).

For simplifying variable name, **minor** is used for non-exclusively-major grooves.

minor = 1 if one of the followings is TRUE:

- $\text{peppts}(3,j) < 4$ AND $\text{peppts}(2,j) < 2$
- $4 \leq \text{peppts}(3,j) \leq 10$ AND $\text{peppts}(1,j) > 4$
- $\text{peppts}(3,j) > 10$ AND $\text{peppts}(2,j) > 4$

minor = 0 if one of the above is FALSE

Calculate Distances between the RNA base and amino acid– No actual distance calculation formula for 2 points in space was applied, since the program utilized the lattice distances to avoid rounding errors for non-whole-integers. The recorded distances are the absolute values of the calculated ones.

r(1) = abs(rnaps(1,i) – peppts(1,j)) # distance with respect to x

r(2) = abs(rnaps(2,i) – peppts(2,j)) # distance (w.r.t) y

r(3) = abs(rnaps(3,i) – peppts(3,j)) # distance (w.r.t) z

dist = r(1)+r(2)+r(3) # Sum all distances – essentially, this is a vector sum of all lattice distances.

Apply the distance constraints to ‘dist’ and weight the values with an alpha number. Smallest distances are weighed the largest, since they show strong interactions, and vice versa for largest distances:

if 2 <= dist <= 4 then:

if dist = 4, alpha = 0.25

if dist = 3, alpha = 0.50

if dist = 2, alpha = 1.00

Calculate Energy:

prsum = prsum+potent(rnanum(i)+4*minor , pepnum(j))*alpha

rnanum(i) gives the RNA base in numerical value, and pepnum(j) gives numerical values for the amino acids (see Section 3.2). Even though the contacts are confined to the major groove, the ‘minor’ values will help get the correct base in case the contact falls in the minor groove from the corresponding column in Figure 3.8.

Therefore, if ‘minor’ = 0, the contact is already in the major groove (columns 1-4 of Table 2). However, if ‘minor’ is 1, the contact has to be obtained from columns 5-8 of Figure 3.8. The potential energy is multiplied by alpha because the distances are weighed. The energy of ONE configuration equals the SUM of ALL interaction contacts

between RNA bases and amino acids ($11 \times 28 = 308$ in total before being weighed).

$$\text{Energy of a configuration} = \sum \text{Potential}(\text{base, amino acid}) \times \alpha$$

} ##### End Energy Calculation

3.6. Mutating Peptide Sequence

Since the program uses an input file `bivtat.pep` to read the peptide sequence, any necessary mutations can be modified using that input file. Because the research project focused on the flexibility calculation of Tat peptide when mutated with the substitution of Lys75 and Arg78 with Gly, the user can open the input file and replace the single letter codes for Lys (K) and Arg (R) of interest with “G”. In section 3.2, all amino acids were converted from one-letter coded sequence to number-coded sequence, so the program can calculate the correct potential energies between the RNA and peptide.

In this research project, there are three scenarios of Gly substitutions: single Gly substitution at either Lys75 or Arg78 and double Gly substitutions at both 75th and 78th positions. There are four different input files for the native and three mutant sequences that users can input when running the main program.

3.7. Other Software Tools

Besides the main program, there are other programs, written in FORTRAN 95 and Python 2.7.3 (see Appendix B and C) to aid the research project. The following list shows some of these programs:

- `splitpdb.py`: reads output `.pdb` files from the main program, creates correct format for `pdb` viewer programs, and split those `.pdb` files into 2 sets: 1 for BIV TAR RNA and the other for Tat peptide. This separation further helps viewing each lattice structure separately and preparing them for other analyses.
- `pdb_2points_distance.py`: prompts user to input 2 points and the number of configurations, so the program can calculate the distance between the 2 points within these configurations. This program was written to check the distance between point 8 and 22 or G11 and C25, the only base pair between the two bugles. It can also be used for any two points.
- `filter_flex.py`: filters some specific configurations from the record file containing all configurations. Filter conditions can be input by user's choice when the program is run.
- `energycalc_unit2_universal.ff95`: calculates energies for the configurations filtered by the `filter_flex.py` program.
- `histogram.f`: creates histograms for obtained energies, original or filtered. It needs the input file *hinput.cmd* to read the record file for energies, outputs another file with organized data, and sets up the maximum and minimum energies as well as the spread of data points.

3.8. Future Modifications

In the future, the '2-residues per move' can be modified to '1-residue per move;' this means that instead of using midpoints, the program can calculate the

moves for each individual point (RNA base or amino acid) to closely resemble the natural folding pathway of peptides. The most current version of the '1-residue per move' code has been tested with moderate success, since the algorithm is not entirely optimized to reduce operation time for output (see Figure 3.9 for operation flowchart). As an estimate, the program may take a significant amount of hard drive space and probably long timespans to finish. The completion time depends on algorithm optimization, read-write rate of hard drive, and how fast the CPU can calculate moves and energy. In other words, this involves both software and hardware issues.

Figure 3.9. General operation schematics. (A) Running the main program. (B) Running extra tools: (1) Examine flexibility based on ‘edge’ and ‘diagonal’ moves for special configurations; (2) Generate .pdb files. Solid arrows represent the order of operation. Dash arrows show that the processes can be connected within the code. In the ‘Input’ section, peptide sequence can be modified to calculate energies for different substitutions in the sequence. All files are modifiable accordingly to different scenarios.

Chapter 4

Results

4.1. Lattice Modeling Results

4.1.1. Contact Frequency Map

One output from the main program is (see Figure 4.1) the contact map, i.e. the frequency (involving all defined configurations) for one amino acid interacting with a nucleotide. Here, the peptide sequence is only represented with position numbers.

The reasoning behind this is the absence of side chain optimization, where we assumed the same lattice contact frequencies for all types of Tat species. This contact map also confirmed that our newly rewritten code generates the identical output as the former program, thus validating our approach (Hsieh, M.; Lustig, B. San Jose State University. Unpublished Data. 2002). An investigation of the contact frequency map shows there are more nucleotide-amino acid contacts at position 78 than 75. Even though early work by Puglisi et al. (1995) described no direct contact for Arg78 with the RNA, a study on RMSD values for an ensemble of NMR structures did show some potential contacts at 78, even though much less diverse than 75 (Hsieh et al, 2002).

RNA		Positions										
Lattice Positions	Real Positions	Amino					Acids		Position			
		70	71	72	73	74	75	76	77	78	79	80
G1	4	0	0	0	0	0	936	13020	4648	34190	7951	24943
G2	5	0	0	0	0	18475	0	27475	127	43008	467	102132
C3	6	0	0	408391	82181	185953	72596	188699	106528	304167	210989	577279
U4	7	0	0	0	29013	352326	124046	369133	161319	472660	275167	651176
C5	8	0	401976	1334134	893002	1218668	804733	1350275	1018560	1921696	1998176	3605176
G6	9	3958901	1044531	1044531	510380	922879	542345	1103245	803151	1848056	1966210	4430234
U7	10	4841827	2473002	3037246	1970765	2654142	2058364	3067330	3039250	5132284	6971643	12047142
G8	11	2638291	1827796	2482779	1803897	2383589	1823324	2631255	2346509	3619567	4010599	6170110
U9	12	4152073	2367308	2776624	1914827	2446411	1665799	2457456	1934355	3035733	2979518	4565580
A10	13	9126149	9126149	4436176	2205943	3322798	2083777	2767675	1952109	3000776	3044111	4382679
G11	14	12047142	12047142	5079686	1190865	2084554	483159	1262924	382059	1238637	545624	1823079
C12	15	12047142	1983082	1983082	766214	645618	227228	360236	135527	248111	76011	186565
U13	16	2920993	3150566	2560328	757585	1023235	304890	471948	129429	245458	21359	2315
C14	17	0	0	32279	2206	9176	405	1136	0	0	0	0
A15	18	0	0	0	0	49513	6561	21017	0	0	0	0
U16	19	3958901	2624767	2624767	1357624	1092147	519472	582044	321776	444405	240865	405069
U17	20	0	0	713261	253209	576583	277380	415203	163890	285122	90826	125806
A18	21	0	0	479843	260530	476821	265273	358865	180647	298599	161500	218656
G19	22	12047142	1799697	1799697	772161	731291	246621	409939	158001	417239	225660	679461
C20	23	2920993	1589229	2614477	832730	1095543	463260	790385	351596	758851	270333	619769
U21	24	12047142	4635433	4635433	2352549	2329734	1401294	1985898	1424201	2432638	2620208	4256875
C22	25	1638067	737159	1426981	835937	1381053	966316	1503394	1207561	1951472	1681129	2758306
C23	26	0	0	0	0	261175	69817	379247	119838	624389	332703	1239751
G24	27	0	0	549183	329247	921629	676122	1434991	1122755	2208326	2054826	3514063
A25	28	0	0	0	0	57479	20335	172357	78679	397692	209089	863619
G26	29	0	0	0	0	0	0	8459	1109	42075	10810	123993
C27	30	0	0	0	0	0	0	0	92	1292	283	13095
C28	31	0	0	0	0	0	0	0	0	119	0	0

Figure 4.1. Contact map between Tat amino acid positions and RNA bases. RNA positions are shown in both lattice positions as used in the main program and their actual sequence locations. Bold/italic RNA nucleotides are those in the major groove.

4.1.2. Configurations for ‘2-residues per move’ Approach

There were 12,047,142 configurations obtained after running the program with two residues per move. All of them are documented in the record file in the forms of possible moves for the bulges and peptides and the energy associated with each configuration to reduce write-to-disk time and file size.

The following Figures, from 4.2 to 4.5, show the sample overlay of ‘2-residues per move’ lattice structures and NMR (1MNB.pdb) structure. Lattice TAR-Tat complex was imposed on 1MNB TAR RNA by using pair_fit command in Pymol (version 1.6) for O3’ (lattice RNA) and O3’ (1MNB RNA) in Pymol. Figure 4.2

shows the best-fit lattice structure, which has the Tat peptide reside in the major groove.

Figure 4.2. TAR RNA (1MNB) backbone and its Tat peptide aligned with lattice RNA and its best-fit native-like peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA.

Besides the almost-native alignment between 1MNB (NMR) Tat peptide and the lattice peptide as shown in Figure 4.2, the sampling also allows other alternative peptide positions relative to the TAR RNA. The following figures show offset alignment (Figure 4.3), non-major groove alignment (Figure 4.4), and minor-groove Tat peptide (Figure 4.5).

Figure 4.3. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its offset peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA.

Figure 4.4. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its nonmajor-groove peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA.

Figure 4.5. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its minor-groove peptide (2-residues per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA.

4.1.3. Configurations for ‘1-residue per move’ Approach

When the program was tested with ‘1-residue per move’ approach, the program could not finish due to hardware failure after a significantly large number of runtime. However, the recovered record file, which contains the possible moves for the bulges and amino acid residues and energies for different configurations, was found to have around 120,000,000 lines corresponding to the number of configurations. An approximation of upper bound for this case was calculated for a 12-segment line from G14 to Pro69 to Arg80 (13 points in total) with each segment having: $18 - 1 = 17$ possible moves, e.g. 18 in total subtracting one move that returns

a point to its predecessor. Only the very first segment connecting G14 to Pro69 can have 18 possible moves. Therefore, in total, the upper limit for the number of structures in this approach is:

$$(18) * (17) * (17) * \dots * (17) = (18) * (17)^{11} \sim 600,000 \text{ billion}$$

This extremely large upper limit includes all decoy structures, which might not represent the natural state or native Tat peptide folding pathway. Figure 4.6 shows one possible configuration that is in almost-native alignment with the 1MNB (NMR) TAR-Tat complex.

Figure 4.6. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its best fit native-like peptide (1-residue per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA.

Because no exhaustive exploration of configurations in ‘1-residue per move’ method was completed, further analyses of results are based on the configurations and energy values from ‘2-residues per move’ approach.

4.1.4. G11-C25 Distance Distribution

The distance between the base pair G11-U25 (base 8 and base 22 as RNA lattice points) interposed between bulge 1 and bulge 2, respectively, was calculated for all 12,047,142 configurations. The distance between G11 and C25 is from 10.65 to 22.17 Å. The distribution appears bimodal (Figure 4.7), and the maximum frequencies are 3,055,141 and 1,799,710 configurations when distances are equal to 15.06 and 18.45 Å, respectively. The average distance is 17.17 Å, and standard deviation is 3.80 Å. The conditions for base pair check in the program require the distances between these two points within 6 to 10 lattice units and individual x, y, and z distances less than or equal 6 lattice units. These are based on the assumption that G11-C25 makes up a straight line on a lattice plane in a 3x3x3 lattice. This lattice is slightly bigger than the 2x2x2 move lattice, so G11 and C25 can avoid other points in their respective bulge region. The purpose of this distribution is check for any bulge topologies and their associated Tat peptide structures in the future.

Figure 4.7. Distance distribution of G11-C25 base pair for all 12,047,142 configurations (2-residues per move).

4.2. Special Configurations

Of the 12,047,142 configurations from ‘2-residues per move’ approach, the Python program `filter_flex.py` filtered out the following special configurations for the Tat peptide based on the moves for each extending segment of the G14- ... Arg80 structure (see Methods) in the record file. The filtering conditions and how to modify these conditions can be found in the Python script. Figure 4.8 illustrates the special configurations.

Figure 4.8. Special configurations in 2D and 3D. (A1 and A2) All Edge. (B1 and B2) Diagonal after 74th position, Diag @74. (C1 and C2) Diagonals at 74th and 78th positions, Diag@74_78. (D1 and D2) Diagonals after 74th and 76th positions, Diag @74_76. (E1 and E2) All Diagonal . (F1 and F2) Diagonal after 78th position, Diag @78. (Red) Edge moves. (Green) Diagonal moves. Note that the 6 segments start from G14 and end at Arg80, arrows show peptide direction, and all odd-numbered CA are midpoints of the segments.

Using the All Edge set, we were able to filter out one of the best-fit native-like TAR-Tat structures (see Figure 4.9). The conditions for this filter, based on Tat positions were the same as those used to get the complex in Figure 4.2, i.e. the x-coordinates of Arg70 less than 4 and the rest from 0 to 3 lattice units. The energies of this structures are -6.264, -6.264, -5.584, and -5.584 kcal/mol for Native, *R75G*, *R78G*, and *K75G-R78G* species, respectively.

Figure 4.9. TAR RNA backbone and its Tat peptide (1MNB) aligned with lattice RNA and its best fit native-like all-edge peptide (2-residue per move). (Orange and Red) NMR RNA and Tat peptide. (Blue and Green line) Lattice RNA and its peptide. (Yellow) C-Terminal at Arg70. (White) N-Terminal at Arg80. (Magenta) U10 and U12 of 1MNB TAR RNA. (Cyan) U10 and U12 of lattice TAR RNA.

The number of configurations appears to be somewhat proportional to the number of diagonals in the lattice structures. The purpose of these special

configurations is to examine how binding energy is effected by relevant Gly substitutions.

Table 4.1. Filtering results for the special configurations.

Special Configurations	Number of Configurations
All Edge	54,017
All Diagonal	999,514
Diag @74	168,375
Diag @78	144,546
Diag @74_76	219,906
Diag @74_78	243,188

4.3. Energy Distribution of All Configurations

There are four scenarios for energy calculations: Native (no Gly substitution); Gly substitution at Lys75 (*K75G*); Gly substitution at Arg78 (*R78G*); Gly substitutions at both positions 75 and 78 (*K75G-R78G*). The TAR RNA nucleotide-Tat amino acid interactions for each case were calculated (see Section 3.4). Because of an extensive number of possible energy values, i.e. 12,047,142 values, these values were put in a histogram. Here, the maximum energy is set at 0 kcal/mol, the minimum at -20 kcal/mol, and the energy interval at -0.2 kcal/mol (see Figure 4.9). In addition, original non-histogram energy values were also used to calculate an average energy.

Figure 4.10. Energy distribution for all 12,047,142 configurations in Native and Mutant peptides (Native, *K75G*, *R78G*, and *K75G-R78G*). X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals (see Appendix A for breakdown of configurations in major and non-major groove space).

The maximum frequencies for Native, *K75G*, *R78G*, and *K75G-R78G* are 537703, 571302, 687145, and 744409 at -3.8, -3.8, -3.2, and -3.00 kcal/mol, respectively. The mean histogram energy values, in the same order as above, are -4.79, -4.66, -4.12, and -4.02 kcal/mol. The median energy values, in the same order, are -8.30, -7.90, -6.50, and -6.50 kcal/mol (see Appendix A). The median values appear larger than the mean histogram energy because these distributions are left-skewed, and the sufficient way to approach these extremely large sets of data is to use the median values (Lund and Lund, 2015).

The following tables show the TAR-Tat binding energy of the ‘2-residues per move’ and ‘1-residue per move’ lattice structures, which show the best fit for the NMR TAR-Tat complex.

Table 4.2. Binding energy (kcal/mol) for ‘2-residues per move’ and ‘1-residue per move’ lattice structures. Energy values were calculated from the best-fit structures (see Section 4.1.2 and 4.1.3) in both native and mutant Tat peptide scenarios.

Binding Energy (kcal/mol) for Lattice structure: Native and Mutant Tat peptides.	2-residues per move	1-residue per move
Native (No Gly substitution)	-5.924	-3.181
<i>K75G</i> Tat peptide	-5.924	-3.181
<i>R78G</i> Tat peptide	-4.224	-2.501
<i>K75G-R78G</i> Tat peptide	-4.224	-2.501

In both lattice move options (see Table 4.2), it appears that substitutions of Gly into position 78 and/or position 75 can destabilize the TAR-Tat binding complex. It should be noted that the nucleotide-amino acid potentials of Lys is considered comparable to those of Gly in that there is no preference for binding within the RNA quartet (Lustig et al., 1997). Although Arg was shown to have significant preference for RNA binding at guanine, Gly substitution at 78 may negate the possible binding from Arg. In addition, there is no evidence of nucleotide-amino acid contact at 78 as shown by averaged NMR structures (Puglisi et al., 1995; Ye et al., 1995). However, it is worth noting that RMSD analysis of CA shows that there are possible alternate configurations whose 78 positions can have contacts with the TAR RNA (Hsieh et al., 2002). The next sections in this chapter attempt to investigate some possibilities of Gly and related flexibility effects at peptide 75 and 78 positions.

4.4. Energy Comparison for “2-residues per move” Special Configuration Sets

There are two comparison scenarios: each Tat mutant compared against the native peptide, and each Tat mutant compared against itself. In both cases, the final state is configurations in row, and the initial state involves the column (see as an example Table 4.3). Median values are used to calculate the energy different for a large sample as follows:

$$\Delta\Delta G = \Delta G_f - \Delta G_i (\Delta G_f = \text{row}, \Delta G_i = \text{column})$$

4.4.1. K75G Tat Mutant

In Table 4.3, *K75G* mutant appears to destabilize the TAR-Tat complex in the All Edge, Diag @74, and All Diagonal configurations compared to the Native All Edge (NatAE) models, i.e. a shift towards more positive energy values and less exothermic. When compared to the Diag @74 (Nat) models, with the exception of Diag @74_76 (*K75G*) and Diag @74_78 (*K75G*), the destabilization effect of Gly75 becomes the general trend in that *K75G* mutants can destabilize the binding complexes. However, there is some indication that increases in flexibility have a small stabilization effect: the double diagonal configurations of *K75G* (at 74_76 and 74_78) seem to stabilize the binding on the order of -0.1 kcal/mol, at least in comparison to All Edge (Nat) models and Diag @74 (Nat).

Table 4.3. Binding energy difference (kcal/mol) between Native (Nat) and mutant (*K75G*) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row)

Binding Energy Difference (kcal/mol) for Native and <i>K75G</i> Complexes	All Edge (Nat)	Diag @74 (Nat)	Diag @74_76 (Nat)	Diag @74_78 (Nat)	All Diagonal (Nat)
All Edge (<i>K75G</i>)	0.6	0.4	1.3	1.1	0.9
Diag @74 (<i>K75G</i>)	0.6	0.4	1.3	1.1	0.9
Diag @74_76 (<i>K75G</i>)	-0.1	-0.3	0.6	0.4	0.2
Diag @74_78 (<i>K75G</i>)	-0.1	-0.3	0.6	0.4	0.2
All Diagonal (<i>K75G</i>)	0.6	0.1	0.2	0.8	0.6

Figure 4.11. Energy distributions of Native and *K75G* for same-configuration species. There are three sets of special configurations: All Edge, Diagonal at position 78, two diagonals at positions 74 and 76, and two diagonals at positions 74 and 78.

In Figure 4.10, we attempt to make detailed direct comparison of the energy distributions that only account for different potentials (see first four diagonal values in Table 4.3). Visually, the differences here are subtle.

4.4.2. *R78G* Tat Mutant

In Table 4.5, the overall trend is that *R78G* mutants destabilize the TAR-Tat complex, regardless of any special configurations. It is clear that substitution of Gly at the 78th position, regardless of any change in flexibility indicates significant destabilization.

Table 4.4. Binding energy difference (kcal/mol) between Native (Nat) and mutant (*R78G*) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).

Binding Energy Difference (kcal/mol) for Native and <i>R78G</i> Complexes	All Edge (Nat)	Diag @78 (Nat)	Diag @74_76 (Nat)	Diag @74_78 (Nat)	All Diagonal (Nat)
All Edge (<i>R78G</i>)	1.2	1.5	1.9	1.7	1.5
Diag @78 (<i>R78G</i>)	1.3	1.6	2	1.8	1.6
Diag @74_76 (<i>R78G</i>)	1.1	1.4	1.8	1.6	1.4
Diag @74_78 (<i>R78G</i>)	1.1	1.4	1.8	1.6	1.4
All Diagonal (<i>R78G</i>)	1.3	1.6	2.0	1.8	1.6

Figure 4.12. Energy distributions of Native and *R78G* for same-configuration species. There are three sets of special configurations: All Edge, Diagonal at position 78, two diagonals at positions 74 and 76, and two diagonals at positions 74 and 78.

Compared to the energy distributions *for K75G* against Native (Figure 4.10), the energy distributions for *R78G* (Figure 4.11) show significant shifts away from those for the native species. This could largely result from the nucleotide-amino acid interacting potentials, where highly favored Arg was replaced by the much less favored Gly.

4.4.3. *K75G-R78G* Tat Mutant

Table 4.5 shows that the *K75G-R78G* mutants can also destabilize the TAR-Tat complexes similarly to the case of *R78G* species, where *R78G* destabilization is

dominant. The energy difference in this mutant species appear higher than that of *R78G* species (Table 4.5), so it appears that the double Gly substitution can also destabilize the binding complex, with *R78G* most likely being the dominant factor.

Table 4.5. Binding energy difference (kcal/mol) between Native (Nat) and mutant (*K75G-R78G*) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).

Binding Energy Difference (kcal/mol) for Native and <i>K75G-R78G</i> Complexes	All Edge (Na)	Diag @74_76 (Nat)	Diag @74_78 (Nat)	All Diagonal (Nat)
All Edge (<i>K75G-R78G</i>)	1.7	2.4	2.2	2.0
Diag @74_76 (<i>K75G-R78G</i>)	1.4	2.1	1.9	1.7
Diag @74_78 (<i>K75G-R78G</i>)	1.5	2.2	2.0	1.8
All Diagonal (<i>K75G-R78G</i>)	1.5	2.2	2.0	1.8

Figure 4.12 shows that the energy distributions for *K75G-R78G*, similar to those of *R78G* (Figure 4.11), also experience shifts in energy distribution when compared to Native species. This could very likely result from the substitution of Gly for Arg at the 78th position.

Figure 4.13. Energy distributions of Native and *K75G-R78G* for same-configuration species. There are three sets of special configurations: All Edge, Diagonal at position 78, two diagonals at positions 74 and 76, and two diagonals at positions 74 and 78.

For all cases examined above, we assume the potential effects, i.e. Lys and Gly are both non-preferential and Arg is preferential for binding to RNA, play a major role in calculating the energy distributions for these sets of special configurations. In our limited set of potentials, in the major groove, Lys and Gly have zero interacting potentials, but Arg does not. Consequently, replacing Lys with Gly in the major groove cannot change the overall binding energy, while replacing Arg with Gly significantly changes the energy. This potential effect, as a result, explains why when investigating the smaller population of Tat species in the major

groove space only, one can find that the native and *K75G* share the same energy distributions, and so do *R78G* and *K75G-R78G* (see Appendix A). Therefore, it is worth further examining the flexibility effects by comparing each Tat mutant to itself in different configurations to isolate the potential effects between mutant and native species.

4.4.4. Flexibility Effects

The change in local Tat flexibility, involving each mutant, can be made by introduction of diagonals at key positions. For *K75G* mutant in Table 4.6, increasing the number of diagonal moves for Tat lattice structure from All Edge and one diagonal only, appears to enhance the stability of the TAR-Tat complex.

Destabilization effects are noted when comparing All Diagonal models to 2

Diagonals ones, i.e. higher flexibility vs. lower flexibility. Note that values less than -0.60 kcal/mol indicate increasing binding stabilization (Hsieh et al, 2002).

Table 4.6. Binding energy difference (kcal/mol) between different mutant (*K75G*) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).

Binding Energy Comparison between <i>K75G</i> models	All Edge	Diag @74	Diag @74_76	Diag @74_78	All Diagonal
All Edge	0	---	---	---	---
Diag @74	0.00	0	---	---	---
Diag @74_76	-0.70	-0.70	0	---	---
Diag @74_78	-0.70	-0.70	0.00	0	---
All Diagonal	-0.30	-0.30	0.40	0.40	0

K75G Tat peptides in the major groove also shows the same overall destabilization and limited stabilization effects (see Appendix A).

Note that flexibility effects in regard to enhancing possibilities for stability around the 78th position are minimal (see Table 4.7). Even when some stabilization effects occur, it is less pronounced than the *K75G* mutants.

Table 4.7. Binding energy difference (kcal/mol) between different mutant (*R78G*) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).

Binding Energy Comparison between <i>R78G</i> models	All Edge	Diag @78	Diag @74_76	Diag @74_78	All Diagonal
All Edge	0	---	---	---	---
Diag @78	0.10	0	---	---	---
Diag @74_76	-0.10	-0.20	0	---	---
Diag @74_78	-0.10	-0.20	0.00	0	---
All Diagonal	0.10	0.00	0.20	0.20	0

By looking at the $\Delta\Delta G$ values for the double Gly Tat mutants in Table 4.8, it appears clearer that with a structure of two diagonals after 74 and 76 or 74 and 78 for all three Tat mutants, some stabilization effects exist. This could result from the fact that by fixing the edges and introducing diagonals after 74, 76, and 78, one could make the peptide flexible enough to gain more access into the major groove. However if we facilitate the flexibility by introducing the Gly at both positions, removing Arg interactions significantly mask the flexibility.

Table 4.8. Binding energy difference (kcal/mol) between different mutant (*K75G-R78G*) TAR-Tat complexes. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).

Binding Energy Comparison between <i>K75G-R78G</i> models	All Edge	Diag @74_76	Diag @74_78	All Diagonal
All Edge	0	---	---	---
Diag @74_76	-0.30	0	---	---
Diag @74_78	-0.20	0.10	0	---
All Diagonal	-0.20	0.10	0.00	0

Appendix A shows the same overall destabilization for all species in the major groove space, although some flexibility can be achieved by introducing some diagonals into the system. Note even *R78G* and *K75G-R78G* species can now achieve some binding stabilization (see Appendix A).

It is worth noting that because of the ‘2-residues per move’ approach that makes 75 a midpoint between 74 and 76, the program cannot directly put a diagonal move before or after 75. Besides, with a small of population of species in the major groove space, the native and *K75G* species share the same energy distributions for different scenarios, and so do *R78G* and *K75G-R78G* (see Appendix A). This happened because of the limited potential map, which only has the interacting potentials of Arg in the major groove space, so the overall energy only changes with the removal of Arg, whereas others do not matter.

Chapter 5

Discussion

It is worth noting that when building lattice models, one has to make a number of assumptions. For this project, the biggest assumption is that each backbone atom, for both RNA and peptide, occupies its own excluded-volume space, but no consideration of specifically sufficient space for its side chain packing. We also assume the same configurational space for both native and mutant Tat peptides. This is important because it can help reduce computational runtime by avoiding any structural and dynamic optimization for atomic level features and provides a faster method to evaluate the energy for each Tat species. In addition, we assume that Arg70 and Arg80 of Tat are always in a range of fixed positions relative to G14 and U10 of TAR RNA. With this relaxed constraint, the Tat peptide can significantly explore the lattice space, including opposite to the major groove. However, because we only put constraints for Arg70 and Arg80, other Tat residues are able to move out of the major groove. Therefore, we have Tat peptides at positions other than the major groove. And we were able to validate our lattice structures by aligning them with the NMR structures as shown in the Results section.

We have also shown that the ‘1-residue per move’ approach for Tat peptides may be more amenable to modeling native-like structures, but there is a potential challenge in this regard. The Results section shows that there is potentially a very large upper limit of all possible configurations. Even the application of flexibility constraints can also be problematic in this regard given the combinatorics. Of course,

‘1-residue per move’ will address the odd-even issues involving introducing flexibility at 75, which currently in ‘2-residues per move’ is fixed as a midpoint. Moreover, the ‘1-residue per move’ may ameliorate some of the side chain packing issues but not all. To answer this definitively will require at least some all-atom modeling. There are a variety of methods that add all-atoms to reduced representation and optimize the packing of side chains, such as PULCHRA for protein and peptide structures (Rotkiewicz and Skolnick, 2008). The packing of side chains may play a key role in peptide conformational entropy. For peptide binding to nucleic acid, there is a possible direct relationship between the total entropy of binding and the conformational entropy, where the latter can determine the magnitude of the former (Tzeng and Kalodimos, 2012). In the future, we can explore the related variation in the density of states with respect to the introduction of flexibility at key peptide positions.

Because of a very large number of configurations, regardless of approaches, all-atom modeling becomes an issue of efficiency, thus requiring sampling as opposed to complete enumeration. Regarding possible Tat peptide configurations, by limiting the moves to all edges, we can significantly reduce the number of them from roughly 12 million to around 54 thousand (in ‘2-residues per move’), and hopefully the same extensive decrease can be seen in ‘1-residue per move’ lattice structures. From the new number of configurations, modeling all-atom peptides can be straightforward given the current tools. However, prediction of an arbitrary RNA structure could be extremely hard to solve (Sripakdeevong et al., 2012). There are no

automatic or even semi-automatic tools to add all-atoms to a lattice RNA model. Current methods still require some sampling and largely depend on previously solved structures as templates. RNA molecules are extremely flexible with diverse secondary and tertiary structures (Schroeder et al., 2004). For our models, because the configurations of bulges can have a direct relationship with peptide flexibility and binding strength, i.e. each bulge configuration can be accompanied by different peptide structures, it should be the focus for all-atom RNA modeling to examine the molecular dynamic effects. Currently, we found at least 240 unique bulge topologies in the low-energy binding complexes (Lustig, B.; Singh, H. San Jose State University. Unpublished Work. 2013; see Appendix C), to which we can apply all-atom modeling if given a suitable tool.

Each coarse-grained methodology comes with its own advantages and disadvantages. One problematic feature involves interacting potentials (Cragnolini et al., 2013). Applied in our work here are the nucleotide-amino acid potentials based on the limited set of RNA-protein available at the time, which may not well characterize RNA binding potentials of Lys and Gly (Lustig et al., 1997). There have been more recent efforts to come up with such statistical potentials (Perez-Cano et al., 2010). Interestingly, current aggregation of RNA-protein complexes (at least some 300) continues to show for the major groove dominant contacts for G-Arg contacts as well as apparent strong G-Lys ones (Lustig, B.; Kimura, T. San Jose State University. Unpublished Data. 2015). Also, minor groove contacts show no significant specificity to binding any one of the RNA quartet of bases. Besides, it is worth

noting that Perez-Cano et al. (2010) published their nucleotide-residue potentials in graphical form. Their original heat map color intensities require assumption in interpretation, and such conversion does not guarantee how consistent the values are to colors. Other potentials have been implemented (Tuszynska and Bujnicki, 2011), but not broadly utilized. Therefore, we hope for a database of much improved potential values, so we can re-evaluate the energies of our lattice structures and/or apply them to all-atom modeling.

The introduction of all-atoms into these coarse-grained models, with improved potentials, may yield significant insight into the role of flexibility in RNA-protein binding. Such an approach may even offer insights into the development of anti-viral drug candidates.

Chapter 6

Conclusion

The project has successfully resurrected and modified the earlier FORTRAN program. Python scripts were also developed to better assist the extensive data analyses required. From the ‘2-residues per move’ approach, we generated 12,047,142 configurations, and from the ‘1-residue per move,’ we found at least the lower bound of 120 million structures. The former approach has been thoroughly described and analyzed.

The energy distributions for both native and mutant Tat peptides in one specific set of moves (all moves, all edges, and others) have shown the overall trend of destabilization in binding. However, it appears some limited stabilization, due to flexibility by introducing Gly at Lys75, is indicated. It appears here that the mutant structures at 78 (*R78G*) dominate the double substitution of Gly (*K75G-R78G*), resulting in comparable destabilization. And there is some preliminary evidence that the population of accessible states involving residue 75 is more limited than 78.

Lastly, we have provided the framework and code for future studies to expand on our methods and results. Hopefully, the obtained results can be useful for designing and interpreting *in vitro* experiments. And it appears our methodology may allow an unbiased exploration of the density of interacting states, including for BIV TAR-Tat transition states.

Chapter 7

Future Studies

Current approaches can be expanded by the following:

- Develop new constraints and conditions to better filter native-like structures.
Apply additional constraints into the ‘1-residue per move’ algorithm to shorten computational time and reduce the number of ‘decoy’ configurations.
- Apply new statistical Nucleotide-Amino acids potentials into energy calculations.
- Model possible low-energy transition states of Tat peptide.
- Model all-atom BIV TAR RNA from low-energy lattice structures.
- Facilitate large-scale data analyses by incorporating new fast hard-disk technology.

References

- Alberts, B.; Johnson, A.; Lewis, J.; Raff, M.; Walter, P. *Molecular biology of the cell*, 5th ed.; Garland Science: New York, 2008.
- Bogerd, H. P.; Wiegand, H. L.; Bieniasz, P. D.; Cullen, B. R. *J. Virol.* **2000**, *74*, 4666-4671.
- Brock, D. C. *Understanding Moore's law*, 2006 ed.; Chemical Heritage Foundation: Philadelphia, 2006; 37-54.
- Covell, D. G.; Jernigan, R. L. *Biochemistry* **1990**, *29*, 3287-3294.
- Cragolini, T.; Derreumaux, P.; Pasquali, S. *J. Phys. Chem. B* **2013**, *117*, 8047-8060.
- Crick, F. *Nature* **1970**, *227*, 561-563.
- Dale, G. E.; Oefner, C.; D'Arcy, A. *J. Struct. Biol.* **2003**, *142*, 88-97.
- Dill, K. A.; MacCallum, J. L. *Science* **2012**, *338*, 1042-1046.
- Erdmann, V.; Barciszewska, M.; Hochberg, A.; de Groot, N.; Barciszewski, J. *Cell Mol. Life Sci.* **2001**, *58*, 960-977.
- Fanales-Belasio, E.; Raimondo, M.; Suligoi, B.; Buttò, S. *Ann. Ist. Super. Sanità* **2010**, *46*, 5-14.
- Flory, P. Chapter X: CONFIGURATIONS OF POLYMER CHAINS. In *Principles of Polymer Chemistry*, 1st ed.; Cornell University Press: New York, 1953.
- Fohrer, J.; Hennig, M.; Carlomagno, T. *J. Mol. Biol.* **2006**, *356*, 280-287.
- Futaki, S.; Suzuki, T.; Ohashi, W.; Yagami, T.; Tanaka, S.; Ueda, K.; Sugiura, Y. *J. Biol. Chem.* **2000**, *276*, 5836-5840.
- Gillet, V. J.; Leach, A. R. Chemoinformatics. *Elsevier* **2007**, 235-264.
- GNU Compiler Collection. Optimize Options.
<https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html> (accessed 2015)
- Greenbaum, N. L. *Structure* **1996**, *4*, 5-9.
- Hsieh, M.; Collins, E. D.; Blomquist, T.; Lustig, B. *J. Biomol. Struct. Dyn.* **2002**, *20*, 243-251.

- Jones, S.; Daley, D. T. A.; Luscombe, N. M.; Berman, H. M.; Thornton, J. M. *Nucleic Acids Res.* **2001**, *29*, 943-954.
- Kaufmann, K. W.; Lemmon, G. H.; DeLuca, S. L.; Sheehan, J. H.; Meiler, J. *Biochemistry* **2010**, *49*, 2987-2998.
- Kitano, H. *Nature* **2002**, *420*, 206-210.
- Kmiecik, S.; Kolinski, A. *Proc. Natl. Acad. Sci. USA* **2007**, *104*, 12330-12335.
- Kmiecik, S.; Gront, D.; Kouza, M.; Kolinski, A. *J. Phys. Chem. B* **2012**, *116*, 7026-7032.
- Lau, K. F.; Dill, K. A. *Macromolecules* **1989**, *22*, 3986-3997.
- Lund, A.; Lund, M. Mean, Mode and Median - Measures of Central Tendency - When to use with Different Types of Variable and Skewed Distributions | Laerd Statistics. <https://statistics.laerd.com/statistical-guides/measures-central-tendency-mean-mode-median.php> (accessed 2015).
- Lustig, B.; Arora, S.; Jernigan, R. L. *Nucleic Acids Res.* **1997**, *25*, 2562-2565.
- Lustig, B.; Baharand, I.; Jernigan, R. L. *Nucleic Acids Res.* **1998**, *26*, 5212-5217.
- Lustig, B.; Covell, D. G.; Jernigan, R. L. *J. Biomol. Struct. Dyn.* **1994**, *12*, 145-161.
- Maciejczyk, M.; Spasic, A.; Liwo, A.; Scheraga, H. A. *J. Chem. Theory Comput.* **2014**, *10*, 5020-5035.
- Madani, F.; Lindberg, S.; Langel, Ü.; Futaki, S.; Gräslund, A. *J. Biophys.* **2011**, *2011*, 1-10.
- Madras, N. N.; Slade, G. The self-avoiding walk. In *Probability and Its Applications*, 1996th ed.; Birkhäuser: Boston, 1996; 1-30
- Maslah, G.; Barraud, P.; Allain, F.H-T *Cell. Mol. Life Sci.* **2013**, *70*, 1875-1895
- Montelione, G. T.; Anderson, S. *Nature Struct. Biol.* **1999**, *6*, 11-12.
- Nguyen, T. M.S. Thesis, San Jose State University, San Jose, CA, December 2012.
- Pande, V. S. Folding@home - Fighting disease with the world's largest distributed super computer. <http://folding.stanford.edu/home> (accessed 2015).

- Perez-Cano, L.; Solernou, A.; Pons, C.; Fernandez-Recio, J. *PSB*, **2010**, *15*, 269-280.
- Puglisi, J. D.; Chen, L.; Blanchard, S.; Frankel, A. D. *Science* **1995**, *270*, 1200-1203.
- Puton, T.; Kozłowski, L.; Tuszynska, I.; Rother, K.; Bujnicki, J. M. *J. Struct. Biol.* **2012**, *179*, 261-268.
- Rizzuti, B.; Daggett, V. *Arch. Biochem. Biophys.* **2013**, *531*, 128-135.
- Roberts, R. J. *Bioinformatics* **2000**, *16*, 2-4.
- Romani, B.; Engelbrecht, S.; Glashoff, R. H. *J. Gen. Virol.* **2009**, *91*, 1-12.
- Romero, P.; Obradovic, Z.; Li, X.; Garner, E. C.; Brown, C. J.; Dunker, A. K. *Proteins: Struct., Funct., Bioinf.* **2001**, *42*, 38-48.
- Rotkiewicz, P.; Skolnick, J. *J. Comput. Chem.* **2008**, *29*, 1460-1465.
- Sripakdeevong, P.; Beauchamp, K.; Das, R. Why Can't We Predict RNA Structure At Atomic Resolution? In *RNA 3D Structure Analysis And Prediction*; Leontis, N.; Westhof, E., Ed.; Springer: Berlin Heidelberg, 2012; Vol. 27; pp. 43-65.
- Tozzini, V. *Curr. Opin. Struct. Biol.* **2005**, *15*, 144-150.
- Tuszynska, I.; Bujnicki, J. M. *BMC Bioinformatics* **2011**, *12*, 1-16
- Tzeng, S. R.; Kalodimos C. G. *Nature* **2012**, *488*, 236-240.
- Weeks, K. M.; Crothers, D. M. *Biochemistry* **1992**, *31*, 10281-10287.
- Wild, D. L.; Tucker, P. A.; Choe, S. *J. Mol. Graph.* **1995**, *13*, 291-298.
- Will, C. L.; Luhrmann, R. *Cold Spring Harbor Perspectives in Biology* **2010**, *3*, a003707-a003707.
- Yao, J. T.; Vasilakos, A. V.; Pedrycz, W. *IEEE Trans. Cybern.* **2013**, *43*, 1977-1989.
- Ye, X.; Kumar, R. A.; Patel, D. J. *Chem. Biol.* **1995**, *2*, 827-840.

Appendices

Appendix A Additional Tables and Figures

Besides the median energy values (kcal/mol) used in the results, there are other energy values from both the histogram distributions and raw output energies.

Table A.1. Energy values (kcal/mol) from energy distributions (histograms) for each type of configurations and each Tat species.

Configurations	Species	Mean Histogram Energy (kcal/mol)	Median Energy (kcal/mol)	Energy Mode (kcal/mol)
All moves	Native	-4.79	-8.30	-3.80
	<i>K75G</i>	-4.66	-7.90	-3.80
	<i>R78G</i>	-4.12	-6.50	-3.00
	<i>K75G-R78G</i>	-4.02	-6.50	-3.20
All Edge	Native	-5.80	-7.50	-6.40
	<i>K75G</i>	-5.63	-6.90	-6.40
	<i>R78G</i>	-4.97	-6.30	-3.60
	<i>K75G-R78G</i>	-4.80	-5.80	-3.60
Diagonals @74 & 76	Native	-5.77	-8.20	-5.60
	<i>K75G</i>	-5.60	-7.60	-4.20
	<i>R78G</i>	-4.95	-6.40	-3.60
	<i>K75G-R78G</i>	-4.78	-6.10	-3.60
Diagonals @74 & 78	Native	-5.45	-8.00	-5.00
	<i>K75G</i>	-5.28	-7.60	-3.60
	<i>R78G</i>	-4.85	-6.40	-5.00
	<i>K75G-R78G</i>	-4.69	-6.00	-3.60
Diagonal @74	Native	-5.80	-7.30	-3.80
	<i>K75G</i>	-4.92	-6.90	-3.80
Diagonal @78	Native	-4.88	-7.80	-3.80
	<i>R78G</i>	-4.30	-6.20	-3.40
All Diagonals	Native	-4.10	-7.80	-3.20
	<i>K75G</i>	-4.01	-7.20	-3.20
	<i>R78G</i>	-3.57	-6.20	-3.20
	<i>K75G-R78G</i>	-3.48	-6.00	-3.20

Table A.2. Energy values (kcal/mol) from energy output files after running the main program and/or energy calculation program for each type of configuration and each Tat species.

Configurations	Species	Energy Mean (kcal/mol)	Median Energy (kcal/mol)	Energy Mode (kcal/mol)
All moves	Native	-4.70	-4.49	-3.31
	<i>K75G</i>	-4.56	-4.36	-3.31
	<i>R78G</i>	-4.02	-3.89	-2.86
	<i>K75G-R78G</i>	-3.93	-3.79	-3.43
All Edge	Native	-5.71	-5.64	-3.60
	<i>K75G</i>	-5.53	-5.54	-3.60
	<i>R78G</i>	-4.88	-4.85	-3.43
	<i>K75G-R78G</i>	-4.71	-4.69	-4.85
Diagonals @74 & 76	Native	-5.68	-5.60	-3.60
	<i>K75G</i>	-5.50	-5.44	-5.62
	<i>R78G</i>	-4.86	-4.80	-4.91
	<i>K75G-R78G</i>	-4.68	-4.80	-4.91
Diagonals @74 & 78	Native	-5.35	-5.15	-4.97
	<i>K75G</i>	-5.19	-4.98	-4.56
	<i>R78G</i>	-4.76	-4.71	-4.85
	<i>K75G-R78G</i>	-4.59	-4.56	-4.85
Diagonal @74	Native	-4.98	-4.75	-3.67
	<i>K75G</i>	-4.83	-4.60	-3.67
Diagonal @78	Native	-4.79	-4.54	-3.61
	<i>R78G</i>	-4.20	-4.03	-3.61
All Diagonals	Native	-4.00	-3.65	-2.78
	<i>K75G</i>	-3.91	-3.57	-2.78
	<i>R78G</i>	-3.48	-3.26	-2.84
	<i>K75G-R78G</i>	-3.38	-3.19	-2.84

Table A.3. Binding energy difference (kcal/mol) between Native (Nat) and mutant (*K75G*) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).

Binding Energy Difference (kcal/mol) for Native and <i>K75G</i> Complexes	All Edge (Nat)	Diag @74 (Nat)	Diag @74_76 (Nat)	Diag @74_78 (Nat)	All Diagonal (Nat)
All Edge (<i>K75G</i>)	0.0	0.3	0.3	0.5	-2.0
Diag @74 (<i>K75G</i>)	0.3	0.0	0.6	0.8	-1.7
Diag @74_76 (<i>K75G</i>)	-0.3	-0.6	0.0	0.2	-2.3
Diag @74_78 (<i>K75G</i>)	-0.5	-0.8	-0.2	0.0	-2.5
All Diagonal (<i>K75G</i>)	2.0	1.7	2.3	2.5	0.0

Table A.4. Binding energy difference (kcal/mol) between different mutant (*K75G*) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).

Binding Energy Comparison between <i>K75G</i> models	All Edge	Diag @74	Diag @74_76	Diag @74_78	All Diagonal
All Edge	0	---	---	---	---
Diag @74	0.3	0	---	---	---
Diag @74_76	-0.3	-0.6	0	---	---
Diag @74_78	-0.3	-0.8	-0.2	0	---
All Diagonal	2.0	1.7	2.3	2.5	0

Table A.5. Binding energy difference (kcal/mol) between Native (Nat) and mutant (*R78G*) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).

Binding Energy Difference (kcal/mol) for Native and <i>R78G</i> Complexes	All Edge (Nat)	Diag @78 (Nat)	Diag @74_76 (Nat)	Diag @74_78 (Nat)	All Diagonal (Nat)
All Edge (<i>R78G</i>)	0.4	-0.4	0.7	0.9	-1.6
Diag @78 (<i>R78G</i>)	1.2	0.4	1.5	1.7	-0.8
Diag @74_76 (<i>R78G</i>)	0.1	-0.7	0.4	0.6	-1.9
Diag @74_78 (<i>R78G</i>)	0.4	-0.4	0.7	0.9	-1.6
All Diagonal (<i>R78G</i>)	2.2	1.4	2.5	2.7	0.2

Table A.6. Binding energy difference (kcal/mol) between different mutant (*R78G*) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).

Binding Energy Comparison between <i>R78G</i> models	All Edge	Diag @74	Diag @74_76	Diag @74_78	All Diagonal
All Edge	0	---	---	---	---
Diag @74	0.8	0	---	---	---
Diag @74_76	-0.3	-1.1	0	---	---
Diag @74_78	0.0	-0.8	0.3	0	---
All Diagonal	1.8	1.0	2.1	1.8	0

Table A.7. Binding energy difference (kcal/mol) between Native (Nat) and mutant (*K75G-R78G*) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in final state (see relevant row).

Binding Energy Difference (kcal/mol) for Native and <i>K5G-R78G</i> Complexes	All Edge (Nat)	Diag @78 (Nat)	Diag @74_76 (Nat)	Diag @74_78 (Nat)	All Diagonal (Nat)
All Edge (<i>R78G</i>)	0.4	-0.4	0.7	0.9	-1.6
Diag @78 (<i>R78G</i>)	1.2	0.4	1.5	1.7	-0.8
Diag @74_76 (<i>R78G</i>)	0.1	-0.7	0.4	0.6	-1.9
Diag @74_78 (<i>R78G</i>)	0.4	-0.4	0.7	0.9	-1.6
All Diagonal (<i>R78G</i>)	2.2	1.4	2.5	2.7	0.2

Table A.8. Binding energy difference (kcal/mol) between different mutant (*K75G-R78G*) TAR-Tat complexes in Major Groove Space. Note positive cell values correspond to destabilization as indicated in the final state (see relevant row).

Binding Energy Comparison between <i>K75G-R78G</i> models	All Edge	Diag @74	Diag @74_76	Diag @74_78	All Diagonal
All Edge	0	---	---	---	---
Diag @74	0.8	0	---	---	---
Diag @74_76	-0.3	-1.1	0	---	---
Diag @74_78	0.0	-0.8	0.3	0	---
All Diagonal	1.8	1.0	2.1	1.8	0

Figure A.1. Energy distribution for all configurations in Native and Mutant peptides (Native, *K75G*, *R78G*, and *K75G-R78G*) located in the Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

Figure A.2. Energy distribution for all configurations in Native and Mutant peptides (Native, *K75G*, *R78G*, and *K75G-R78G*) in the Non-Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

Figure A.3. Energy distribution for All-Edge configurations of Native and Mutant peptides (Native, *K75G*, *R78G*, and *K75G-R78G*) in the Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

Figure A.4. Energy distribution for configurations with diagonals after the 74th and 76th positions of Native and Mutant peptides (Native, *K75G*, *R78G*, and *K75G-R78G*) in Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

Figure A.5. Energy distribution for configurations with diagonals after the 74th and 78th positions of Native and Mutant peptides (Native, *K75G*, *R78G*, and *K75G-R78G*) in Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

Figure A.6. Energy distribution for All-Diagonal configurations positions of Native and Mutant peptides (Native, *K75G*, *R78G*, and *K75G-R78G*) in Major Groove Space. X-axis shows histogram energies (kcal/mol). Y-axis shows frequency, i.e. the number of configurations that have similar energy values within 0.2 kcal/mol intervals.

Appendix B FORTRAN Program Listings

Program: *bivtartat.f95*

```
program bivtartat
character*50 fname, matrx, jctfil, basfil
character*50 rnaseq, dnaseq, pepseq
character*50 tmpfil
character*50 tatseq,configs,energy_out
integer i,j,k
integer s1,s2,b1,b2,t1
integer s1n,s2n,b1n,b2n,t1n
integer s1n2,s2n2,b1n2,b2n2,t1n2
integer b1bp1,b2bp1,pair
integer bcnt,acnt,rnanum,pepnum
integer rmacnt,pepcnt
integer jct
integer bump1
integer histb1,histb2
integer contact
integer errname
real potent
integer*8 prcnt
dimension prcnt(11,28,32)
dimension rnanum(28)
dimension pepnum(14)
dimension s1(3,119),s2(3,119)
dimension b1(3,7),b2(3,3)
dimension t1(3,13)
dimension latmov(3,18)
dimension histb1(3),histb2(1)
dimension contact(2,5)
dimension jct(4)
dimension ptb(3),d(3)
integer rstp,move,n,ptb,d,long
integer lowsum,reqmov,remain
character*3 rna3,pep3,bname3,aname3
character*1 rna,pep,bname,aname
character*1 a,b,c
dimension rna(28),rna3(28),pep(14),pep3(14)
dimension acnt(20),aname(20),aname3(20)
dimension bcnt(4),bname(4),bname3(4)
dimension potent(8,20)

data bcnt/4*0/
```

```

data acnt/20*0/
data bname/'A','C','G','U'/
data aname/'A','C','D','E','F','G','H','I','K','L','M','N','P',&
'Q','R','S','T','V','W','Y'/
data bname3/'ADE','CYT','GUA','URA'/
data aname3/'ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS',&
'LEU','MET','ASN','PRO','GLN','ARG','SER','THR','VAL','TRP','TYR'/

s1n=12
s1n2=119
s2n=12
s2n2=119
b1n=3
b2n=1
t1n=6

b1n2=2*b1n+1
b2n2=2*b2n+1
t1n2=2*t1n+1

rncnt=s1n+s2n+b1n+b2n
pepcnt=2*t1n-1
! Make s1
  open(unit=1,file='shelix8d_2.dat',status='old')
  read(1,*) s1
  close(1)
! Make s2
  do n=1,s2n2
    s2(1,n)=-s1(1,n)+6
    s2(2,n)=-s1(2,n)+6
    s2(3,n)=s1(3,n)+8
  end do

! Make 'move' matrix
  open(unit=1,file='idcub2.dat',status='old')
  read(1,*) latmov
  close(1)

! Make junction
  open (unit=1,file='jction.dat',status='old')
  read(1,*) jct
  close(1)
! Make basepair

```



```

open(unit=1,file='bases.dat',status='old')
read(1,*) b1bp1
read(1,*) b2bp1
read(1,*) pair
close(1)
! Make contact matrix
open(unit=1,file='contact5.dat',status='old')
read(1,*) contact
close(1)

! Make contact table
open(unit=9,file='contact_table.txt',action='write',status='replace')

print*, 'Enter configs output filename:'
read(*,*) configs
open(unit=17,file=configs,action='write',status='replace')
print*, 'Enter energy output filename:'
read(*,*) energy_out
open(unit=18,file=energy_out,action='write',status='replace')

! Make RNA and peptide sequences
200 print*, 'Enter Tat sequence:'
read(*,*) tatseq
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
open(unit=1,file=tatseq,status='old',iostat=errname)
if (errname==0) then
read(1,*) pepseq
close(1)
else
print *, 'Wrong Tat sequence file. Enter again.'
goto 200
end if

! Write collection of coordinates for bases 8 and 22 (G11 and C25 base pairs)
! open(unit=22,file='base8_22.txt',action='write',status='replace')
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
open(unit=1,file='bivtar.rna',status='old')
read(1,*) rnaseq
close(1)
! Make matrix of potential
open(unit=1,file='potent4.dat',status='old')
read(1,*) potent
close(1)
!print *,pepseq

```

```

!print *,rnaseq!print *, bname
!ßprint *, aname
!print *, bname3
!print *, aname3
!-----
! Input is one-letter code.
! Count number of residues/bases of each type.
! Reassign numerical index to bases and amino acids.
! Assign three letter codes.
! One-letter symbols numbered alphabetically.
!-----
      do i=1,rnacnt
        rna(i)=rnaseq(i:i)
        do j=1,4
          if (rna(i).eq.bname(j)) then
            rna3(i)=bname3(j)
            bcnt(j)=bcnt(j)+1
            rnanum(i)=j
          end if
        end do
      end do
      do j=1,pepcnt
        pep(j)=pepseq(j:j)
        do k=1,20
          if (pep(j).eq.aname(k)) then
            pep3(j)=aname3(k)
            acnt(k)=acnt(k)+1
            pepnum(j)=k
          end if
        end do
      end do

      call bulge1(s1,s1n,s2,s2n,b1,b1n,b2,b2n,jct,contact,b1bp1,b2bp1,&
        t1,t1n,latmov,rnacnt,pepcnt,prcnt,potent,rnanum,pepnum,s1n2,s2n2,&
        b1n2,b2n2,t1n2,pair)

!   write(9,'(8x,14a12)') (pep(j),j=1,pepcnt)
      write(9,'(8x,15i12)') ((i+69),i=1,pepcnt)
      k=32
      do i=1,rnacnt
        write(9,'(a4,i4,14i12)') rna(i),i,(prcnt(j,i,k),j=1,pepcnt)
      end do
    end program bivtartat

```



```

b1(1,1)=s1(1,jct(1))
b1(2,1)=s1(2,jct(1))
b1(3,1)=s1(3,jct(1))
ptb(1)=s2(1,jct(2))
ptb(2)=s2(2,jct(2))
ptb(3)=s2(3,jct(2))

rstp=1
move=1

100 b1(1,2*rstp+1)=b1(1,2*rstp-1)+latmov(1,move)
   b1(2,2*rstp+1)=b1(2,2*rstp-1)+latmov(2,move)
   b1(3,2*rstp+1)=b1(3,2*rstp-1)+latmov(3,move)
   b1(1,2*rstp)=b1(1,2*rstp-1)+latmov(1,move)/2
   b1(2,2*rstp)=b1(2,2*rstp-1)+latmov(2,move)/2
   b1(3,2*rstp)=b1(3,2*rstp-1)+latmov(3,move)/2
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   d(1)=abs(b1(1,2*rstp+1)-ptb(1))
   d(2)=abs(b1(2,2*rstp+1)-ptb(2))
   d(3)=abs(b1(3,2*rstp+1)-ptb(3))
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   long=max(d(1),d(2),d(3))
   lowsum=(d(1)+d(2)+d(3))-long
   if (long.ge.lowsum) then
       reqmov=long
   else if (lowsum.gt.long) then
       reqmov=lowsum
   end if
   remain=2*(b1n-rstp+1)

   if (remain.lt.reqmov) goto 50
   call chk1(s1,s1n,s2,s2n,b1,b1n,rstp,bump1,jct,s1n2,s2n2,b1n2)

   if (bump1.eq.0) goto 40
   if (bump1.eq.1) goto 50

40 histb1(rstp)=move
   if (rstp.ne.b1n) then
       rstp = rstp + 1
       move = 1
       goto 100
   else if (rstp.eq.b1n) then
!   print *, 'Call bulge 2'
       call bulge2(s1,s1n,s2,s2n,b1,b1n,histb1,b2,b2n,jct,contact,&

```

```

        b1bp1,b2bp1,latmov,rnacnt,pepcnt,rnanum,pepnum,potent,s1n2,&
        s2n2,b1n2,b2n2,t1,t1n,t1n2,pair,prcnt)
!      print *, 'After bulge 2'
      if (move.ne.18) then
        move=move+1
        goto 100
      else if (move.eq.18) then
        goto 20
      end if
    end if

50 if (move.ne.18) then
  move=move+1
  goto 100
else if (move.eq.18) then
  goto 20
end if

20 if (rstp.ne.1) then
  rstp=rstp-1
  move=histb1(rstp)
  if (move.ne.18) then
    move=move+1
    goto 100
  else if (move.eq.18) then
    goto 20
  end if
else if (rstp.eq.1) then
  goto 999
end if
999 return

end subroutine bulge1

```

```

!!!!!!!!!!!!SUBROUTINE BULGE 2!!!!!!!!!!!!!!!!!!!!!!
subroutine bulge2(s1,s1n,s2,s2n,b1,b1n,histb1,b2,b2n,jct,&
contact,b1bp1,b2bp1,latmov,rnacnt,pepcnt,rnanum,&
pepnum,potent,s1n2,s2n2,b1n2,b2n2,t1,t1n,t1n2,pair,prcnt)
integer s1,s2,b1,b2,histb1,contact
dimension s1(3,119),s2(3,119)
dimension b1(3,7),b2(3,3)
dimension histb1(3)
integer s1n,s2n,b1n,b2n,jct,rnacnt,pepcnt,rnanum,pepnum
integer s1n2,s2n2,b1n2,b2n2

```

```

integer b1bp1,b2bp1
integer t1,t1n,t1n2,pair
integer*8 prcnt
dimension prcnt(11,28,32)
real potent
dimension latmov(3,18)
dimension jct(4)
dimension contact(2,5)
dimension rnanum(rnact),pepnum(pepcnt)
dimension t1(3,t1n2)
dimension potent(8,20)
! Local Variables
integer histb2
integer rstp,move,bump3,paired,closed,n,pta,ptb,&
long,lowsum,reqmov,remain,d
dimension histb2(1),pta(3),ptb(3),d(3)

do n=1,b2n
histb2(n)=0
end do
b2(1,1)=s2(1,jct(3))
b2(2,1)=s2(2,jct(3))
b2(3,1)=s2(3,jct(3))
ptb(1)=s1(1,jct(4))
ptb(2)=s1(2,jct(4))
ptb(3)=s1(3,jct(4))

rstp=1
move=1

100 b2(1,2*rstp+1)=b2(1,2*rstp-1)+latmov(1,move)
b2(2,2*rstp+1)=b2(2,2*rstp-1)+latmov(2,move)
b2(3,2*rstp+1)=b2(3,2*rstp-1)+latmov(3,move)
b2(1,2*rstp)=b2(1,2*rstp-1)+latmov(1,move)/2
b2(2,2*rstp)=b2(2,2*rstp-1)+latmov(2,move)/2
b2(3,2*rstp)=b2(3,2*rstp-1)+latmov(3,move)/2

d(1)=abs(b2(1,2*rstp+1)-ptb(1))
d(2)=abs(b2(2,2*rstp+1)-ptb(2))
d(3)=abs(b2(3,2*rstp+1)-ptb(3))

long=max(d(1),d(2),d(3))
lowsum=(d(1)+d(2)+d(3))-long
if (long.ge.lowsum) then

```

```

    reqmov=long
else if (lowsum.gt.long) then
    reqmov=lowsum
end if

remain=2*(b2n-rstp+1)

if (remain.lt.reqmov) goto 50
call chk3 (s1,s1n,s2,s2n,b1,b1n,b2,b2n,jct,rstp,bump3,&
s1n2,s2n2,b1n2,b2n2)

if (bump3.eq.0) goto 40
if (bump3.eq.1) goto 50
40 histb2(rstp) = move

if (rstp.ne.b2n) then
    rstp=rstp+1
    move=1
    goto 100
else if (rstp.eq.b2n) then
    if (pair.eq.1) then
        call basepr(b1,b1n,b2,b2n,b1bp1,b2bp1,paired)
    else if (pair.eq.0) then
        paired=1
    end if
    if (paired.eq.1) then
!       print *, 'Paired'
!       print *, 'Call tatmov'
        call tatmov(s1,s1n,s2,s2n,b1,b1n,b2,b2n,histb1,histb2,&
latmov,t1,t1n,contact,rnacnt,pepcnt,rnanum,pepnum,s1n2,&
s2n2,b1n2,b2n2,t1n2,potent,prcnt)
!       print *, 'After tatmov'
    end if
    if (move.ne.18) then
        move=move+1
        goto 100
    else if (move.eq.18) then
        goto 20
    end if
end if

50 if (move.ne.18) then
    move=move+1
    goto 100

```



```

(b1(3,2*rstp+1).eq.s1(3,n))) goto 100
end do

if (rstp.ne.1) then
  do n=s1n+1,s1n2
    if ((b1(1,2*rstp).eq.s1(1,n)).and.&
(b1(2,2*rstp).eq.s1(2,n)).and.&
(b1(3,2*rstp).eq.s1(3,n))) goto 100
  end do
end if

do n=1,s2n
  if ((b1(1,2*rstp+1).eq.s2(1,n)).and.&
(b1(2,2*rstp+1).eq.s2(2,n)).and.&
(b1(3,2*rstp+1).eq.s2(3,n))) goto 100
end do
do n=s2n+1,s2n2
  if ((b1(1,2*rstp).eq.s2(1,n)).and.&
(b1(2,2*rstp).eq.s2(2,n)).and.&
(b1(3,2*rstp).eq.s2(3,n))) goto 100
end do
bump1=0
return
100 bump1=1
return
end subroutine chk1

!!!!!!!!!! SUBROUTINE CHK3!!!!!!!!!!!!!!!!!!!!!!
subroutine chk3 (s1,s1n,s2,s2n,b1,b1n,b2,b2n,jct,rstp,bump3,&
s1n2,s2n2,b1n2,b2n2)
integer s1,s1n,s2,s2n,b1,b1n,b2,b2n,jct,rstp,bump3,s1n2,s2n2,&
b1n2,b2n2
dimension s1(3,119),s2(3,119),b1(3,7),b2(3,3),jct(4)
! Local Variables
integer i,j,k,m
bump3=-1

!-----
! Check itself after first move.
!-----

if (rstp.ge.2) then
! print *, 'chk3 :rstp is 2 here'
do i=1,rstp-1
  if ((b2(1,2*rstp+1).eq.b2(1,2*i-1)).and.&

```

```

(b2(2,2*rstp+1).eq.b2(2,2*i-1)).and.&
(b2(3,2*rstp+1).eq.b2(3,2*i-1))) goto 77
  if ((b2(1,2*rstp).eq.b2(1,2*(i-1))).and.&
(b2(2,2*rstp).eq.b2(2,2*(i-1))).and.&
(b2(3,2*rstp).eq.b2(3,2*(i-1)))) goto 77
  end do
end if
!-----
! Skip the attachment site on s2. Skip first midpoint check to allow
! diagonal moves out from s2.
!-----
do j=1,s2n
  if (j.ne.jct(3)) then
    if ((b2(1,2*rstp-1).eq.s2(1,j)).and.&
(b2(2,2*rstp-1).eq.s2(2,j)).and.&
(b2(3,2*rstp-1).eq.s2(3,j))) goto 77
    end if
  end do
  if (rstp.ne.1) then
    do j=s2n+1,s2n2
      if ((b2(1,2*rstp).eq.s2(1,j)).and.&
(b2(2,2*rstp).eq.s2(2,j)).and.&
(b2(3,2*rstp).eq.s2(3,j))) goto 77
    end do
  end if
!-----
do k=1,b1n
  if ((b2(1,2*rstp+1).eq.b1(1,2*k+1)).and.&
(b2(2,2*rstp+1).eq.b1(2,2*k+1)).and.&
(b2(3,2*rstp+1).eq.b1(3,2*k+1))) goto 77
  if ((b2(1,2*rstp).eq.b1(1,2*k)).and.&
(b2(2,2*rstp).eq.b1(2,2*k)).and.&
(b2(3,2*rstp).eq.b1(3,2*k))) goto 77
end do
!-----
do m=1,s1n
  if ((b2(1,2*rstp+1).eq.s1(1,m)).and.&
(b2(2,2*rstp+1).eq.s1(2,m)).and.&
(b2(3,2*rstp+1).eq.s1(3,m))) goto 77
end do
do m=s1n+1,s1n2
  if ((b2(1,2*rstp).eq.s1(1,m)).and.&
(b2(2,2*rstp).eq.s1(2,m)).and.&
(b2(3,2*rstp).eq.s1(3,m))) goto 77

```

```

    end do
!-----
    bump3=0
!   print *, 'Returning from chk3 since bump3 is 0'
    return
77 bump3=1
!   print *, 'Returning IFROM chk3 since bump3 is 1'
    return
end subroutine chk3

!!!!!!!!!!!!!!!!SUBROUTINE BASEPR!!!!!!!!!!!!!!!!!!
    subroutine basepr(b1,b1n,b2,b2n,b1bp1,b2bp1,paired)
    integer b1,b1n,b2,b2n,b1bp1,b2bp1,paired
    dimension b1(3,7),b2(3,3)
! Local Variables
    integer d
    dimension d(3)

    paired=-1
    d(1)=abs(b2(1,2*b2bp1+1)-b1(1,2*b1bp1+1))
    d(2)=abs(b2(2,2*b2bp1+1)-b1(2,2*b1bp1+1))
    d(3)=abs(b2(3,2*b2bp1+1)-b1(3,2*b1bp1+1))

!-----
! Base pairing distance is tested for 3,3,3 units and total of less
! than 5 units rectilinear.
!-----
    if (((d(1).le.6).and.(d(2).le.6).and.(d(3).le.6)).and.&
        (((d(1)+d(2)+d(3)).ge.6).and.((d(1)+d(2)+d(3)).le.10))) then
        paired=1
    else
        paired=0
    end if
end subroutine basepr

!!!!!!!!!!!!!!!!!!!!!!!! SUBROUTINE TATMOV!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine tatmov(s1,s1n,s2,s2n,b1,b1n,b2,b2n,histb1,histb2,&
        latmov,t1,t1n,contact,rnact,pepcnt,rnanum,pepnum,s1n2,&
        s2n2,b1n2,b2n2,t1n2,potent,prcnt)
    integer s1,s1n,s2,s2n,b1,b1n,b2,b2n,histb1,histb2,&
        latmov,t1,t1n,contact,rnact,pepcnt,rnanum,pepnum,s1n2,&
        s2n2,b1n2,b2n2,t1n2
    integer*8 prcnt
    dimension prcnt(11,28,32)

```

```

real potent
integer pepmov
dimension pepmov(3,18)
dimension s1(3,s1n2),s2(3,s2n2)
dimension b1(3,b1n2),b2(3,b2n2)
dimension t1(3,t1n2)
dimension latmov(3,18),contact(2,5)
dimension rnanum(rnact),pepnum(pepcnt)
dimension histb1(b1n),histb2(b2n)
dimension potent(8,20)

! Local Variables
integer histt1
integer tstp,move,bump4,far2
integer n,i,first
integer d
integer pta,ptb
integer idx
integer long,lowsum,reqmov,remain
dimension histt1(t1n)
dimension d(3)
dimension pta(3),ptb(3)
!   dimension idx(7)

open(unit=1,file='idcub2.dat',status='OLD')
read(1,*) pepmov
close(1)

first=1
do n=1,t1n
histt1(n)=0
end do

! First contact
i=1
if (contact(1,i).eq.1) then
t1(1,1)=s1(1,contact(2,i))
t1(2,1)=s1(2,contact(2,i))
t1(3,1)=s1(3,contact(2,i))
else if (contact(1,1).eq.2) then
t1(1,1)=s2(1,contact(2,i))
t1(2,1)=s2(2,contact(2,i))
t1(3,1)=s2(3,contact(2,i))
else if (contact(1,1).eq.3) then

```

```

    t1(1,1)=b1(1,2*contact(2,i)-1)
    t1(2,1)=b1(2,2*contact(2,i)-1)
    t1(3,1)=b1(3,2*contact(2,i)-1)
else if (contact(1,1).eq.4) then
    t1(1,1)=b2(1,2*contact(2,i)-1)
    t1(2,1)=b2(2,2*contact(2,i)-1)
    t1(3,1)=b2(3,2*contact(2,i)-1)
else
    goto 999
end if
! Second contact
i=2
if (contact(1,i).eq.1) then
    ptb(1)=s1(1,contact(2,i))
    ptb(2)=s1(2,contact(2,i))
    ptb(3)=s1(3,contact(2,i))
else if (contact(1,2).eq.2) then
    ptb(1)=s2(1,contact(2,i))
    ptb(2)=s2(2,contact(2,i))
    ptb(3)=s2(3,contact(2,i))
else if (contact(1,2).eq.3) then
    ptb(1)=b1(1,2*contact(2,i)+1)
    ptb(2)=b1(2,2*contact(2,i)+1)
    ptb(3)=b1(3,2*contact(2,i)+1)
else if (contact(1,2).eq.4) then
    ptb(1)=b2(1,2*contact(2,i)+1)
    ptb(2)=b2(2,2*contact(2,i)+1)
    ptb(3)=b2(3,2*contact(2,i)+1)
else if (contact(1,2).eq.5) then
    ptb(1)=t1(1,2*contact(2,i)+1)
    ptb(2)=t1(2,2*contact(2,i)+1)
    ptb(3)=t1(3,2*contact(2,i)+1)
end if
!-----
    tstp=1
    move=1
!-----
! 100 t1(1,2*tstp+1)=t1(1,2*tstp-1)+latmov(1,move)
!    t1(2,2*tstp+1)=t1(2,2*tstp-1)+latmov(2,move)
!    t1(3,2*tstp+1)=t1(3,2*tstp-1)+latmov(3,move)
!    t1(1,2*tstp)=t1(1,2*tstp-1)+latmov(1,move)/2
!    t1(2,2*tstp)=t1(2,2*tstp-1)+latmov(2,move)/2
!    t1(3,2*tstp)=t1(3,2*tstp-1)+latmov(3,move)/2

```

```

100 t1(1,2*tstp+1)=t1(1,2*tstp-1)+pepmov(1,move)
   t1(2,2*tstp+1)=t1(2,2*tstp-1)+pepmov(2,move)
   t1(3,2*tstp+1)=t1(3,2*tstp-1)+pepmov(3,move)
   t1(1,2*tstp)=t1(1,2*tstp-1)+pepmov(1,move)/2
   t1(2,2*tstp)=t1(2,2*tstp-1)+pepmov(2,move)/2
   t1(3,2*tstp)=t1(3,2*tstp-1)+pepmov(3,move)/2
!-----
   d(1)=abs(t1(1,2*tstp+1)-ptb(1))
   d(2)=abs(t1(2,2*tstp+1)-ptb(2))
   d(3)=abs(t1(3,2*tstp+1)-ptb(3))
   long=max(d(1),d(2),d(3))
   lowsum=(d(1)+d(2)+d(3))-long

   if (long.ge.lowsum) then
     reqmov=long
   else if (lowsum.gt.long) then
     reqmov=lowsum
   end if
   remain=2*(t1n-tstp+1)
   if (remain.lt.reqmov) goto 50
   call chk4(s1,s1n,s2,s2n,b1,b1n,b2,b2n,t1,t1n,contact,tstp,bump4,&
     s1n2,s2n2,b1n2,b2n2,t1n2)

   if (bump4.eq.0) goto 40
   if (bump4.eq.1) goto 50
40 histt1(tstp)=move

   if (tstp.ne.t1n) then
     tstp=tstp+1
     move=1
     goto 100
   else if (tstp.eq.t1n) then
     idx=idx+1
!   print *, idx
     call savecfg(s1,s1n,s2,s2n,b1,b2,t1,b1n,b2n,t1n,histb1,&
       histb2,histt1,idx,rnacnt,pepcnt,rnanum,pepnum,potent,&
       s1n2,s2n2,b1n2,b2n2,t1n2,prent)
     goto 50
   end if

50 if (move.ne.18) then
   move=move+1
   goto 100
else if (move.eq.18) then

```

```

        goto 20
    end if

20 if (tstp.ne.1) then
    tstp=tstp-1
    move=hist1(tstp)
    if (move.ne.18) then
        move=move+1
        goto 100
    else if (move.eq.18) then
        goto 20
    end if
else if (tstp.eq.1) then
    goto 999
end if
999 return
end subroutine tatmov

```

!!!!!!!!!!!!!! SUBROUTINE CHK4!!!!!!!!!!!!!!

```

subroutine chk4(s1,s1n,s2,s2n,b1,b1n,b2,b2n,t1,t1n,contact,tstp,bump4,&
s1n2,s2n2,b1n2,b2n2,t1n2)
integer s1,s1n,s2,s2n,b1,b1n,b2,b2n,t1,t1n,contact,tstp,bump4,&
s1n2,s2n2,b1n2,b2n2,t1n2
dimension s1(3,s1n2),s2(3,s2n2)
dimension contact(2,5)
dimension t1(3,t1n2)
dimension b1(3,b1n2),b2(3,b2n2)
bump4=-1

```

!-----
! Below does not seem to work.
! Need to check with all pre-existing structures.
!-----

```

    if (tstp.ge.2) then
        do i=1,tstp-1
            if ((t1(1,2*tstp+1).eq.t1(1,2*i-1)).and.&
(t1(2,2*tstp+1).eq.t1(2,2*i-1)).and.&
(t1(3,2*tstp+1).eq.t1(3,2*i-1))) goto 77
            if ((t1(1,2*tstp).eq.t1(1,2*(i-1))).and.&
(t1(2,2*tstp).eq.t1(2,2*(i-1))).and.&
(t1(3,2*tstp).eq.t1(3,2*(i-1)))) goto 77
        end do
    end if

```

```

end if
!-----
do j=1,s2n
  if ((t1(1,2*tstp+1).eq.s2(1,j)).and.&
(t1(2,2*tstp+1).eq.s2(2,j)).and.&
(t1(3,2*tstp+1).eq.s2(3,j))) goto 77
end do
do j=s2n+1,s2n2
  if ((t1(1,2*tstp).eq.s2(1,j)).and.&
(t1(2,2*tstp).eq.s2(2,j)).and.&
(t1(3,2*tstp).eq.s2(3,j))) goto 77
end do
!-----
do n=1,b2n
  if ((t1(1,2*tstp+1).eq.b2(1,2*n+1)).and.&
(t1(2,2*tstp+1).eq.b2(2,2*n+1)).and.&
(t1(3,2*tstp+1).eq.b2(3,2*n+1))) goto 77
  if ((t1(1,2*tstp).eq.b2(1,2*n)).and.&
(t1(2,2*tstp).eq.b2(2,2*n)).and.&
(t1(3,2*tstp).eq.b2(3,2*n))) goto 77
end do
!-----
do m=1,s1n
  if ((t1(1,2*tstp+1).eq.s1(1,m)).and.&
(t1(2,2*tstp+1).eq.s1(2,m)).and.&
(t1(3,2*tstp+1).eq.s1(3,m))) goto 77
end do
do m=s1n+1,s1n2
  if ((t1(1,2*tstp).eq.s1(1,m)).and.&
(t1(2,2*tstp).eq.s1(2,m)).and.&
(t1(3,2*tstp).eq.s1(3,m))) goto 77
end do
!-----
do k=1,b1n
  if ((t1(1,2*tstp+1).eq.b1(1,2*k+1)).and.&
(t1(2,2*tstp+1).eq.b1(2,2*k+1)).and.&
(t1(3,2*tstp+1).eq.b1(3,2*k+1))) goto 77
  if ((t1(1,2*tstp).eq.b1(1,2*k)).and.&
(t1(2,2*tstp).eq.b1(2,2*k)).and.&
(t1(3,2*tstp).eq.b1(3,2*k))) goto 77
end do
!-----
bump4=0
return

```



```

77 bump4=1
   return
end subroutine chk4

```

!!!!!!!!!!!!!! SUBROUTINE SAVECFG!!!!!!!!!!!!!!

```

subroutine savecfg(s1,s1n,s2,s2n,b1,b2,t1,b1n,b2n,t1n,histb1,&
histb2,histt1,idx,rnact,pepcnt,rnanum,pepnum,potent,s1n2,s2n2,&
b1n2,b2n2,t1n2,prcnt)
integer s1,s1n,s2,s2n,t1,s1n2,s2n2,b1n2,b2n2,t1n2,b1,b2
integer b1n,b2n,t1n,histb1,histb2,histt1,idx,i,n,x,j,k
integer rnact,pepcnt,rnanum,pepnum
integer r
integer*8 tmpsum
integer*8 prcnt
integer dist
real prsum
real potent
real dd,d
real r1,r2,r3,r4,dot
real alpha
dimension s1(3,s1n2),s2(3,s2n2)
dimension b1(3,b1n2),b2(3,b2n2),t1(3,t1n2)
dimension histb1(b1n),histb2(b2n),histt1(t1n)
dimension rnanum(rnact),pepnum(pepcnt)
dimension potent(8,20)
dimension r(3)
dimension tmpsum(pepcnt,rnact)
dimension prcnt(pepcnt,rnact,32)
dimension r1(3),r2(3),r3(3),r4(3)
real colsum,rowsum
integer colcnt,rowcnt
integer outcfg
real testcfg
dimension testcfg(11)
real,dimension(11)::nactfg
integer::major=0
integer rnaps,peppts,peppts1
dimension rnaps(3,rnact),peppts(3,pepcnt),peppts1(3,pepcnt)

```

!!!!!!!!!!!!!! Energy Calculation !!!!!!!!!!!!!!!

```

dd=1
do i=1,s1n/2
  rnaps(1,i)=s1(1,i)

```

```

    rnaps(2,i)=s1(2,i)
    rnaps(3,i)=s1(3,i)
end do
do i=1,b1n
    rnaps(1,s1n/2+i)=b1(1,2*i+1)
    rnaps(2,s1n/2+i)=b1(2,2*i+1)
    rnaps(3,s1n/2+i)=b1(3,2*i+1)
end do
do i=1,s2n
    rnaps(1,s1n/2+b1n+i)=s2(1,i)
    rnaps(2,s1n/2+b1n+i)=s2(2,i)
    rnaps(3,s1n/2+b1n+i)=s2(3,i)
end do
do i=1,b2n
    rnaps(1,s1n/2+b1n+s2n+i)=b2(1,2*i+1)
    rnaps(2,s1n/2+b1n+s2n+i)=b2(2,2*i+1)
    rnaps(3,s1n/2+b1n+s2n+i)=b2(3,2*i+1)
end do
do i=1,s1n/2
    rnaps(1,s1n/2+b1n+s2n+b2n+i)=s1(1,i+s1n/2)
    rnaps(2,s1n/2+b1n+s2n+b2n+i)=s1(2,i+s1n/2)
    rnaps(3,s1n/2+b1n+s2n+b2n+i)=s1(3,i+s1n/2)
end do
! Turn off comment for the next line only when open(file=22) for checking bases
8&22
! has its comments turned off
!   call writebase(rnaps)
!-----
! Assign mid-point coordinates to peptide residues.
! The residue points are the existing t1 coordinates minus the first
! two points.
!-----
    do j=1,2*t1n-1
        peppts(1,j)=t1(1,j+2)
        peppts(2,j)=t1(2,j+2)
        peppts(3,j)=t1(3,j+2)
    end do

!!!!!!!!!!!!!!!!!!!!Filter out best native-like, major groove configs!!!!!!!!!!!!!!!!!!!!
!   if (&
!       (peppts(1,1)<=4).and.&
!       (peppts(1,2)<4).and.&
!       (peppts(1,2)>0).and.&
!       (peppts(1,3)<4).and.&

```

```

! (peppts(1,3)>0).and.&
! (peppts(1,4)<4).and.&
! (peppts(1,4)>0).and.&
! (peppts(1,5)<4).and.&
! (peppts(1,5)>0).and.&
! (peppts(1,6)<4).and.&
! (peppts(1,6)>0).and.&
! (peppts(1,7)<4).and.&
! (peppts(1,7)>0).and.&
! (peppts(1,8)<4).and.&
! (peppts(1,8)>0).and.&
! (peppts(1,9)<4).and.&
! (peppts(1,9)>0).and.&
! (peppts(1,10)<4).and.&
! (peppts(1,10)>0).and.&
! (peppts(1,11)<4).and.&
! (peppts(1,11)>0).and.&
! (peppts(2,1)<4).and.&
! (peppts(2,2)<4).and.&
! (peppts(2,3)<4).and.&
! (peppts(2,4)<=4).and.&
! (peppts(2,5)>=4).and.&
! (peppts(2,6)>=4).and.&
! (peppts(2,7)>=4).and.&
! (peppts(2,8)<4).and.&
! (peppts(2,9)<4).and.&
! (peppts(2,10)<4).and.&
! (peppts(2,11)<4)&
! ) then
! major=major+1 !!!!!Count major configurations
! print*, '*****Print Configs*****'
! do j=1,pepcnt
!   peppts1(1,j)=peppts(1,j)
!   peppts1(2,j)=peppts(2,j)
!   peppts1(3,j)=peppts(3,j)
!   print*,j,'***x:',peppts1(1,j),&
!     '***y:',peppts1(2,j),'***z:',peppts1(3,j)
! end do
! call calc_en(peppts1,s1,s1n,s2,s2n,b1,b2,t1,b1n,b2n,t1n,histb1,&
! histb2,histt1,idx,rnacnt,pepcnt,rmanum,pepnum,potent,s1n2,s2n2,&
! b1n2,b2n2,t1n2,rnaps)
! end if
!! print *, 'Print major:',major
!! print *, '*****'

```

!!!!!!!!!!!!!! Turn on comments for this portion if the above comments are turned off!!!!

!!!!!!!!!!!!!! This part is for ALL configurations !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

do j=1,pepcnt
  peppts1(1,j)=peppts(1,j)
  peppts1(2,j)=peppts(2,j)
  peppts1(3,j)=peppts(3,j)
end do
call calc_en(peppts1,s1,s1n,s2,s2n,b1,b2,t1,b1n,b2n,t1n,histb1,&
histb2,histt1,idx,rnacnt,pepcnt,rnanum,pepnum,potent,s1n2,s2n2,&
b1n2,b2n2,t1n2,rnaps,prcnt)

```

!!

```

end subroutine savecfg
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

```

subroutine calc_en(peppts1,s1,s1n,s2,s2n,b1,b2,t1,b1n,b2n,t1n,histb1,&
histb2,histt1,idx,rnacnt,pepcnt,rnanum,pepnum,potent,s1n2,s2n2,&
b1n2,b2n2,t1n2,rnaps,prcnt)
integer s1,s1n,s2,s2n,t1,s1n2,s2n2,b1n2,b2n2,t1n2,b1,b2
integer b1n,b2n,t1n,histb1,histb2,histt1,idx,i,n,x,j,k
integer rnacnt,pepcnt,rnanum,pepnum
! integer i,j,k,l,m,n
integer r
integer*8 tmpsum
integer*8 prcnt
integer dist
real prsum
real potent
real dd,d
real r1,r2,r3,r4,dot
real alpha
dimension s1(3,s1n2),s2(3,s2n2)
dimension b1(3,b1n2),b2(3,b2n2),t1(3,t1n2)
dimension histb1(b1n),histb2(b2n),histt1(t1n)
dimension rnanum(rnacnt),pepnum(pepcnt)
dimension potent(8,20)
dimension r(3)
dimension tmpsum(pepcnt,rnacnt)
dimension prcnt(pepcnt,rnacnt,32)
dimension r1(3),r2(3),r3(3),r4(3)

```

```

real testcfg
dimension testcfg(11)
integer rnaps,peppts,peppts1
dimension rnaps(3,rnact),peppts1(3,pepcnt)
!-----
! Tabulate the pairs count in the big matrix. Also count the number of
! possible pairs per configuration.
!-----
      prsum=0
      do i=1,rnact
        do j=1,pepcnt
          tmpsum(j,i)=0
        end do
      end do
!-----
! Begin energy calculations
!-----
      do j=1,pepcnt
        colsum=0
        colcnt=0
        do i=1,rnact
!-----
! Identify major or minor groove using coarse planes.
!-----
          if (peppts1(3,j).lt.4) then
            if(peppts1(2,j).gt.2) then
              minor=0
            else
              minor=1
            end if
          else if ((peppts1(3,j).ge.4).and.(peppts1(3,j).le.10)) then
            if(peppts1(1,j).lt.4) then
              minor=0
            else
              minor=1
            end if
          else if (peppts1(3,j).gt.10) then
            if(peppts1(2,j).lt.4) then
              minor=0
            else
              minor=1
            end if
          end if
!-----

```

```

    r(1)=abs(rnaps(1,i)-peppts1(1,j))
    r(2)=abs(rnaps(2,i)-peppts1(2,j))
    r(3)=abs(rnaps(3,i)-peppts1(3,j))
    dist=r(1)+r(2)+r(3)
!-----
! Distance criteria.
!-----
! More stringent distance constraint.
!-----
    if ((dist.le.4).and.(dist.ge.2)) then
!-----
! Weights
!-----
        if (dist.eq.4) alpha=0.25
        if (dist.eq.3) alpha=0.50
        if (dist.eq.2) alpha=1.00
!-----
!-----
        prsum=prsum+potent(rnnum(i)+4*minor,pepnum(j))*alpha
        tmpsum(j,i)=1
        colcnt=colcnt+1
    end if
!-----
end do
!-----
!-----
end do
do i=1,b1n
testcfg(i)=histb1(i)
end do
do i=1,b2n
testcfg(i+3)=histb2(i)
end do
do i=1,t1n
testcfg(i+4)=histt1(i)
end do
testcfg(11)=prsum
!   write(17,'(20(10i5,f8.2))') (int(testcfg(i)),i=1,10),testcfg(11)
!   write(18,'(f7.2)') prsum

if (prsum.ge.30) then
    k=1
else if (prsum.ge.29) then
    k=2

```

```
else if (prsum.ge.28) then
  k=3
else if (prsum.ge.27) then
  k=4
else if (prsum.ge.26) then
  k=5
else if (prsum.ge.25) then
  k=6
else if (prsum.ge.24) then
  k=7
else if (prsum.ge.23) then
  k=8
else if (prsum.ge.22) then
  k=9
else if (prsum.ge.21) then
  k=10
else if (prsum.ge.20) then
  k=11
else if (prsum.ge.19) then
  k=12
else if (prsum.ge.18) then
  k=13
else if (prsum.ge.17) then
  k=14
else if (prsum.ge.16) then
  k=15
else if (prsum.ge.15) then
  k=16
else if (prsum.ge.14) then
  k=17
else if (prsum.ge.13) then
  k=18
else if (prsum.ge.12) then
  k=19
else if (prsum.ge.11) then
  k=20
else if (prsum.ge.10) then
  k=21
else if (prsum.ge.9) then
  k=22
else if (prsum.ge.8) then
  k=23
else if (prsum.ge.7) then
  k=24
```

```

else if (prsum.ge.6) then
  k=25
else if (prsum.ge.5) then
  k=26
else if (prsum.ge.4) then
  k=27
else if (prsum.ge.3) then
  k=28
else if (prsum.ge.2) then
  k=29
else if (prsum.ge.1) then
  k=30
else if (prsum.ge.0) then
  k=31
end if
do i=1,rnact
  do j=1,pepcnt
    prcnt(j,i,k)=prcnt(j,i,k)+tmpsum(j,i)
    prcnt(j,i,32)=prcnt(j,i,32)+tmpsum(j,i)
  end do
end do
end subroutine calc_en

```

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!! SUBROUTINE WRITEBASE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!! Save coordinates for base 8 and 22 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  subroutine writebase(rnaps)
    integer, dimension(3,28) :: rnaps
    write(22, 1000) (rnaps(i,8)*3.075,i=1,3),(rnaps(j,22)*3.075,j=1,3)
1000 FORMAT (6f8.3)
  end subroutine writebase

```

Program: *makepdb_unit2_universal.f95*

```

program makepdb
  implicit NONE
  character*50 configname
  integer input
  integer numrec
  integer :: count=0
  integer :: erno
  real dummy

100 print*, 'Enter the configuration record name'
  read(*,*) configname

```



```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    open(unit=1,file=configname,status='old',iostat=ername)
    if (ername==0) then
105 read(1,*,end=110) dummy
        count=count+1
        goto 105
110 print *, 'There are: ',count,' configurations'
        close(1)
    else
        print*, 'File not found. Enter the correct file name again.'
        goto 100
    end if

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
120 print *, 'How many configurations to make? '
    read(*,*) input
!!!!!!!!!! Set number of configurations to make !!!!!!!!!!!!!!!!!!!!!
    if (input==count) then
        numrec=count
    end if
    if (input>count) then
        print *, 'Input value exceeds number of recorded configurations. &
        Enter a new value.'
        goto 120
    end if
    if (input<count) then
        numrec=input
    end if

    print *, 'There are ',numrec,' configurations to be made.'
    call makefile(numrec,configname)
    end program makepdb

!!!!!!!!!!!!!!!!!!!!!! Call make pdbs based on number of configurations to make!!!!!!!!!!!!!!
    subroutine makefile(numrec,configname)
    character*50 rnaseq,pepseq
    character*50 pdbout
    character*50 configname
    character*50 tatseq
    integer sln,s2n,b1n,b2n,t1n
    integer sln2,s2n2,b1n2,b2n2,t1n2
    integer bcnt,acnt,rnanum,pepnum
    integer rnacnt,pepcnt
    integer jct

```

```

integer i,j,k
integer contact
integer latmov
character*3 rna3,pep3,bname3,aname3
character*1 rna,pep,bname,aname
dimension rna(28),rna3(28),pep(14),pep3(14)
dimension acnt(20), aname(20),aname3(20)
dimension bcnt(4),bname(4),bname3(4)
dimension rnanum(28),pepnum(14)
dimension latmov(3,18)
dimension contact(2,5)
dimension jct(4)
integer length,numrec
data bcnt/4*0/
data acnt/20*0/
data bname/'A','C','G','U'/
data aname/'A','C','D','E','F','G','H','I','K','L','M','N','P',&
'Q','R','S','T','V','W','Y'/
data bname3/'ADE','CYT','GUA','URA'/
data aname3/'ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS',&
'LEU','MET','ASN','PRO','GLN','ARG','SER','THR','VAL','TRP','TYR'/
s1n=12
s1n2=119
s2n=12
s2n2=119
b1n=3
b2n=1
t1n=6
b1n2=2*b1n+1
b2n2=2*b2n+1
t1n2=2*t1n+1
rnacnt=s1n+s2n+b1n+b2n
pepcnt=2*t1n-1

print*, '1. Initialization'
! Make file name
open(unit=1,file='names.dat',status='old')
read(1,*) pdbout
close(1)
! Make contact matrix
open(unit=1,file='contact5.dat',status='old')
read(1,*) contact
close(1)
! Make junction

```

```

    open (unit=1,file='jction.dat',status='old')
    read(1,*) jct
    close(1)
! Make 'move' matrix
    open(unit=1,file='idcub2.dat',status='old')
    read(1,*) latmov
    close(1)
! Make RNA and peptide sequences
    print*, 'Enter Tat sequence file'
    read(*,*) tatseq
    open(unit=1,file=tatseq,status='old')
    read(1,*) pepseq
    close(1)
    open(unit=1,file='bivtar.rna',status='old')
    read(1,*) rnaseq
    close(1)
    do i=1,rnacnt
        rna(i)=rnaseq(i:i)
        do j=1,4
            if (rna(i).eq.bname(j)) then
                rna3(i)=bname3(j)
                bcnt(j)=bcnt(j)+1
                rnanum(i)=j
            end if
        end do
    end do
    do j=1,pepcnt
        pep(j)=pepseq(j:j)
        do k=1,20
            if (pep(j).eq.aname(k)) then
                pep3(j)=aname3(k)
                acnt(k)=acnt(k)+1
                pepnum(j)=k
            end if
        end do
    end do
! Read reclen.dat
    open(unit=1,file='reclen.dat',status='old')
    read(1,*) length
    close(1)
! Read number of configs

    call gencfg(s1n,s2n,b1n,b2n,t1n,jct,contact,latmov,rnacnt,&
    pepcnt,rnanum,pepnum,s1n2,s2n2,b1n2,b2n2,t1n2,length,numrec,&

```

```

    pdbout,rna,pep,pep3,rna3,configname)
end subroutine makefile

!!!!!!!!!!!!!!!!!!!! Generate PBDs!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine gencfg(s1n,s2n,b1n,b2n,t1n,jct,contact,latmov,&
    rnacnt,pepcnt,rnanum,pepnum,s1n2,s2n2,b1n2,b2n2,t1n2,&
    length,numrec,pdbout,rna,pep,pep3,rna3,configname)
    character*50 configname
    character*50 pdbout
    character*1 rna,pep
    character pep3,rna3
    integer s1n,s2n,b1n,b2n,t1n
    integer s1n2,s2n2,b1n2,b2n2,t1n2
    integer bcnt,acnt,rnanum,pepnum
    integer rnacnt,pepcnt
    integer jct
    integer contact
    integer latmov
    dimension rnanum(28),pepnum(14)
    dimension rna(pepcnt),pep(pepcnt)
    dimension rna3(pepcnt),pep3(pepcnt)
    dimension latmov(3,18)
    dimension contact(2,5)
    dimension jct(4)
    integer length,numrec
!!!!!!!!!!!!!!!!!!!! Local Vars !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    integer s1,s2,t1,b1,b2
    integer histb1,histb2,histt1
    integer rnaps,peppts
    integer d,f,i,j,k,m,n
    real cfigs
    integer cfignum
    dimension s1(3,s1n2),s2(3,s1n2)
    dimension b1(3,b1n2),b2(3,b2n2)
    dimension t1(3,t1n2)
    dimension histb1(b1n),histb2(b2n),histt1(t1n)
    dimension cfigs(length,numrec)
    dimension rnaps(3,s1n+b1n+s2n+b2n)
    dimension peppts(3,2*t1n-1)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    print*, '2. Making PDBs'
    open(unit=1,file=configname,status='old')
    read(1,*) cfigs

```

```

        close(1)
!=====
=====
! Start of reconstruction
!c=====
=====
!c Re-read first stem data file.
!c-----
        open(unit=1,file='shelix8d_2.dat',status='old')
        read(1,*) s1
        close(1)
!c-----
!c First half of first stem.
!c-----
        do i=1,s1n/2
            rnaps(1,i)=s1(1,i)
            rnaps(2,i)=s1(2,i)
            rnaps(3,i)=s1(3,i)
        end do
!c-----
!c Second half of first stem.
!c-----
        do i=1,s1n/2
            rnaps(1,rnact-s1n/2+i)=s1(1,s1n/2+i)
            rnaps(2,rnact-s1n/2+i)=s1(2,s1n/2+i)
            rnaps(3,rnact-s1n/2+i)=s1(3,s1n/2+i)
        end do
!-----
        print*, '3. Getting into generation loop. Start outputting pdbs.'
        do cfnun=1,numrec
!           prsum=0
!-----
! This back calculates bulge 1.
!-----
            do i=1,b1n
                histb1(i)=cfnun(i,cfnun)
            end do
            b1(1,1)=s1(1,jct(1))
            b1(2,1)=s1(2,jct(1))
            b1(3,1)=s1(3,jct(1))
            do i=1,b1n
                b1(1,2*i+1)=b1(1,2*i-1)+latmov(1,histb1(i))
                b1(2,2*i+1)=b1(2,2*i-1)+latmov(2,histb1(i))
                b1(3,2*i+1)=b1(3,2*i-1)+latmov(3,histb1(i))
            end do
        end do
    end sub
end

```

```

end do
do i=1,b1n
  rnaps(1,s1n/2+i)=b1(1,2*i+1)
  rnaps(2,s1n/2+i)=b1(2,2*i+1)
  rnaps(3,s1n/2+i)=b1(3,2*i+1)
end do
!-----
! Second stem. Since this is for the fixed stem scenario.
! This is the same second stem generation as before.
!-----
! Make sure this transformation is the same as the earlier one.
!-----
do n=1,s2n
  s2(1,n)=-s1(1,n)+6
  s2(2,n)=-s1(2,n)+6
  s2(3,n)=s1(3,n)+8
end do
!-----
do i=1,s2n
  rnaps(1,s1n/2+b1n+i)=s2(1,i)
  rnaps(2,s1n/2+b1n+i)=s2(2,i)
  rnaps(3,s1n/2+b1n+i)=s2(3,i)
end do
!-----
! This generates bulge 2.
!-----
do i=1,b2n
  histb2(i)=cfgs(i+3,cfignum)
end do
b2(1,1)=s2(1,jct(3))
b2(2,1)=s2(2,jct(3))
b2(3,1)=s2(3,jct(3))
do i=1,b2n
  b2(1,2*i+1)=b2(1,2*i-1)+latmov(1,histb2(i))
  b2(2,2*i+1)=b2(2,2*i-1)+latmov(2,histb2(i))
  b2(3,2*i+1)=b2(3,2*i-1)+latmov(3,histb2(i))
end do
do i=1,b2n
  rnaps(1,s1n/2+b1n+s2n+i)=b2(1,2*i+1)
  rnaps(2,s1n/2+b1n+s2n+i)=b2(2,2*i+1)
  rnaps(3,s1n/2+b1n+s2n+i)=b2(3,2*i+1)
end do
!-----
! This generates tat chain.

```

```

!-----
do i=1,t1n
  histt1(i)=cfgs(i+4,cfignum)
end do
!-----
! First Contact (copied from tatmov)
!-----
! What's the first t1 point?
!-----
i=1
if (contact(1,i).eq.1) then
  t1(1,1)=s1(1,contact(2,i))
  t1(2,1)=s1(2,contact(2,i))
  t1(3,1)=s1(3,contact(2,i))
else if (contact(1,i).eq.2) then
  t1(1,1)=s2(1,contact(2,i))
  t1(2,1)=s2(2,contact(2,i))
  t1(3,1)=s2(3,contact(2,i))
else if (contact(1,i).eq.3) then
  t1(1,1)=b1(1,contact(2,i))
  t1(2,1)=b1(2,contact(2,i))
  t1(3,1)=b1(3,contact(2,i))
else if (contact(1,i).eq.4) then
  t1(1,1)=b2(1,contact(2,i))
  t1(2,1)=b2(2,contact(2,i))
  t1(3,1)=b2(3,contact(2,i))
end if
!-----
! Generate in-between points for the peptide.
!-----
do i=1,t1n
  t1(1,2*i+1)=t1(1,2*i-1)+latmov(1,histt1(i))
  t1(1,2*i)=t1(1,2*i-1)+latmov(1,histt1(i))/2
  t1(2,2*i+1)=t1(2,2*i-1)+latmov(2,histt1(i))
  t1(2,2*i)=t1(2,2*i-1)+latmov(2,histt1(i))/2
  t1(3,2*i+1)=t1(3,2*i-1)+latmov(3,histt1(i))
  t1(3,2*i)=t1(3,2*i-1)+latmov(3,histt1(i))/2
end do
!-----
peppts(1,1)=t1(1,3)
peppts(2,1)=t1(2,3)
peppts(3,1)=t1(3,3)
do j=4,2*t1n+1
  peppts(1,j-2)=t1(1,j)

```



```

open(unit=1,file='names.dat',status='old')
read(1,*) root
read(1,*) ext
period="."
close(1)
write(cfgtxt,'(i8.8)') cfgnum
outfil=root//cfgtxt//period//ext
open(unit=3,file=outfil,status='replace')
n=0

do p=1,rnacnt
n=n+1
write(3,1010) "ATOM",n,"O3'",n,(rnaps(m,p)*3.075,m=1,3)
end do
do p=1,2*t1n-1
n=n+1
write(3,1010) "ATOM",n,"CA",n,(peppts(m,p)*3.075,m=1,3)
end do
1010 FORMAT(A4,3X,I4,1X,A4,6X,I3,4X,3f8.3)
1100 FORMAT(A6,1X,I4,1X,I4)
close(3)
end subroutine writepdb

```

Program: *energycalc_unit2_universal.f95*

```

program makepdb
character*50 rnaseq,pepseq
character*50 pdbout
character*150 configname
character*150 energyname
integer s1n,s2n,b1n,b2n,t1n
integer s1n2,s2n2,b1n2,b2n2,t1n2
integer bcnt,acnt,rnanum,pepnum
integer rnacnt,pecnt
integer jct
integer i,j,k
integer contact
integer latmov
integer ername
character*3 rna3,pep3,bname3,aname3
character*1 rna,pep,bname,aname
dimension rna(28),rna3(28),pep(14),pep3(14)
dimension acnt(20), aname(20),aname3(20)
dimension bcnt(4),bname(4),bname3(4)
dimension rnanum(28),pepnum(14)

```

```

dimension latmov(3,18)
dimension contact(2,5)
dimension jct(4)
integer length,count,length1
integer errname
real dummy

data bcnt/4*0/
data acnt/20*0/
data bname/'A','C','G','U'/
data aname/'A','C','D','E','F','G','H','I','K','L','M','N','P',&
'Q','R','S','T','V','W','Y'/
data bname3/'ADE','CYT','GUA','URA'/
data aname3/'ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS',&
'LEU','MET','ASN','PRO','GLN','ARG','SER','THR','VAL','TRP','TYR'/
s1n=12
s1n2=119
s2n=12
s2n2=119
b1n=3
b2n=1
t1n=6
b1n2=2*b1n+1
b2n2=2*b2n+1
t1n2=2*t1n+1
rnacnt=s1n+s2n+b1n+b2n
pepcnt=2*t1n-1
! Make file name
open(unit=1,file='names.dat',status='old')
read(1,*) pdbout
close(1)
! Make contact matrix
open(unit=1,file='contact5.dat',status='old')
read(1,*) contact
close(1)
! Make junction
open (unit=1,file='jction.dat',status='old')
read(1,*) jct
close(1)
! Make 'move' matrix
open(unit=1,file='idcub2.dat',status='old')
read(1,*) latmov
close(1)
! Make RNA and peptide sequences

```

```

open(unit=1,file='bivtat_nat.pep',status='old')
read(1,*) pepseq
close(1)
open(unit=1,file='bivtar.rna',status='old')
read(1,*) rnaseq
close(1)
do i=1,rnactnt
  rna(i)=rnaseq(i:i)
  do j=1,4
    if (rna(i).eq.bname(j)) then
      rna3(i)=bname3(j)
      bcnt(j)=bcnt(j)+1
      rnanum(i)=j
    end if
  end do
end do
do j=1,pepcnt
  pep(j)=pepseq(j:j)
  do k=1,20
    if (pep(j).eq.aname(k)) then
      pep3(j)=aname3(k)
      acnt(k)=acnt(k)+1
      pepnum(j)=k
    end if
  end do
end do
! Read reclen.dat
open(unit=1,file='reclen.dat',status='old')
read(1,*) length1
close(1)
length=length1-1
! Read number of configs
count=0
100 print*, 'Enter the configuration record name'
read(*,*) configname

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  open(unit=1,file=configname,status='old',iostat=ername)
  if (ername==0) then
105 read(1,*,end=110) dummy
    count=count+1
    goto 105
110 print *, 'There are: ',count,' configurations'
    close(1)

```



```

real cfgs
integer cfignum
integer r
dimension r(3)
real r1,r2,r3,r4
dimension r1(3),r2(3),r3(3),r4(3)
dimension s1(3,s1n2),s2(3,s1n2)
dimension b1(3,b1n2),b2(3,b2n2)
dimension t1(3,t1n2)
dimension histb1(b1n),histb2(b2n),histt1(t1n)
dimension cfgs(length-1,count)
dimension rnaps(3,s1n+b1n+s2n+b2n)
dimension peppts(3,2*t1n-1)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  open(unit=1,file=configname,status='old')
  read(1,*) cfgs
  close(1)
  open(unit=1,file='potent4.dat',status='old')
  read(1,*) potent
  close(1)
!=====
=====
! Start of reconstruction
!c=====
=====
!c Re-read first stem data file.
!c-----
  open(unit=1,file='shelix8d_2.dat',status='old')
  read(1,*) s1
  close(1)
!c-----
!c First half of first stem.
!c-----
  do i=1,s1n/2
    rnaps(1,i)=s1(1,i)
    rnaps(2,i)=s1(2,i)
    rnaps(3,i)=s1(3,i)
  end do
!c-----
!c Second half of first stem.
!c-----
  do i=1,s1n/2
    rnaps(1,rnact-s1n/2+i)=s1(1,s1n/2+i)
    rnaps(2,rnact-s1n/2+i)=s1(2,s1n/2+i)

```

```

        rnaps(3,rnact-s1n/2+i)=s1(3,s1n/2+i)
    end do
!-----
    do cfnun=1,count
!       prsum=0
!-----
! This back calculates bulge 1.
!-----
        do i=1,b1n
            histb1(i)=cfgs(i,cfnun)
        end do
        b1(1,1)=s1(1,jct(1))
        b1(2,1)=s1(2,jct(1))
        b1(3,1)=s1(3,jct(1))
        do i=1,b1n
            b1(1,2*i+1)=b1(1,2*i-1)+latmov(1,histb1(i))
            b1(2,2*i+1)=b1(2,2*i-1)+latmov(2,histb1(i))
            b1(3,2*i+1)=b1(3,2*i-1)+latmov(3,histb1(i))
        end do
        do i=1,b1n
            rnaps(1,s1n/2+i)=b1(1,2*i+1)
            rnaps(2,s1n/2+i)=b1(2,2*i+1)
            rnaps(3,s1n/2+i)=b1(3,2*i+1)
        end do
!-----
! Second stem. Since this is for the fixed stem scenario.
! This is the same second stem generation as before.
!-----
! Make sure this transformation is the same as the earlier one.
!-----
        do n=1,s2n
            s2(1,n)=-s1(1,n)+6
            s2(2,n)=-s1(2,n)+6
            s2(3,n)=s1(3,n)+8
        end do
!-----
        do i=1,s2n
            rnaps(1,s1n/2+b1n+i)=s2(1,i)
            rnaps(2,s1n/2+b1n+i)=s2(2,i)
            rnaps(3,s1n/2+b1n+i)=s2(3,i)
        end do
!-----
! This generates bulge 2.
!-----

```

```

do i=1,b2n
  histb2(i)=cfgs(i+3,cfgnum)
end do
b2(1,1)=s2(1,jct(3))
b2(2,1)=s2(2,jct(3))
b2(3,1)=s2(3,jct(3))
do i=1,b2n
  b2(1,2*i+1)=b2(1,2*i-1)+latmov(1,histb2(i))
  b2(2,2*i+1)=b2(2,2*i-1)+latmov(2,histb2(i))
  b2(3,2*i+1)=b2(3,2*i-1)+latmov(3,histb2(i))
end do
do i=1,b2n
  rnaps(1,s1n/2+b1n+s2n+i)=b2(1,2*i+1)
  rnaps(2,s1n/2+b1n+s2n+i)=b2(2,2*i+1)
  rnaps(3,s1n/2+b1n+s2n+i)=b2(3,2*i+1)
end do
!-----
! This generates tat chain.
!-----
do i=1,t1n
  histt1(i)=cfgs(i+4,cfgnum)
end do
!-----
! First Contact (copied from tatmov)
!-----
! What's the first t1 point?
!-----
i=1
if (contact(1,i).eq.1) then
  t1(1,1)=s1(1,contact(2,i))
  t1(2,1)=s1(2,contact(2,i))
  t1(3,1)=s1(3,contact(2,i))
else if (contact(1,i).eq.2) then
  t1(1,1)=s2(1,contact(2,i))
  t1(2,1)=s2(2,contact(2,i))
  t1(3,1)=s2(3,contact(2,i))
else if (contact(1,i).eq.3) then
  t1(1,1)=b1(1,contact(2,i))
  t1(2,1)=b1(2,contact(2,i))
  t1(3,1)=b1(3,contact(2,i))
else if (contact(1,i).eq.4) then
  t1(1,1)=b2(1,contact(2,i))
  t1(2,1)=b2(2,contact(2,i))
  t1(3,1)=b2(3,contact(2,i))

```

```

end if
!-----
! Generate in-between points for the peptide.
!-----
do i=1,t1n
  t1(1,2*i+1)=t1(1,2*i-1)+latmov(1,histt1(i))
  t1(1,2*i)=t1(1,2*i-1)+latmov(1,histt1(i))/2
  t1(2,2*i+1)=t1(2,2*i-1)+latmov(2,histt1(i))
  t1(2,2*i)=t1(2,2*i-1)+latmov(2,histt1(i))/2
  t1(3,2*i+1)=t1(3,2*i-1)+latmov(3,histt1(i))
  t1(3,2*i)=t1(3,2*i-1)+latmov(3,histt1(i))/2
end do
!-----
peppts(1,1)=t1(1,3)
peppts(2,1)=t1(2,3)
peppts(3,1)=t1(3,3)
do j=4,2*t1n+1
  peppts(1,j-2)=t1(1,j)
  peppts(2,j-2)=t1(2,j)
  peppts(3,j-2)=t1(3,j)
end do
prsum=0
!-----
! Begin energy calculations
!-----
do j=1,pepcnt
  do i=1,rnacnt
!-----
! Identify major or minor groove using coarse planes.
!-----
    if (peppts(3,j).lt.4) then
      if(peppts(2,j).gt.2) then
        minor=0
      else
        minor=1
      end if
    else if ((peppts(3,j).ge.4).and.(peppts(3,j).le.10)) then
      if(peppts(1,j).lt.4) then
        minor=0
      else
        minor=1
      end if
    else if (peppts(3,j).gt.10) then
      if(peppts(2,j).lt.4) then

```



```

        minor=0
    else
        minor=1
    end if
end if

!-----
r(1)=abs(rnaps(1,i)-peppts(1,j))
r(2)=abs(rnaps(2,i)-peppts(2,j))
r(3)=abs(rnaps(3,i)-peppts(3,j))
dist=r(1)+r(2)+r(3)
!-----
! Distance criteria.
!-----
! More stringent distance constraint.
!-----
    if ((dist.le.4).and.(dist.ge.2)) then
!-----
! Weights
!-----
        if (dist.eq.4) alpha=0.25
        if (dist.eq.3) alpha=0.50
        if (dist.eq.2) alpha=1.00
!-----
!-----
!      print *,pepnum(j)
      prsum=prsum+potent(rnanum(i)+4*minor,pepnum(j))*alpha
    end if
!-----
    end do
!-----
!-----
    end do
    call writeprsum(prsum)
end do
end subroutine gencfg

!!!!!!!!!!!!!!!!!!!! SUBROUTINE WRITEPRSUM !!!!!!!!!!!!!!!!!!!!!
subroutine writeprsum(prsum)
real prsum
write(11, '(f7.2)') prsum
end subroutine writeprsum

```

Appendix C Python Program Listings

Program: *filter_flex.py*

```
#####
#   Check configs.txt file to filter
#   out specific moves for peptide
#####
#           Possible moves:
#           1 < move < 6: edge moves
#           move > 6: diagonal moves
#   This is based on configurations file
#   which has 11 columns. In python, 11
#   columns as put in an array with indices
#   go from 0 to 10. For indices:
#           0 -> 2: Bulge 1 moves
#           3: Bulge 2 moves
#           4: 1st nuc-peptide segment GUA14-R70
#           5: 2nd segment R70-T72
#           6: 3rd segment T72-G74
#           7: 4th segment G74-G76
#           8: 5th segment G76-R78
#           9: 6th segment R78-R80
#           10: Energy
#   Change conditions for peptide from
#   lines 93 -> 97 (flanked by '#')
#   This final version can have user input
#   conditions when running the program.
#   User can input name for output file
#   for each case after changing filter
#   output file for 1 run has _flex in its
#   name to tell users this is an output
#   file of a previous run.
#   Output file has '_flex' in its name.
#####
import os
newconf=0
count=0
# cond = []
flex = '_flex'
end = '.txt'
def print_stars():
    print '*'*80
def print_dash():
    print '-'*80
```

```

def checked(userin):
    if userin == 'e' or userin == 'E':
        temp = [i for i in range(1,7)]
    else:
        temp = [i for i in range(7,19)]
    return temp

path = os.getcwd()
dir = os.listdir(path)
print_stars()
print 'Here are all the possible Configurations and Energy files:'
print_dash()
for file in dir:
    if file.endswith('.txt'):
        print '\t',file
print_dash()

file_found = False
file_same = False
while not file_found:
    print_stars()
    configs_f = raw_input\
    ("Please enter the Configurations file name or 'break' to stop: \n\t")
    caplet = configs_f.upper()
    if os.path.isfile(configs_f):
        file_found=True
        if (os.stat(configs_f).st_size > 0):
            print_dash()
            while not file_same:
                outf = raw_input\
                ("Enter output file name (no file extension) for this
specific run:\n\t")
                if (outf != configs_f) or (os.path.isfile(outf)==False):
                    file_same = True
                    outf=outf+flex+end
                    outwrite = open(outf,'w')
                    input = open(configs_f,'r')
                    print_dash()
                    print 'Start entering conditions for Tat
segments.'

                    print 'Segment 1 (GUA15-R70) is always edge.'
                    print 'Enter E/e for Edge OR D/d for Diagonal.'
                    ts2 = raw_input('\tR70-T72: ')
                    ts3 = raw_input('\tT72-G74: ')

```

```

        ts4 = raw_input('\tG74-G76: ')
        ts5 = raw_input('\tG76-R78: ')
        ts6 = raw_input('\tR78-R80: ')
        seg2 = checked(ts2)
        seg3 = checked(ts3)
        seg4 = checked(ts4)
        seg5 = checked(ts5)
        seg6 = checked(ts6)
        for line in input:
            count+=1
            splits=line.split()
#####
#####
                                if(int(splits[5]) in seg2)and\
                                (int(splits[6]) in seg3)and\
                                (int(splits[7]) in seg4)and\
                                (int(splits[8])in seg5)and\
                                (int(splits[9]) in seg6):
#####
#####
                                outwrite.write(line)
                                newconf+=1
        else:
            print_dash()
            print 'You have entered a name for an existing
file.'
            print_dash()
    print_stars()
    if (newconf > 0) and (newconf < count):
        percent = newconf*100/float(count)
        print 'There are ',count,' previous configurations.'
        print 'There are ',newconf,' new configurations.'
        print 'Ratio (%):', '{:.3f}'.format(percent)
        print 'Output file is: <', outf,>'
        print_stars()
    elif newconf == 0:
        print 'No special configurations were found.'
        print_stars()
    elif newconf == count:
        print '# new configs is equal to # configs in input file.'
        print 'You have input this file: ',configs_f
        print 'Check input file name and restart the program.'
        print_stars()
    else:

```

```

        print_stars()
        print 'Input file is empty.'
        print_stars()
        break
    elif (caplet=='BREAK'):
        print_stars()
        print 'Nothing is done.'
        print_stars()
        break
    else:
        print_stars()
        print 'File not found.'
        print 'Here are all the possible files again:'
        print_dash()
        for file in dir:
            if file.endswith('.txt'):
                print '\t',file
        print_dash()

```

Program: *pdb_2points_distance.py*

```

import sys
import csv
import math
import os, errno
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np

beg = 'biv'
end = '.pdb'
end1 = '.csv'
fileList=[]
dixt=[]
pbdigit = 0
unified = open('coordsys_test.txt', 'w') #test output file
unified2 = open('biv_number_check.txt', 'w')
unified3 = open('distance_list.txt', 'w') #output list of distances between 8 and 22
unified4 = open('distance_list_biv.txt', 'w') #output list of 8-22 distances with pdb
names
#dixtwrite = open('dist_distribution.txt', 'w') #output distance distribution

#####
#Adding defined number of zeros to pdb file name

```

```

def add_zero(int, cnt):
    zeros = str(int)
    for i in range(cnt - len(str(int))):
        zeros = '0' + zeros
    return zeros
#####
def print_stars():
    print '*'*80
#####
def userInput():
    number = False
    number1 = False
    global pdbdigit
    while not number:
        usrchoice = raw_input("Number of pdb files to be processed: ")
        try:
            numfile = int(usrchoice)
            number = True
            while not (number1):
                usrdigit = raw_input("How many digits in pdb file
name (by looking at .pdb output files): ")
                try:
                    filedigit = int(usrdigit)
                    pdbdigit = filedigit
                    number1 = True
                    for u in range(1, numfile+1):
                        name = beg + add_zero(u, filedigit)+end
                        fileList.append(name)
                except: ("Please enter number only!")
            except ValueError:
                print("Please enter number only!")
#####
#####
#Calculate distance between 2 atoms
def distance(filename, atomnr1, atomnr2):
    pdbfile = open(filename, 'r').readlines()

    for line in pdbfile:
        try:
            if int(line.split()[1]) == int(atomnr1):
                x1 = float(line.split()[4])
                y1 = float(line.split()[5])
                z1 = float(line.split()[6])
            elif int(line.split()[1]) == int(atomnr2):

```

```

        x2 = float(line.split()[4])
        y2 = float(line.split()[5])
        z2 = float(line.split()[6])

    except:
        continue
    distance = math.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)
    return distance
#####
#####
file_found = False
while not file_found:
    print_stars()
    userinput()
#####
    if os.path.isfile(fileList[0]):
        file_found = True
        print_stars()
        print 'PDB files processed: ' + str(len(fileList)) + ' files.'
#        for print_j in fileList:
#            print print_j
        print_stars()
#####
        print 'Enter 2 O3' to calculate distances.'
        o3_1 = raw_input("First O3': ")
        o3_2 = raw_input("Second O3': ")
        o3_1_num = int(o3_1)
        o3_2_num = int(o3_2)
        print_stars()
        print 'Now calculate and output distances for ' + str(len(fileList)) + '
files.'
#####
        for j in fileList:
            current = open(j)
            unified.write(j+'\n')
            unified2.write(j+'\n')
            for k in current:
                splits = k.split()
                if (splits[0] == 'ATOM') and (splits[1] == '8'):
                    unified.write(splits[4]+'\\t')
                    unified.write(splits[5]+'\\t')
                    unified.write(splits[6]+'\\n')
                elif (splits[0] == 'ATOM') and (splits[1] == '22'):
                    unified.write(splits[4]+'\\t')
                    unified.write(splits[5]+'\\t')

```

```

unified.write(splits[6]+'\\n\\n')
#####

#Output list of distances between 8 and 22
r8_22 = distance(j, o3_1_num, o3_2_num)
distx.append(str.format("{0:.3f}",r8_22))
unified3.write(str.format("{0:.3f}",r8_22) + '\\t')
unified4.write(j + '\\t' + str.format("{0:.3f}",r8_22) + '\\n')
#print str.format("{0:.3f}",r8_22)
print_stars()
print 'Calculate distances for ' + str(len(fileList)) + ' files Finished!'
print_stars()
#####
#
dist_str = []
list_qq = []
##### Check all distances and calculate frequencies #####
linex = Counter()
linex.update(distx)
pdbdist = len(distx)
discnt = linex.values() #store frequencies of distances
#####
##
dist_f = [float(i) for i in discnt]
dist_q = [100*x/pdbdist for x in dist_f] #calculate percentage of each
distance
for x in dist_q:
    dist_str.append(str.format("{0:.1f}",x))
#####
###
test_q = linex.items() #store distance and resp. freq
list_d, list_f = zip(*test_q)
list_qq = zip(list_d, list_f, dist_str)
disthis = [float(i) for i in distx]
#####
#####
print 'Frequencies of calculated distances.'
print 'Distance' + '\\tFrequency' + '\\tPercent'
#distwrite.write('Distance'+ '\\tFrequency'+ '\\tPercent\\n')

for x,y in zip(linex, dist_str):
    print '%s\\t\\t%d\\t\\t%s' % (x, linex[x], y)
    #distwrite.write('%s\\t\\t%d\\t\\t%s\\n' % (x, linex[x], y))
print_stars()

```



```

#####
####
##### Modify the path to save files to different folder #####
#         cloud = '....'
#         print 'Online/Different save folder: ' + cloud
#         path=os.getcwd()
#         print 'Current folder :',path
#         print_stars()
#         out_name = raw_input("Enter Excel file name w/o extension: ")
#         out_csv = out_name + end1
#         out_cloud = out_name + end1
#         cloud_save = os.path.join(cloud, out_cloud)
#         print 'Now output excel distriution file ' + out_csv + ' in same folder.'
##### Save to local folder same as pdb folder #####

        with open(out_csv, 'wb') as csvout:
            csv_out = csv.writer(csvout)
            csv_out.writerow(['Distance','Frequency','Percent'])
            for row in list_qq:
                csv_out.writerow(row)
##### Save to online storage or other folder #####
#         with open(cloud_save, 'wb') as csvout1:
#             csv_out1 = csv.writer(csvout1)
#             csv_out1.writerow(['Distance','Frequency','Percent'])
#             for row1 in list_qq:
#                 csv_out1.writerow(row1)
#         print_stars()
##### Plot histogram
#####
#         cloud_fig = os.path.join(path, out_name+'.png')
#         print 'Now start to plot histogram.'
#         plt.hist(disthis,10)
#         plt.xlabel('Distance', fontsize=20)
#         plt.ylabel('Frequency', fontsize=20)
#         plt.xticks(disthis, rotation=5, fontsize=10)
#         plt.savefig(cloud_fig, figsize = (4, 3), dpi = 300, bbox_inches='tight')
#         plt.show()
#         print 'Done.'
#         print_stars()
#####
####

    else:
        print_stars()
        print 'No pdb files with ' + str(pdbdigit) + ' digits in names found!'

```

```

        print 'Restart process with a different number of digits matching those
pdb file names.'
        print_stars()
        fileList = []

```

Program: *checkbase.py*

```

import sys
import csv
import math
import os, errno
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np

count8=0
count22=0
count=0
distx=[]
dist_str,list_qq = [],[]
end1='.csv'

def print_stars():
    print '*'*85

distlist = open('distlist.txt','w')
with open('base8_22.txt','r') as base8:
    for line in base8:
        x8 = float(line.split()[0])
        y8 = float(line.split()[1])
        z8 = float(line.split()[2])
        x22 = float(line.split()[3])
        y22 = float(line.split()[4])
        z22 = float(line.split()[5])
        distance = math.sqrt((x8-x22)**2 + (y8-y22)**2 + (z8-z22)**2)
        distx.append(str.format("{0:.3f}",distance))
        distlist.write(str.format("{0:.3f}",distance)+'\n')
base8.close()

linex = Counter()
linex.update(distx)
pdbdist = len(distx)
discent = linex.values()

dist_f = [float(i) for i in discent]

```

```

dist_q = [100*x/pdbdist for x in dist_f]
for x in dist_q:
    dist_str.append(str.format("{0:.1f}",x))
#####
###
test_q = linex.items() #store distance and resp. freq
list_d, list_f = zip(*test_q)
list_qq = zip(list_d, list_f, dist_str)
disthis = [float(i) for i in distx]
#####
#####
print_stars()
print 'There are ', pdbdist, ' distances recorded.'
print 'Frequencies of calculated distances.'
print 'Distance' + '\tFrequency' + '\tPercent'
#distwrite.write('Distance'+'\tFrequency'+'\tPercent\n')
for x,y in zip(linex, dist_str):
    print '%s\t\t%d\t\t%s' % (x, linex[x], y)
    #distwrite.write('%s\t\t%d\t\t%s\n' % (x, linex[x], y))
print_stars()
#####
#####
##### Modify the path to save files to different folder #####
# cloud = '/Users/locnguyen/Box Sync/Distance distribution New'
# print 'Online/Different save folder: ' + cloud
print 'Current folder :'+ os.getcwd()
print_stars()
out_name = raw_input("Enter Excel file name: ")
out_csv = out_name + end1
# out_cloud = out_name + end1
# cloud_save = os.path.join(cloud, out_cloud)
print 'Now output excel distriution file ' + out_csv + ' in same folder.'
##### Save to local folder same as pdb folder #####

with open(out_csv, 'wb') as csvout:
    csv_out = csv.writer(csvout)
    csv_out.writerow(['Distance','Frequency','Percent'])
    for row in list_qq:
        csv_out.writerow(row)

print_stars()
print 'Finished!'

```

Program: *count_configs.py*

import os

```

count = 0
def print_stars():
    print '*'*80
def print_dash():
    print '-'*80
path = os.getcwd()
dir = os.listdir(path)
print_stars()
print 'Here are all the possible Configurations and Energy files:'
print_dash()
for file in dir:
    if file.endswith('.txt'):
        print '\t',file
print_dash()
print_stars()
file_found = False
while not file_found:
    energy_f = raw_input('Please enter the energy OR configurations file name:
\n\t')
    if os.path.isfile(energy_f):
        file_found=True
        with open(energy_f,'r') as f:
            for line in f:
                count+=1
    else:
        print_stars()
        print 'File not found.'
        print 'Here are all the possible files again:'
        print_dash()
        for file in dir:
            if file.endswith('.txt'):
                print '\t',file
        print_dash()
print_stars()
print 'Opened file is: ', energy_f
print 'There are ',count,' configurations.'
print_stars()

```

Program: *change_extension.py*

```

import os

def printstars():
    print '*'*100
end = '.engy'

```

```

path = os.getcwd()
files = []
dirs = os.listdir(path)

printstars()
print 'Change file extension'
printstars()
print 'Here are the old files'
for i in dirs:
    if i.startswith('energy_out') and i.endswith('.txt'):
        print i
        (root,ext)=os.path.splitext(i)
        os.rename(i, root+end)
        files.append(root+end)

printstars()
print 'Here are the new files'
for i in range(len(files)):
    print files[i]
printstars()

```

Program: *hist_energy_summary.py*

```

import sys
import os
import csv
# import numpy
from collections import Counter

def print_stars():
    print '*'*90
def print_dash():
    print '-'*90
#####
count = 0
sumfreq, sumen = 0,0
enfiles = []
engmed = []
freqmode = []
# outname = 'average_hist_summary.txt'
# outmed = 'median_hist.txt'
allcol = 'all_hist_energy_summary.txt'
# encoll = open(outname,'w')
# enmed = open(outmed, 'w')
allcoll = open(allcol, 'w')
#####

```

```

print_stars()
path = os.getcwd()
print 'Current folder: ', path
dirs = os.listdir(path)
print_stars()
print 'Here are all energy histogram files: '
print_dash()
for file in dirs:
    if file.endswith('.hist'):
        print file
        enfiles.append(file)

print_dash()
print_stars()
#####
max_width = max(len(filename) for filename in enfiles)
# encoll.write('Files'.ljust(max_width+5) + 'Average Frequency'.ljust(max_width) + \
# 'Mean Average Hist Energy' + '\n')
# enmed.write('Files'.ljust(max_width+5) + 'Median Energy' + '\n')

#####
allcoll.write('*'*90+'\n')
allcoll.write('This file contains Total Configurations, Mean Hist Energy'+\
', and Median Energy.'+'\n')
allcoll.write('It also contains the mode for energy of the maximum number'+\
' of configurations.\n')
allcoll.write('*'*90+'\n')
allcoll.write('Files' + '\n')
#
for i in range(len(enfiles)):
    #
    allcoll.write('{:3d}'.format(i+1) + ') ' + enfiles[i] + '\n')    #
    if i < (len(enfiles) - 1):
        #
        i = i + 1
#
else:
#
i = 0
#

allcoll.write('-'*90+'\n')
#
allcoll.write('Files'.ljust(max_width+5) + 'Total Configs'.ljust(max_width) + \
'Mean Hist Energy'.ljust(max_width) + 'Median Energy'.ljust(max_width) + \
'Mode\n')#

```

```
#####

#####
for j in range(len(enfiles)):
    print 'Processing ', enfiles[j]
    current = open(enfiles[j])
    for line in current:
        engy = (-1)*float(line.split()[0])
        freq = int(line.split()[1])
        freqmode.append(freq)
        maxfreq = max(freqmode)
        if freq!=0:
            sumfreq+=freq
            histen = engy*freq
            sumen += histen
            enavg = sumen/float(sumfreq)
            engmed.append(engy)
        # Get the mode for energy based on the maximum # configs
        if freq == maxfreq:
            enmode = engy
    # Check and set median value #
    lst = sorted(engmed)
    if len(lst) < 1:
        median = 0
    if (len(lst) %2) == 1:
        median = lst[((len(lst)+1)/2)-1]
    else:
        median = float(sum(lst[(len(lst)/2)-1:(len(lst)/2)+1]))/2.0
#####
#     encoll.write(enfiles[j].ljust(max_width+6) +
# '{:14.2f}'.format(sumfreq).ljust(max_width+3) + \
#     '{:4.2f}'.format(enavg) + '\n')
#
#     enmed.write(enfiles[j].ljust(max_width+6) + '{:4.2f}'.format(median) + '\n')
#####
#
#     allcoll.write('{:3d}'.format(j+1)+''.ljust(6)+'{:>8d}'.format(sumfreq).ljust(20)
+ \
#     '{:4.2f}'.format(enavg).ljust(23)+'{:4.2f}'.format(median).ljust(19)+\
#     '{:4.2f}'.format(enmode)+'\n')
#     allcoll.write(enfiles[j].ljust(max_width+5)+'{:>8d}'.format(sumfreq).ljust(max_
x_width)+\
#     '{:4.2f}'.format(enavg).ljust(max_width)+'{:4.2f}'.format(median).ljust(max_
width)+\

```

```

        '{:4.2f}'.format(enmode)+'\n')
#####
# Clear array and reset temp values to 0
#
    sumfreq = 0
    sumen = 0
    engmed[:] = []
    freqmode[:] = []
print_stars()
#####
print 'Output file is: ', allcol
print_stars()

```

Program: *raw_energy_summary.py*

```

import sys
import os
import csv
import numpy
from collections import Counter

def mode(list):
    d={}
    for elm in list:
        try:
            d[elm]+=1
        except(KeyError):
            d[elm]=1
    keys=d.keys()
    max = d[keys[0]]

    for key in keys[1:]:
        if d[key]>max:
            max=d[key]
    max_k = []
    for key in keys:
        if d[key]==max:
            mode = key
    return mode

def print_stars():
    print '*'*90
def print_dash():
    print '-'*90
#####

```



```

count = 0
sumfreq, sumen = 0,0
enfiles = []
engmed = []
allcol = 'all_raw_energy_summary.txt'
allcoll = open(allcol, 'w')
#####
print_stars()
path = os.getcwd()
print 'Current folder: ', path
dirs = os.listdir(path)
print_stars()
print 'Here are all energy files: '
print_dash()
for file in dirs:
    if file.endswith('.engy') or file.endswith('.energy'):
        print file
        enfiles.append(file)

print_dash()
print_stars()
# print_stars()
# path = os.getcwd()
# print 'Current folder: ', path
# dirs = os.listdir(path)
# print_stars()
# print 'Here are all energy histogram files: '
# print_dash()
# for file in dirs:
#     if file.endswith('.hist'):
#         print file
#         enfiles.append(file)
# print_dash()
# print_stars()
#####
max_width = max(len(filename) for filename in enfiles)
#####
allcoll.write('*'*90+'\n')
allcoll.write('This file contains Total Configurations, Mean Hist Energy'+\
', and Median Energy.'+'\n')
allcoll.write('It also contains the mode for energy of the maximum number'+\
' of configurations.\n')
allcoll.write('There are '+'{:3d}'.format(len(enfiles))+ ' in total.\n')
allcoll.write('*'*90+'\n')

```

```

allcoll.write('Files' + '\n')
#
for i in range(len(enfiles)):
#
    allcoll.write('{:3d}'.format(i+1) + ') ' + enfiles[i] + '\n')    #
    if i < (len(enfiles) - 1):
#
        i = i + 1
#
    else:
#
        i = 0
#
allcoll.write('-*90+\n')
#
allcoll.write('Files' + '\t' + 'Total Configs' + '\t' + 'Mean Hist Energy' + \
'\t' + 'Median Energy\t\t' + 'Mode and Frequency\n')
#
#####

#####

for j in range(len(enfiles)):
    print 'Processing ', enfiles[j]
    current = open(enfiles[j])
    for line in current:
        engy = (-1)*float(line.split()[0])
        sumen += engy
        count += 1
        enavg = sumen/float(count)
        engmed.append(engy)
#    enmode = mode(engmed)
    engmed1 = engmed
    data = Counter(engmed1)
    mode1 = data.most_common(1)
    lst = sorted(engmed)
    if len(lst) < 1:
        median = 0
    if (len(lst) %2) == 1:
        median = lst[((len(lst)+1)/2)-1]
    else:
        median = float(sum(lst[(len(lst)/2)-1:(len(lst)/2)+1]))/2.0
#    allcoll.write('{:3d}'.format(j+1)+'').ljust(6)+'{:>8d}'.format(count).ljust(20)+'\
#    '{:4.2f}'.format(enavg).ljust(23)+'{:4.2f}'.format(median).ljust(19)+'\
#    '{:4.2f}'.format(enmode)+'\n')

```

```

allcoll.write('{:3d}'.format(j+1)+'').ljust(6)+'{:>8d}'.format(count).ljust(20)+'\
'{:4.2f}'.format(enavg).ljust(23)+'{:4.2f}'.format(median).ljust(19)+'\
''.join('{:2f}: {:>6d}'.format(k,v) for k,v in mode1) +'\n')

#####
# Clear array and reset temp values to 0
#
# print mode1
# count = 0
# sumen = 0
# engmed[:] = []
print_stars()
#####
print 'Output file is: ', allcol
print_stars()

```

Program: *pro.py*

```

#input1 = open('biv00001.pdb')
fileList = ['biv00001.pdb', 'biv00002.pdb', 'biv00003.pdb', 'biv00004.pdb',
            'biv00005.pdb', 'biv00006.pdb', 'biv00007.pdb', 'biv00008.pdb',
            'biv00009.pdb']
beg = 'biv000'
end = '.pdb'
for i in range(10, 100):
    name = beg + str(i) + end
    fileList.append(name)
fileList.append('biv00100.pdb')

unified = open('unified.txt', 'w')
#unified.write("test")

for j in fileList:
    current = open(j)
    unified.write(j+'\n')
    for k in current:
        splits = k.split()
        print(splits)
        if (splits[0] == 'ATOM') and (splits[1] == '7'):unified.write(k+'\n')
        elif (splits[0] == 'ATOM') and (splits[1] == '8'): unified.write(k+'\n')
        elif (splits[0] == 'ATOM') and (splits[1] == '9'): unified.write(k+'\n')
        elif (splits[0] == 'ATOM') and (splits[1] == '22'): unified.write(k+'\n')

```

Program: *unique.py*

```

def add( filename ):

```

```

openFile = open(filename)

allArr = []
local = []
filtered = []
temp = ""
out = []
for i in openFile:
    #i.strip()
    if ( i is not ""): allArr.append(i.strip())
for j in allArr:
    if j != "":
        filtered.append(j)
print(len(filtered))
for k in range(0, 500, 5):
    current = filtered[k: k+5]
    noName = current[1:5]
    if(noName not in allArr):
        allArr.append(noName)
        out.append(current)

    prettyPrint( out)
def checkUnique(large, small):
    if small in large:
        return True
    return False

def prettyPrint( arr):
    output = open('unique.txt', 'w')
    for i in arr:
        for j in i:
            output.write(j+'\n')
        output.write('\n')
    output.close()
add('unified.txt')

```