Fall 2016

RNA-Protein Structure Classifiers Incorporated into Second-Generation Statistical Potentials

Takayuki Kimura
San Jose State University

RNA-PROTEIN STRUCTURE CLASSIFIERS INCORPORATED INTO SECOND-GENERATION STATISTICAL POTENTIALS

A Thesis

Presented to

The Faculty of the Department of Chemistry

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Takayuki Kimura

December 2016

The Designated Thesis Committee Approves the Thesis Titled

RNA-PROTEIN STRUCTURE CLASSIFIERS INCORPORATED INTO SECOND-GENERATION STATISTICAL POTENTIALS

by

Takayuki Kimura

APPROVED FOR THE DEPARTMENT OF CHEMISTRY

SAN JOSÉ STATE UNIVERSITY

December 2016

Dr. Brooke Lustig	Department of Chemistry
Dr. Daryl K. Eggers	Department of Chemistry
Dr. Alberto A. Rascón, Jr.	Department of Chemistry

ABSTRACT

RNA-PROTEIN STRUCTURE CLASSIFIERS INCORPORATED INTO SECOND-GENERATION STATISTICAL POTENTIALS

by Takayuki Kimura

Computational modeling of RNA-protein interactions remains an important endeavor. However, exclusively all-atom approaches that model RNA-protein interactions via molecular dynamics are often problematic in their application. One possible alternative is the implementation of hierarchical approaches, first efficiently exploring configurational space with a coarse-grained representation of the RNA and protein. Subsequently, the lowest energy set of such coarse-grained models can be used as scaffolds for all-atom placements, a standard method in modeling protein 3D-structure. However, the coarse-grained modeling likely will require improved ribonucleotide-amino acid potentials as applied to coarse-grained structures. As a first step we downloaded 1,345 PDB files and clustered them with PISCES to obtain a non-redundant complex data set. The contacts were divided into nine types with DSSR according to the 3D structure of RNA and then 9 sets of potentials were calculated. The potentials were applied to score fifty thousand poses generated by FTDock for twenty-one standard RNA-protein complexes. The results compare favorably to existing RNA-protein potentials. Future research will optimize and test such combined potentials.

ACKNOWLEDGEMENTS

This research project would not have been possible without the support of many people. First, I wish to express my gratitude to my research advisor, Dr. Lustig who guided me and taught me the tremendous joy of pursuing research and excitement in solving problems. He also showed me that scientific research requires a lot of energy and time to achieve. By accomplising this research, I acquired various skills including python coding, R, statistical analysis, and scientific writing. I would also like to thank my committee members, Dr. Eggers and Dr. Rascón, for spending their precious time to answering my questions and reviewing my thesis. I would also like to thank Phuc Tran, a former undergrad student who helped me to proceed with my project. I would also like to thank current grad students at Dr. Lustig's lab, Thanh Le and Artem Soshnikov for providing me useful advice. Dr. Sami Khuri, the chair of Computer Science Department also mentored me since before I was accepted to this university. I would also like to thank him and his wife, Dr. Natalia Khuri for their practical advice. For my family, I appreciate my beloved wife who financially and mentally supported me for, at least, these two and half years in San Jose State University. To others who supported me, I thank you so much.

Takayuki Kimura

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Interactions between RNA and Protein Are Important

Protein and RNA are the two critical macromolecular classes in biology as evidenced by the central dogma. The interaction between RNA and protein is essential for many regulatory processes in cells, especially in post-transcriptional regulation. Many processes in development and differentiation are related to this RNA-protein interaction.[1] Areas of interest include interactions involving the ribosome and spliceosome, but characterization of nonhuman and noncoding RNA protein interactions remain at the frontiers of science.[2] To understand relevant mechanisms of action, obtaining 3D structures is often required and usually involves x-ray crystallography or NMR.[3] The former requires protein crystallization, an often arduous and difficult task. Moreover, purifying RNA-protein structures is difficult because of the nature of RNA structure. One issue is that the interface of RNA has many phosphate groups that are negatively charged. Because of the repulsive force between such negative charges, crystallization is problematic. The second issue is that the shape of RNA is often not globular unlike proteins, where crystallization of non-globular molecules can also be problematic due to the difficulty of forming regularized structures.[4] An existing option for the crystallization involves in vitro preparation that mixes pure RNA and pure protein together. Here, in vivo expression and purification using recombinant RNA in E coli has recently been implemented. Recently, a method for co-expression and co-purification of both RNA and protein was presented[5] but it is still in its infancy.

Though NMR resolution is often less robust than x-ray, NMR provides information on dynamics of flexible structures such as RNA. Recently, problems

analyzing extended RNA molecules[6] were in part overcome by combining NMR with x-ray crystallography or cryo-EM.[7] In addition to the intrinsic flexibility, RNAs show conformational rearrangement in contact with other macromolecules such as protein.[6] These dynamics of RNA caused by ligand can be elucidated with NMR and x-ray crystallography.[8] Still, with increasing demand for the 3D structures of RNA-protein complexes, reliable computational prediction of modeling 3D structures is required. Though there has been a 50-fold increase in the number of high quality structures, in general that is just a fraction of complexes identified biologically.

1.2. Prediction of RNA Structure

Although the computational prediction of RNA secondary structure has been successfully applied when combined with experimental results using SHAPE chemistry,[9] the prediction of 3D structure is not yet as well developed. The predictions of RNA 3D structure include physics-based bottom-up predictions[10] and knowledge-based predictions.[11] The prediction of RNA 3D structure from sequence has utilized knowledge-based modeling and machine learning, showing some success for short RNAs.[12] However, compared with the prediction of protein structure, prediction of RNA 3D structures especially for long chain RNAs of more than 50 nucleotides remains problematic.[13]

One of the latest successful methods employed the Nash equilibrium of Game Theory in sampling of configurational space.[14] Development of 3D modeling is in part limited by the amount and diversity of structural data on RNA at the atomic level. More importantly, the development of suitable RNA-protein potentials is required to do computational modeling of 3D structure.

1.3. Prediction of 3D Structure of Protein

Today, many protein 3D structures can be obtained at the RSCB Protein Data Bank (PDB). The quantity of uploaded protein structures is much more than those of RNA: 46,985 PDB entries including protein chains and 4,559 PDB entries including RNA chains were found (Protein Data Bank, 2016). However, the available 3D structural data for RNA and protein are only a small portion of known sequences. At the GenBank, more than 190 million sequences can be downloaded (GenBank, 2016), but available 3D structures of RNA-protein complexes at the PDB are less than 1,800. Similar to RNA, this difference of the availability in part comes from limitations of experimental methods such as x-ray crystallography[15] and NMR.[16] This is why computational prediction is still essential for the 3D structure determination of protein.

Computational prediction of 3D structure is divided into two broad approaches: physical and comparative modeling.[17] The former is based on physical principles that calculate forces and interactions to estimate the structure with minimum potential energy. Comparative modeling typically utilizes physical principles and known sequence and structural data. The comparative strategy includes homology modeling[18] and folding recognition.[19] One of the recent successful programs, Rosetta,[20] searches fragments of similar sequence and assembles them using potentials calculated from experimental data.[21] Most de novo protein modeling does not work for proteins of more than 150 residues.[22] However, recent application of methods that predict conserved tertiary pair has shown promise.[23]

1.4. Prediction of 3D Structure of RNA-protein Complex

The prediction of RNA-protein 3D structure is still an ongoing issue. Although crystallization and subsequent x-ray crystallography itself or NMR can provide high resolution coordinates of complex structure, implementing the approaches can be problematic.[24] Computational prediction lags experimental approaches in overall accuracy. Cryo-EM (Electron Microscopy) is emerging as a powerful method for 3D structures at high resolution, especially for large molecules such as membrane proteins and viruses.[25] Cryo-EM does not require any crystallization. Instead, it freezes the purified solution, takes a large number of images using electron microscopy, and aggregates the images to determine the 3D structure of the macromolecule by analyzing the pictures at atomic level. Note that 231 entries of RNA-protein structures have been obtained by electron microscopy compared to 1,416 entries via x-ray crystallography and 105 via solution NMR (Protein Data Bank, 2016).

The computational universe of RNA-protein docking algorithms and software is much smaller than those involving protein-protein docking. Note that GRAMM,[26] FTDock,[27] and Rosetta[20] are relatively popular docking programs that accept atomic coordinates for RNA and protein 3D structures. FTDock orients macromolecules into an orthogonal grid and samples the configurational space for translational and rotational movement. Fast Fourier transform can be used to increase the speed of calculation.[28] GRAMM and FTDock employ rigid structures of RNA and protein, and evaluate bound structures by scoring with nucleotide-amino acid potentials and not all-atom ones. Such potentials applied to these rigid all-atom configurations are consistent with coarse-grained modeling of simpler structural representation (e.g. lattice models) that

allows exhaustive sampling of configurational space, but may sacrifice additional details associated with all-atoms. FTDock and GRAMM align both backbones of the macromolecules on a grid, and then add the remaining atoms such as those in the side chains. On the other hand, Rosetta, as the first step, samples configurational space with only the backbones of the rigid-body macromolecules; then side chains are repacked and the displacement of both the side chains and backbone can be optimized using Monte Carlo minimization. In summary, such methods sample configurational space in docking two molecules, and then score the resulting poses with potentials, and in some programs, additionally refine the generated poses.

In the typical coarse-grained models, the score of a pose is calculated as a sum of scores of RNA-protein interactions. The statistical potential is well known to be a simple but powerful approach for scoring such interactions. It is calculated from a propensity that is, in most cases, obtained from the expected or theoretically deduced probability normalized by the observed probability of a certain type of contact. If a certain propensity is much larger than others, it means that the type of interaction happens more often than expected, which implies there is a stronger preference for the interaction than others. The statistical potential ΔG is often calculated from propensity P by the following equation,[29,30] in which C is a constant.

$$\Delta G = -C \times ln\, P$$

Calculating propensity depends on the classification of the contacts. A simple classification based solely on amino acid and nucleotide type was developed.[29] A subsequent approach included accessible surface area in the calculation of propensity.[30] Some approaches classify interactions based on their geometric and electrostatic

properties.[31] One of the most successful methods employs distance criteria[32,33] as well as angles.[34] Typically these interactions are hydrogen bonds.

The differences in binding potentials for RNA-protein versus DNA-protein interactions were explored in terms of recognizing four bases. Lustig et al. calculated pairwise statistical potentials between the amino acid and the base component of RNA.[29] They counted hydrogen bonds between RNA and protein for U1RNA-spliceosomal protein, and for seryl, aspartyl, and glutaminyl-tRNA synthetases with their variants, where the protein sequence at the hydrogen bonds was assumed to be conserved. Here, the normalization involved the logarithm of the frequency for a given amino acid, averaging with respect to the four bases such that the sum of the appropriately weighted logs is zero. The normalized relative potentials were calculated for ten amino acids (Arg, Asn, Lys, Asp, Gln, Glu, Ala, Tyr, Ser, and Thr). Ser and Thr were plotted as one because the two sets of frequency data were identical. Initially, the definition of major groove interactions included not only the RNA A-form helix but alternative forms that afford contacts with atoms allocated to the major groove. In addition, potentials at specifically identified major grooves were separately calculated for Arg and Asn. Comparing RNA and DNA in the major groove, Arg most prefers guanine in both cases (the order of preference was G, A, U, and C for RNA, and G, U, C, and A for DNA in descending order).

The correlation coefficient between RNA and DNA for the potentials of Arg and Asn in the major groove was 0.5 (p-value < 0.21). The Arg and Asn data included those not in the major groove but interacting with base atoms that are usually accessible in the major groove. The correlation plots indicate a somewhat weak correlation between RNA

and DNA for potentials of Arg and Asn in the major groove, even though the specific structural differences are minimal. In that correlation plot for Arg and Asn, Asn was clearly more correlated between RNA and DNA.

Notably Arg and Lys had strong interactions, mostly with the major groove of RNA, and the comparison of the statistical potentials showed strong similarities between the rank orders of contacting bases of RNA and DNA.[29] It is known that double stranded RNA (dsRNA) has a stronger affinity for protein than single stranded RNA or double stranded DNA.[35] Here, a zinc finger protein called ZNF346 has a strong affinity to dsRNA, especially with regard to Lys and Arg contacts which appear particularly suited to allow access in the deep and narrowed major groove of RNA.[29] This type of binding is essential for protein moieties such as the zinc fingers to recognize RNA.

2. METHODS

2.1. Research Overview

This research calculates statistical potentials and evaluates them over a test set of twenty-one standard complexes.[30] It consist of five components: (1) preparation of training data set, (2) calculation of potentials, (3) docking of the test set, (4) scoring the generated poses, and (5) evaluation of the potentials (see Table 1). Third-party programs such as FTDock were employed in all of the five components, and the in-house software developed here automates the implementation of the five components. The work discussed here puts an emphasis on developing and analyzing potentials, especially in regards to the novel classification of contacts. The learning set used in the classification of contacts is determined by all possible hydrogen bonds. With regard to the test set, the various potentials are implemented.

Table 1. List of associated computational methods.

1. Calculation of Potential Sets	2. Docking and Scoring with Potential Sets
Obtain 1345 mmCIF files	Generate 50,373 poses (FTDock)
Cluster protein chains (PISCES)	Calculate hydrogen bonds (DSSR)
Calculate hydrogen bonds (DSSR)	Obtain RNA 3D structures for poses (DSSR)
Obtain RNA 3D structures (DSSR)	Score the poses with the nine potential sets
Classify contacts into nine categories	Rank native structures by score
Calculate nine sets of potentials	

2.2. Calculation of Potential Sets

2.2.1. PISCES. The 1,345 RNA-protein mmCIF files were downloaded (Protein Data Bank, 2015). Then the protein chains were clustered by PISCES (PISCES, 2015) at 25% similarity for the PDB entries, and 165 clusters were obtained. Only the structures determined by x-ray crystallography with resolution of 3.5 Å or below and with one-letter chain ID were used. The mmCIF files that have the best resolution in each cluster were selected as the complexes of record. The default maximum R-value of 0.3 and a minimum chain length of 40 were used. Note that the structural coordinates were downloaded as an mmCIF format instead of a pdb format for easy updating of our program in the future, although in this study one could not utilize the other information in the mmCIF files at all. One of the problems with the pdb format is that the pdb format has just one digit to identify a chain. For example, 4V6X (structure of the human 80S ribosome) does not have pdb files but only mmCIF files because 4V6X has 89 chains and one digit cannot accommodate that many chain IDs. However, we could not find any third party programs for analyzing hydrogen bonds and RNA 3D structures, which deal with mmCIF files and can handle those large complexes.

2.2.2. X3DNA-DSSR. X3DNA-DSSR[36] was first employed for two purposes, to obtain hydrogen bonds between protein and RNA, and to determine the secondary

structure of RNA. Here, hydrogen bonds are obtained from the 663 non-redundant complexes by applying X3DNA-DSSR. Finally, the hydrogen bonds were categorized according to the RNA structure, position, and base pair type. For example, if a hydrogen bond is at N6 of a base, and if the base makes a canonical base pair involved in an A-form helix as noted by DSSR, the hydrogen bonding is at the major groove, so the hydrogen bond will be categorized as 1.

Hydrogen bonds between RNA and protein are determined by the distance and angle between the donor atom and acceptor atom. The algorithm of detecting hydrogen bonds in X3DNA-DSSR is not fully transparent, but it tends to provide more contacts than HBPLUS[37] and is considered a standard procedure.[36]

2.2.3. Nine Categories. Hydrogen bonds between RNA and protein for the training set were classified into nine categories (Figure 1). Then, in each category, 80 (20 amino acid × 4 bases) pairwise potentials were calculated. Therefore, 720 potentials in total were calculated from the training set. Contacts in Category 1 include RNA atoms belonging to the major groove side of an A-form helix. A-form helix is regarded as the most common secondary structure for RNA[38]. Each category is assigned by the helicity of RNA,

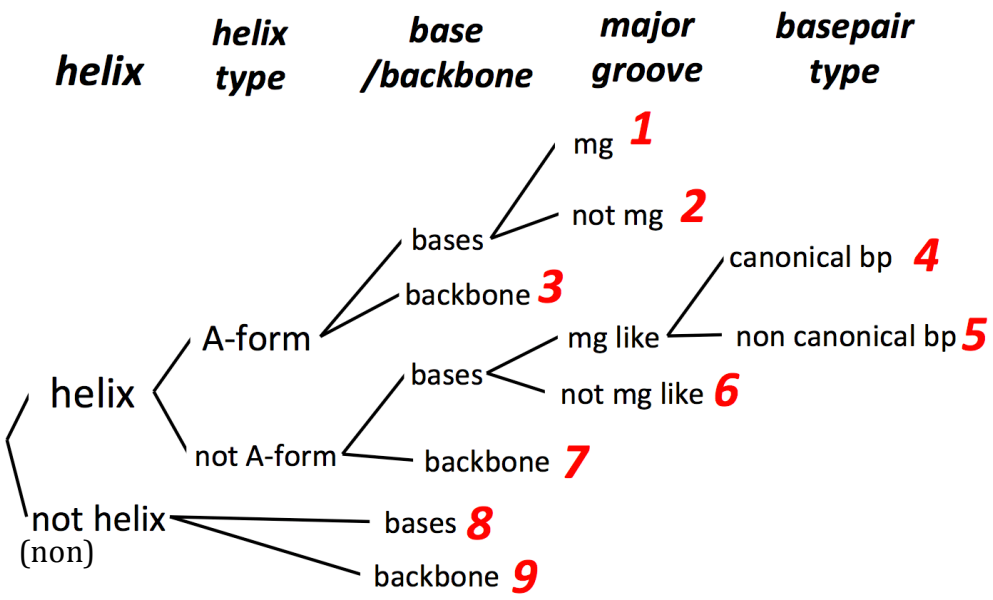

Figure 1. Nine categories defined by RNA structure (mg = major groove, bp = base pair).

helix type, location of the atom (base or backbone, ribose is included in the backbone), and base pair type. Accordingly, contacts in Categories 3, 7, and 9 include RNA backbone, and contacts in Categories 1, 2, 4, 5, 6, and 8 include a base of RNA.

Specifically, if a contact includes N4 of cytosine, N7 or O6 of guanine, O4 of uracil, N6 or N7 of adenine, or O4 or N7 of uracil in U-U base pair, in A-form helix RNA, the contact is classified as Category 1. Here, keeping our program simple, only the contacts in a canonical base pair (indicated cW-w in DSSR) were classified as Category 1. Those contacts that include RNA atoms on the major groove side of a non-canonical base pair were classified as Category 2 (not major groove).

For contacts on the major groove side not in A-form helix, they were classified into two categories according to the base pair type. If the contact was in the canonical base pair, it was classified as Category 4; otherwise it was classified as Category 5. Identification of the helix type (A-form or not) depends on the output of DSSR.

Categories 4 to 7 include contacts in a B-form helix, Z-form helix, unclassified helix, and any helix with backbone breaks.

2.2.4. Statistical Potentials. The statistical potentials were calculated from the following equations.[29,30] The statistical potential was calculated from the propensity that is a value of observed probability of the pairwise (e.g. Arg-guanine) contact in the category (1-9) divided by the expected or theoretical probability of the pairwise contacts (Equation 1).

Propensity:

$$P(p,q,s) = \frac{N(p,q,s)/\sum_{pq} N(p,q)}{N(p,s)/\sum_{p} N(p) \times N(q,s)/\sum_{q} N(q)} \qquad (1)$$

p : amino acid (1-20)
q : base (1-4)
s : category (1-9)

where N(p,q,s) is the number of contacts between amino acid p and base q in the classification of the category s, and N(p,q) is the number of contacts between amino acid p and base q. Therefore, the numerator of equation 1 is the observed probability for the pairwise contact for a particular category. The denominator, the theoretical probability of the contact, is the product of fractions of an amino acid and a base occurring in all chains in the training set. In the denominator, N(p,s) is the number of amino acids for s classification in all protein chains in the training set, and N(p) is the number of the amino acids for all proteins. Note, because identifying Category 1 or 2 requires base pairing, Categories 1 and 2 have the same value for N(p,s). Categories 4, 5, and 6 also have among themselves the same N(p,s) values. Potential energy is calculated from the propensity by the following equation:

Potential Energy:

$$\Delta G(p,q,s) = -RT \times ln\,(P(p,q,s))$$

where $\Delta G(p, q, s)$ is the potential for the amino acid p and the base q in the category s, R is the gas constant, and T is temperature in K. For RT, 0.59 was used as the value. Both the propensity and the potentials are the functions of three arguments: amino acid, base and structure category of RNA. For example, the propensity between Arg and guanine in Category 1 is calculated as follows.

$$P^I(\text{Arg, guanine,1}) = \frac{\frac{count\ of\ Arg-guanine\ pairs\ in\ category\ 1}{count\ of\ all\ pairs}}{\frac{count\ of\ Arg\ in\ category\ 1}{count\ of\ all\ amino\ acids} \times \frac{count\ of\ guanine}{count\ of\ all\ nucleotides}} \qquad (2)$$

In one of the cases involving redundant set, the number of pairs in Category 1 was 755, and 437 of the pairs were Arg-guanine. The count of all amino acids in Category 1 was 71,335, and 6891 of them were Arg. The count of all nucleotides in Category 1 was 809,393, and 259,504 of them were guanine. Amino acids and nucleotides were counted in any protein that had at least one Arg-guanine hydrogen bond. Then the value of the propensity was calculated as follows.

$$P^I(\text{Arg, guanine,1}) = \frac{\frac{437}{755}}{\frac{6891}{71335} \times \frac{259504}{809393}} = 18.69$$

Alternatively, potentials without a structure category of RNA were calculated as an "aggregate average." As in equation 3, when no contact is found in the training set, 0.001 is used instead of 0 as the propensity for calculating the statistical potential. In that case, statistical potentials will be 4.076 (RT = 0.59 kcal/mol).

$$0.59 \times ln(0.001) = 4.076 \qquad (3)$$

For example, because Ala-adenine in Category 1 has no contact in the training set, P^I(Ala, adenine, 1)$= 0.001$, therefore $\Delta G(Ala, adenine, 1) = -0.59 \times ln\, 0.001 = 4.076$.

Propensity (aggregate average):

$$P^I(p,q) = \frac{N^I(p,q)/\sum_{pq} N^I(p,q)}{N(p)/\sum_p N(p) \times N(q)/\sum_q N(q)} = \frac{Observed\ probability\ of\ the\ pair\ type}{Expected\ probability\ of\ the\ pair\ type}$$

Potential Energy (aggregate average): $\Delta G(p,q) = -RT \times ln\,(P^I(p,q))$

For instance, in one case, the number of pairs was 247,044, and 23,623 of them were Arg-guanine. Then the value of the propensity was calculated as follows.

$$P^I(\text{Arg, guanine}) = \frac{\frac{23623}{247044}}{\frac{868}{10373} \times \frac{37886}{114036}} = 3.440$$

2.2.5. FTDock. FTDock is a rigid-body docking program based on Fourier transform.[28] FTDock 2.0.3 was employed to generate 50,373 poses for standard complexes and ran with default options including no calculation of electrostatics.[30] A Perl parameter file of FTDock was edited so that FTDock recognized all atoms in RNA. Since FTDock is a rigid-body docking program, and it allows an exhaustive exploration of a particular ligand conformation binding to a fixed target. Of course the caveat is that side chain repacking and flexible features in RNA are not fully accounted. Methods for such modeling are not yet fully developed. However, we employed FTDock for two reasons. First, one of the objectives for this study is to develop and evaluate potentials for comparing with previously reported ones, and not the prediction itself. Secondly, FTDock is capable of RNA-protein docking and easily incorporates customized RNA-

protein potentials.[30] Courtesy of Dr. Graham Smith, we employed the developer's version of FTDock (FTDock v2.0.3) which is compatible with multiprocessing (openmpi).

2.3. Docking and Scoring with Potential Sets

2.3.1. Scoring. Each pose was scored with nine sets of potentials. First, hydrogen bonds and the corresponding RNA structures were calculated by X3DNA-DSSR and all the hydrogen bonds were assigned a category according to the RNA structure and the position. Secondly, each pose was scored by adding up all the scores of the hydrogen bonds in the pose. Here, the distance of hydrogen bonds was not taken into account in scoring.

2.3.2. Evaluating Benchmarks. Shown in Table 2 are the test complexes and unbound protein and RNA chains. Docking a protein chain (Column 3) and an RNA chain (Column 6) using FTDock generates 50,373 poses, and then RMSD value between each pose and a complex structure (Column 2) is calculated after structure alignment. PyMol (version 1.8) is used to achieve both of the structural alignment and calculation of RMSD value. In the structure alignment, an unbound or bound protein structure (Column 3) is aligned to the protein structure in the corresponding complex structure (Column 2) using only the alpha carbon atoms. Then the RMSD value is calculated using all atoms in the two RNA structures (Column 2 and Column 6). This process is coded using python package 'pymol' that allows PyMol command written in a python program. The generated raw output is parsed by 'RMSD2.py' and a table of pose id and RMSD is made (RMSD2.out).

The pose was regarded as a native structure when RMSD was less than 10 angstrom.[30] Twenty-one standard complexes that have their unbound protein/RNA structures were chosen (Table 2). In other words, RNA chain and protein chain were taken from different PDB entries and unbound structures are considered to be more difficult cases as test complexes than bound structures.[32]

Table 2. Test set of protein-RNA complexes.

	Complex (bound)	Protein (unbound)	Size[a]	RMSD[b]	RNA (unbound)	Size[c]	RMSD[d]
1	1WSU_a_e	1LVA_a	2191	0.7	1MFK_a	740	0
2	2PJP_a_b	2PJP_a	982	0	1MFK_a	740	3.1
3	1LNG_a_b	1LNG_a	727	0	1Z43_a	2169	2.1
4	1E7K_a_c	2JNB_a	2031	3.2	1E7K_c	365	0
5	1WPU_a_c	1WPV_a	1095	0.2	1WPU_c	145	0
6	2QUX_a_c	2QUD_a	2006	0.7	2QUX_c	531	0
7	2JEA_a_c	2JE6_a	2119	0	2JEA_c	88	0
8	2FMT_a_c	1FMT_a	2350	1.2	3CW5_a	1645	2.9
9	1MFQ_c_a	1QB2_b	966	3.1	1L9A_b	2683	5.1
10	1U0B_b_a	1LI7_a	2961	1	1B23_r	1584	6.6
11	1EC6_a_d	1DTJ_a	525	1.6	1EC6_d	1081	0
12	1HC8_a_c	1FOY_a	1169	2.9	1HC8_c	1219	0
13	1JBR_b_d	1AQZ_a	1129	0.6	1JBR_d	664	0
14	1KOG_a_i	1EVL_a	3265	0.6	1KOG_i	785	0
15	1M8W_a_c	1M8Z_a	2750	1.2	1M8W_c	167	0
16	1F7U_a_b	1BS2_a	4874	3.4	1F7U_b	1629	0
17	1K8W_a_b	1R3F_a	2158	2.2	1K8W_b	466	0
18	1N78_a_c	1J09_a	3814	1.9	1N78_c	1597	0
19	1U63_a_b	1I2A_a	1682	1.3	1U63_b	1055	0
20	2BTE_a_b	1H3N_a	6642	4.1	2BTE_b	1674	0
21	2HW8_a_b	1AD2_a	1712	6.7	2HW8_b	774	0

[a] Number of protein atoms.
[b] RMSD (Å) of a complex structure (Column 2) and a protein structure (Column 3) calculated with alpha carbon atoms of both structures.
[c] Number of RNA atoms.
[d] RMSD (Å) of complex structure (Column 2) and RNA structure (Column 6) calculated with phosphorous atoms of both structures.

Note that three complexes in Table 2 (2FMT, 1MFQ, and 1U0B) consist of three different PDB entries and others consist of two different PDB entries. Therefore, at least one of the RNA chains or protein chains has a non-zero RMSD value, even before

docking. For example, the protein chain 1AD2_a has RMSD value of 6.6 (Row 22). Perez-Cano et al. adopted unbound docking set,[30] and this study also adopted an unbound docking set to compare our results with theirs. Each complex was excluded from the training set in each calculation. As a comparison, the QUASI-RNP scoring program and DARS-RNP were downloaded and applied to the same poses. The amino acid-nucleotide potentials from the study of Perez-Cano were obtained by digitizing the color intensity (Park and Lustig, unpublished results) of given graphics.[30] In addition, the best rankings from potential sets with no filtering and no clustering were also calculated.

2.3.3. Best Rank for the Test Set. The 50,373 poses of a test set complex were ranked by binding energy, and then the rank of the most stable, low RMSD (< 10 Å) native-like complex structure was calculated as belonging to the relevant percentile of the 50,373 poses. For example, if the test set 1KOG had 3 native like structures among 50,373 poses and the lowest score of these native-like structures (most native-like) was the 10,000th score ranked by binding energy, the best rank for 1KOG would be 19.85%. The average of the best ranks of the twenty-one test set complexes was also calculated to evaluate the scoring method.

2.3.4. Success Rate of Prediction. Another evaluation for the scoring method is calculated as the success rate for each test set complex.[30] And in addition, the success rate is calculated for six scenarios (see Table 3). For instance, one calculates the success rate among all twenty-one-test complexes, for whether the native-like structure's energy is identified among the threshold-filtered values. For example, for a threshold of ten, one identifies whether the native-like structure and its energy is among the lowest ten energies. The success rate is the fraction of the 21 test proteins that meet that criteria.

For example, when the threshold was 1000 (i.e., checking among the first 1000 poses) and the current nine-category set of potentials had only one test set complex that had the best score within that top 1000, the success rate would be 4.76% (1/21×100). In the ideal prediction, the success rate would be always 100% no matter the range.

2.3.5. Six Scenarios. In this study, six scenarios (the first three calculated here) for calculating potentials (Table 3) are evaluated. These six scoring scenarios are applied to the 50,373 FTDock poses to compare the success rate, best rank and other analytics. Potentials for Perez-Cano were calculated from non-redundant hydrogen bonds between RNA and protein.[30] The contacts with more than 70% sequence identity were clustered, then the x-ray coordinates with the best resolution for each cluster was chosen as a representative of the cluster. As a result, 282 RNA-protein contacts were obtained and used to calculate potentials. The equation of Perez-Cano is the same as Equation 1, but their calculation includes only the atoms on an accessible surface area. The potentials are pairwise base-amino acid potentials, so the total number of the potential is 80 (4 bases × 20 canonical amino acids). They employed FTDock to generate binding modes for a test set. These potentials for Perez-Cano are presented graphically.[30]

Table 3. Six scoring scenarios.

Scenario	Description
Current	Nine-category set of potentials, calculated from non-redundant contacts as selected by PISCES (R-value < 0.30, x-ray resolution < 3.5 Å, sequence identity < 25%).
Current Redundant	Nine-category set of potentials, calculated from complete list of RNA-protein contacts (without clustering and filtering).
Aggregate Average	One category set of potentials (without classifying by RNA secondary structure).
Perez-Cano	Potentials obtained by digitizing the color intensity of the heat map in the literature.
DARS-RNP	https://genesilico.pl/index.php/software/35.html?sectionid=1
QUASI-	https://genesilico.pl/index.php/software/35.html?sectionid=1

The potentials for QUASI-RNP and DARS-RNP are also calculated from the redundant contacts.[34] The training set complexes are selected by choosing from 3.5 Å or better resolution x-ray crystal structures and clustered by sequence identity of more than 30% for protein chains and 70% for RNA chains. They obtained seventy-two RNA-protein complexes to make the scoring function. The arguments of the scoring function are amino acid (20), base (4), and a contact distance divided by angle of the hydrogen bond. QUASI-RNP calculates potentials purely from observed native structures, but DARS-RNP includes coordinates from decoy structures generated by GRAMM.[27]

2.4. Flow Chart of Automated Process

2.4.1. Overview. Almost all processes from preparing training data set to calculating relative ranks for the test set were automated with our python programs and shell scripts. The whole code was divided into many files so that we could debug with ease and analyze each output at every step. All the separated programs are called and executed by one program file 'director.py' in order.

2.4.2. Preparing Training Set. Figure 2 describes how the non-redundant contacts were prepared automatically. The figure describes the relationships and order of python programs (red boxes) and data files (blue boxes). Arrows represent outputs and inputs of the programs. Note that the purple boxes represent programs or websites other than our python programs and the arrows for the purple boxes represent manual processes. For example, '5let_inputchains.txt' and mmCIF files were downloaded manually from PDB (shown in the top-right corner of Figure 2).

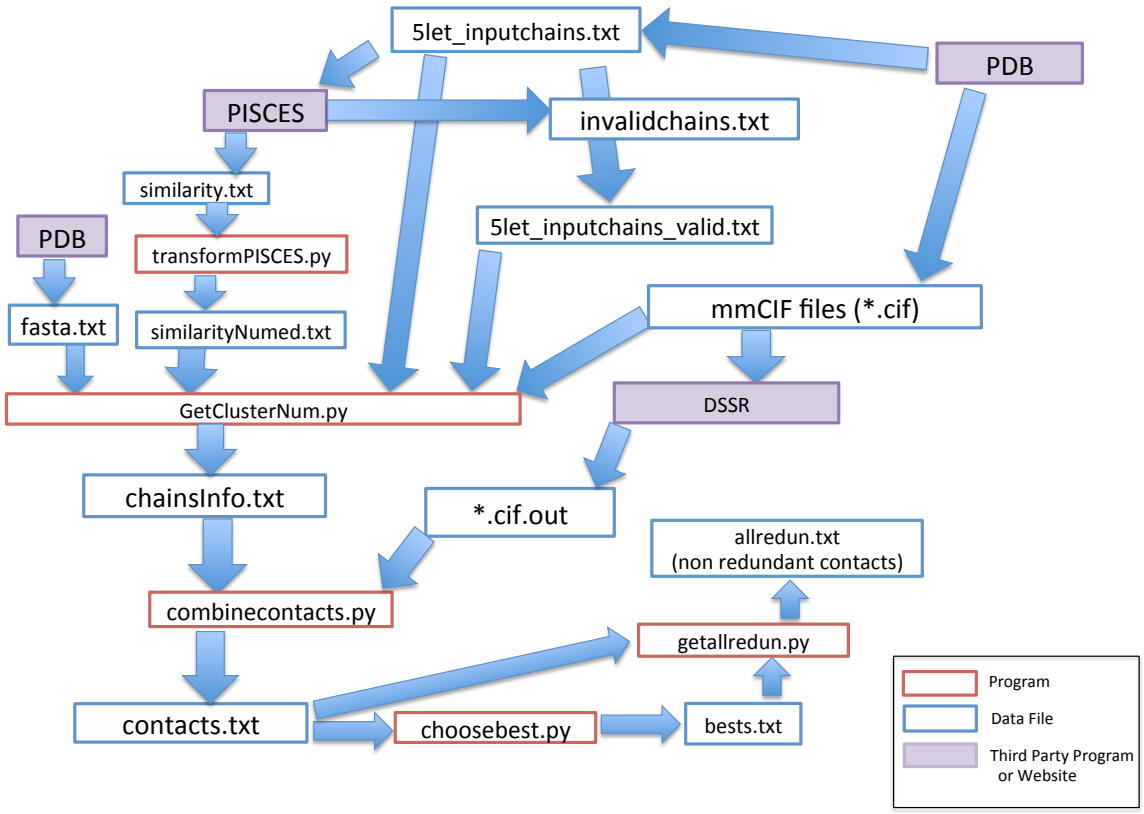

Figure 2. The flowchart for preparing training set. Red, blue, and purple boxes, respectively, represent program, data file, and third party program or website. Arrows starting from or to purple boxes represent the processes manually done before running the program. Other arrows represent input or output for the python program.

Initially, the structural coordinate files for the training set (1,345 complexes) were downloaded manually from PDB in mmCIF format. Then the complexes were culled at chain level using the PISCES online server to filter and obtain cluster identifiers. Hydrogen bonds for the downloaded 1,345 complexes were obtained by manually running DSSR.

The valid chains, cluster numbers, and resolution for x-ray crystal structures were summarized in one file (by GetClusterNum.py). Hydrogen bonds between RNA and protein were extracted from the output files for DSSR and combined with the chain information (by combinecontacts.py). Finally, the contact with the lowest resolution in

each cluster was chosen (by choosebest.py), and all the other contacts of that pair of chains were restored to make a file of non-redundant contacts (by getallredun.py).

2.4.3. Calculating Potentials. The filtered redundant contacts are then used to calculate statistical potentials in three paths (Figure 3). One path starting with 'assignTNA3Dall.py' calculates potentials for the aggregate average scenario (see Table 3). The second path starting with

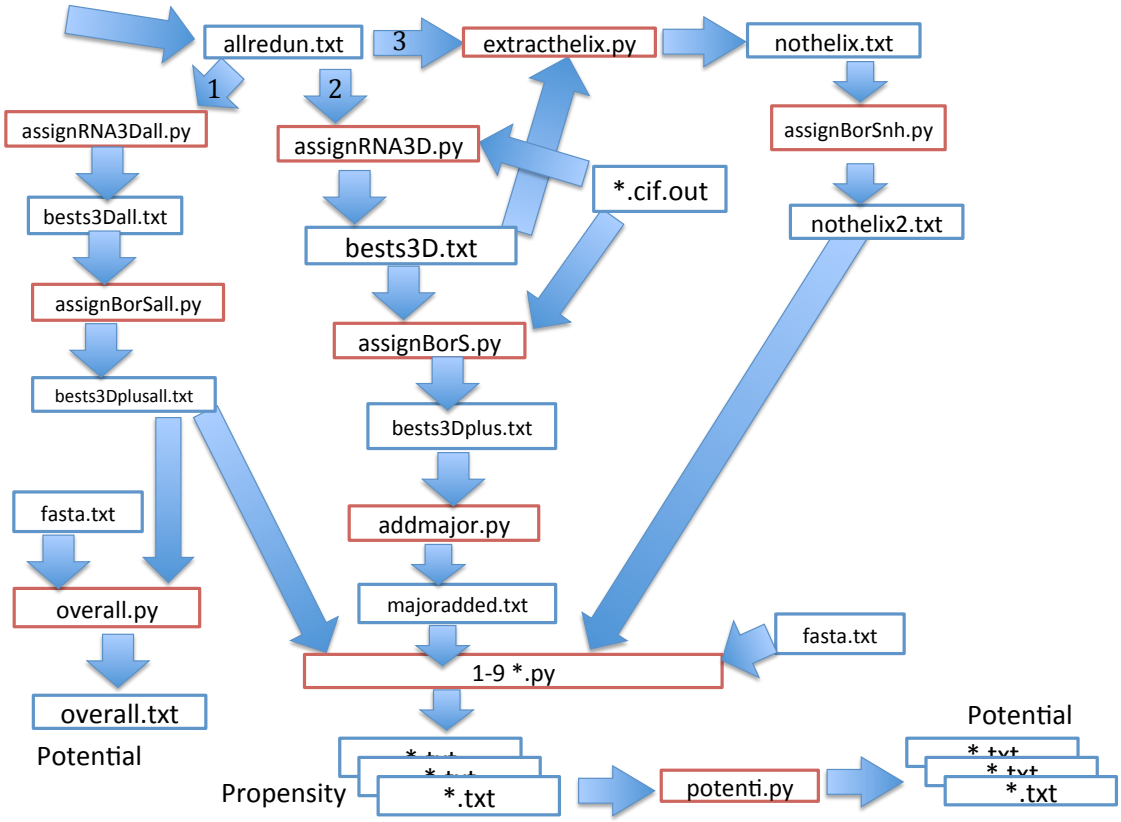

Figure 3. The flowchart for calculating potentials. Red and blue boxes, respectively, represent program and data file. Arrows represent input or output for the python program. The 'allredun.txt' comes from the previous process. The 'allredun.txt' is then used as inputs for three programs that calculate propensity for all contacts ('assignRNA3Dall.py'), contacts in helix (assignRNA3D.py), and contacts in not helix ('extracthelix.py').

'assignRNA3D' calculates potentials involving RNA stem for current scenario. The third path starting with 'extracthelix.py' is for calculating potentials in non-helical classifications for the current scenario. The output files from DSSR are used as inputs for

RNA structural information in each path, and the FASTA file downloaded from PDB is also used as an input file to calculate the fractions of amino acid or nucleotide for Equation 1. Calculated propensities for the current scenario are then transformed into potentials by 'potent.py.'

2.4.4. Evaluating Potentials. The next process is to evaluate the calculated potentials using test complexes (Figure 4). The test complex files (pdb files) are downloaded from PDB (PDB 2015). The mmCIF gives the same results. The coordinates for necessary chains (see Table 2) are extracted from the downloaded pdb files by 'correct.py.' The protein chain and RNA chain are used as inputs for FTDock, and 50,373 docking poses are generated. RMSD values are calculated by 'RMSD.py' and hydrogen bonds for each pose are obtained by 'DSSR_hbonds.py.' RNA secondary structure is also obtained by running 'DSSR.py' and each atom in the RNA coordinates file is assigned the corresponding structure category (see Figure 1) by 'assign3D.py.' All of the information, including hydrogen bonds, category for each atom, and RMSD for each pose, are combined and the score for the pose is calculated by 'calpot.py.' Here for a given test set complex, its corresponding learning set is excluded and a potential set is newly calculated in each prediction. Running FTDock, DSSR (for hydrogen bonds), and 'calpot.py' utilizes multiprocessing python package (openmpi) to improve running times.

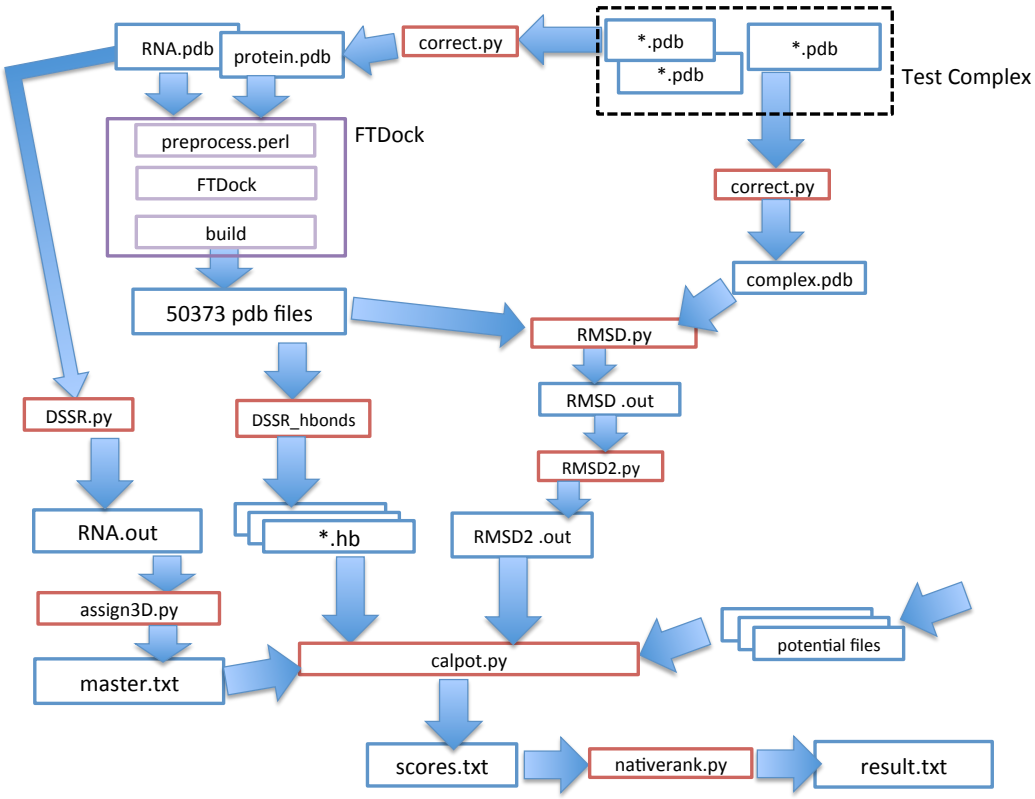

Figure 4. The flowchart for evaluating calculated potentials. Red, blue, and purple boxes, respectively, represent program, data file, and third party program. Arrows represent input or output for the python program. The black dotted line represents test complex files that consist of RNA coordinates, protein coordinates, and complex coordinates files. The potential files to score generated poses are obtained from the previous process. FTDock is embedded in a shell script 'FTDock.sh' and called from python program 'director.py' and will also implement the characterization of the hydrogen bonds and RMSD for the test set.

3. RESULTS

3.1. Classified Contacts in the Training Set

Figure 5 shows the classification of nine categories for the protein-RNA contacts in the learning set as shown in Figure 1 (see Appendix A for details), but the number of calculated contacts is now included. Categories 3, 7, and 9 indicate the contacts in RNA backbones and have 3,171, 5,881, and 3,330 contacts, respectively. These three categories occupy 74.3% of all contacts.

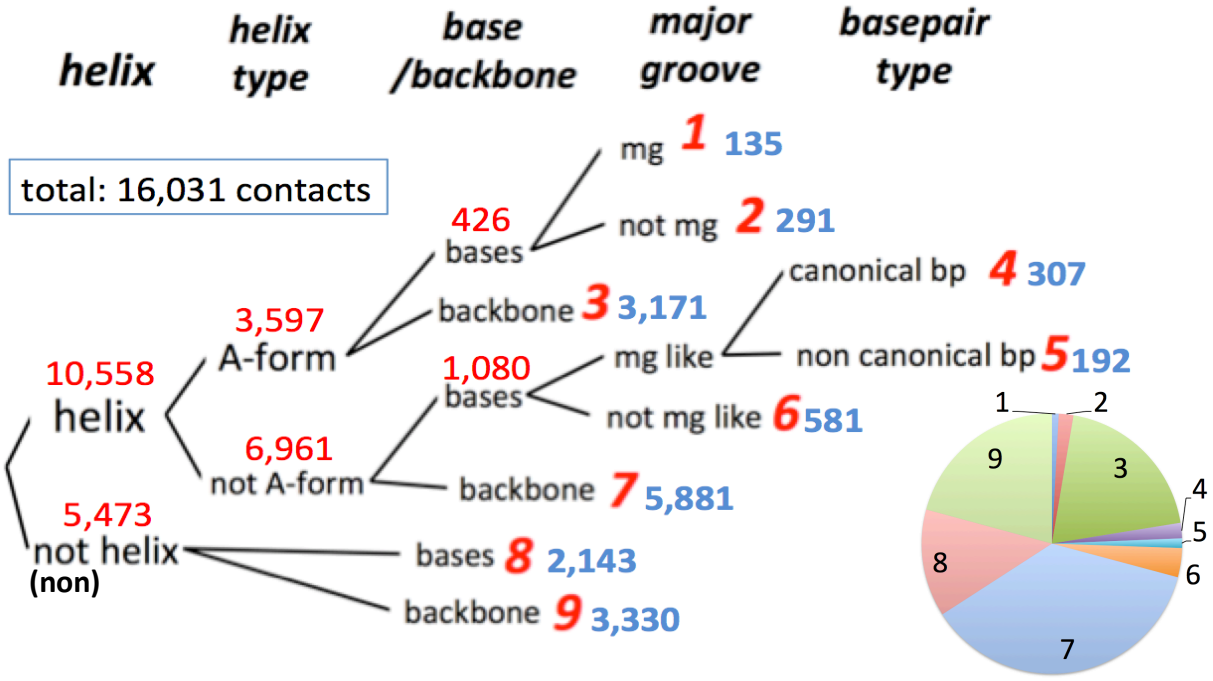

Figure 5. The number of hydrogen bond contacts between RNA and protein in each of the nine classes. All hydrogen bonds (16,031) were divided into nine categories by the classification of RNA structure as shown above. The classification derives from the output of DSSR that defines A-form helix as the helices with canonical base pair and without any break of backbone. Backbone includes phosphate and ribose. Contacts in major groove side and not in A-form helix are classified as 'mg like.' 'Non canonical bp' is the irregular base pair in which the base is flipped or rotated (except "cW-W" in the DSSR output). The inserted pie graph shows the fractions of hydrogen bonds in each class. Please note that bifurcated hydrogen bonds are calculated twice (e.g. hydroxyl hydrogen atom for Ser329 in 1ASY_B makes two contacts with O2 and N3 for C674 for 1ASY_S).

3.2. Best Rank for the Test Set

The relative rank for each test complex and the six scenarios is shown in Table 4. For instance, for the first entry 1E7K (crystal structure of the spliceosomal 15.5 KD protein bound to a U4 snRNA fragment), there are 22 native-like structures, and the best rank percentile for the current scenario is 0.042% (column 3, $21\times100/50373$). The best rank for aggregate average for 1E7K was 0.008% (column 4, $4\times100/50373$). The best rank for current redundant (potentials calculated without clustering and filtering training set) was 0.008% (column 5, $4\times100/50373$). Using Perez-Cano potentials resulted in

the best relative rank of 0.004% (column 6, $2\times100/50373$). The best rank for QUASI-RNP for 1E7K was 0.993% (column 7, $500\times100/50373$) and DARS-RNP was 0.280% (column 8, $141\times100/50373$).

Table 4. Rank of native structures (percentage of total poses).

Complex	Number of Native Structures[a]	Current	Aggregate Average	Current (redundant)[b]	Perez-Cano	QUASI-RNP	DARS-RNP
1E7K	22	0.042	0.008	0.008	0.004	0.993	0.280
1EC6	200	0.105	0.451	0.143	0.927	12.431	5.364
1F7U	5	0.977	0.250	1.856	0.435	1.136	0.191
1HC8	95	8.751	4.125	7.810	2.001	0.244	0.083
1JBR	74	1.642	1.139	1.239	0.393	0.054	0.123
1K8W	4	4.203	25.123	8.221	11.667	70.750	62.154
1KOG	1	4.260	37.349	8.914	50.386	99.144	99.927
1LNG	28	1.441	0.056	0.236	0.038	0.099	0.038
1M8W	278	0.218	0.034	0.169	0.177	0.004	0.058
1MFQ	3	61.118	2.150	27.844	3.649	41.139	50.730
1N78	7	11.345	18.764	13.394	18.071	70.010	13.620
1U0B	6	14.204	10.462	4.572	7.399	4.137	1.632
1U63	48	0.177	0.099	0.103	0.046	0.129	0.095
1WPU	222	0.478	0.189	0.439	3.053	0.107	0.040
1WSU	465	0.022	0.119	0.018	0.113	2.718	2.124
2BTE	14	1.304	0.534	0.613	0.276	0.250	0.609
2FMT	9	0.030	0.788	0.026	0.695	2.257	1.914
2HW8	36	1.374	0.905	1.697	0.709	4.494	5.036
2JEA	642	0.002	0.008	0.002	0.006	4.342	1.090
2PJP	141	0.067	0.179	0.081	0.119	0.069	0.026
2QUX	17	0.629	1.257	0.435	0.621	3.538	2.607
Mean		5.352	4.952	3.706	4.799	15.145	11.797

[a] Structures with less than 10 Å RMSD.
[b] Using potentials calculated without filtering and clustering.

The last row of Table 4 shows the mean value of the 21 best ranks for each method. For example, the average relative rank over all twenty-one complexes for the current scenario with redundant training set was 3.706%, which was remarkably the best mean value as compared with other potentials. Even in terms of the lowest performing

relative rank, the current scenario has the best rank, 27.844% for 1MFQ. This means that native-like structure is always among top 27.844% in the current scenario.

3.3. Potentials for Current Scenario

3.3.1. General. All 720 potentials for the 9 categories (see Figure 1 and 5) of the current scenario (4 bases × 20 canonical amino acid × 9 categories) are plotted in Figure 6 (see Appendix B for details). The abscissa is the category type and ordinate represents calculated contact potential energy.

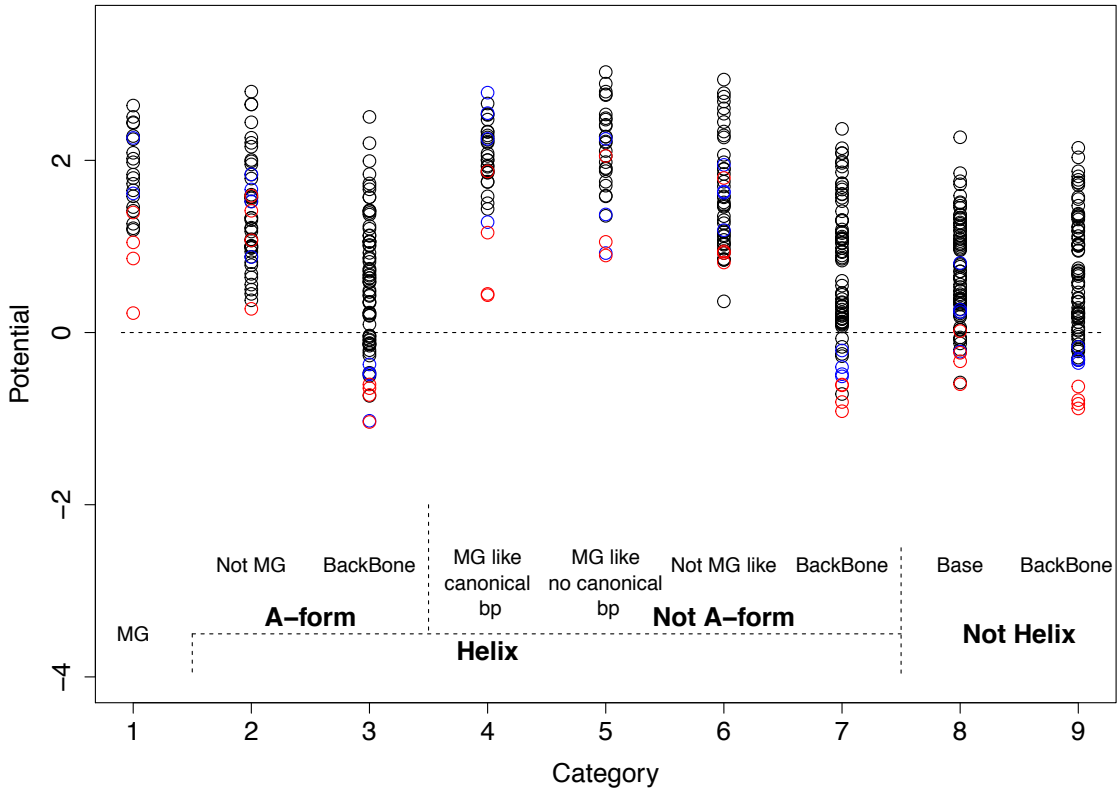

Figure 6. All potentials for current scenario. Abscissa represents category from 1 to 9, and ordinate represents potential energy. Red and blue points represent potentials involving Arg and Lys, respectively. Potentials for missing contacts are not shown.

Each category has 80 potentials (20 canonical amino acid, 4 bases). The strongest interaction involves Arg in Category 3. Categories involving RNA backbone (Categories 3, 7, and 9) have strong interactions (negative potential values), and the strongest

interaction in each category involves Arg (red point). Even in the categories involving the base in RNA (Categories 1, 4, 5, and 8), the strongest interactions involve Arg. For categories involving the base in RNA, only Category 8 has negative potential values. Category 1 has an outlier (the strongest interaction) involving Arg-G. Category 4 has two outliers involving Arg-G and Arg-A.

3.3.2. Strong Interactions. For the current scenario (nine-category set of potentials), 720 potentials were calculated. The strongest interactions from the 20 potentials among all the 720 possible ones for the nine categories are shown in Table 5. The columns represent the rank (1 to 20) out of 720 possibilities, category (1 to 9), pair type of the potential, and potential, respectively from the left. Most of the strongest potentials (eighteen of the twenty) involve RNA backbone (Categories 3, 7, and 9). The strongest interaction involved Arg-cytosine. However, interactions for Arg-cytosine, Arg-uracil, Arg-guanine, and

Table 5. The strongest 20 interactions.

Rank	Category	Pair Type	Potential	Rank	Category	Pair Type	Potential
1	3	ARG_C	-1.039	11	3	ARG_G	-0.646
2	3	LYS_C	-1.025	12	9	ARG_G	-0.627
3	7	ARG_C	-0.914	13	7	ARG_U	-0.613
4	9	ARG_U	-0.882	14	7	ARG_G	-0.604
5	9	ARG_C	-0.828	15	3	ARG_A	-0.603
6	7	ARG_A	-0.807	16	8	ARG_C	-0.600
7	9	ARG_A	-0.786	17	8	ASN_U	-0.581
8	3	TYR_C	-0.736	18	7	LYS_C	-0.509
9	3	ARG_U	-0.725	19	3	HIS_A	-0.503
10	7	TRP_U	-0.714	20	3	LYS_G	-0.485

Arg-adenine are of the similar strength (-0.646 to -1.039). For potentials involving bases, potentials in Category 8 ranked 16th (Arg-cytosine), and 17th (Asn-uracil).

3.3.3. Potentials between Arg and Four Bases. Because Arg is the dominant amino acid residue in the calculated potentials (Table 5), potentials related to Arg were

studied more closely. Figure 7 shows the potentials of Arg in the current scenario and those in the aggregate average. The first four bars represent the potentials for Arg-adenine, Arg-cytosine, Arg-guanine, and

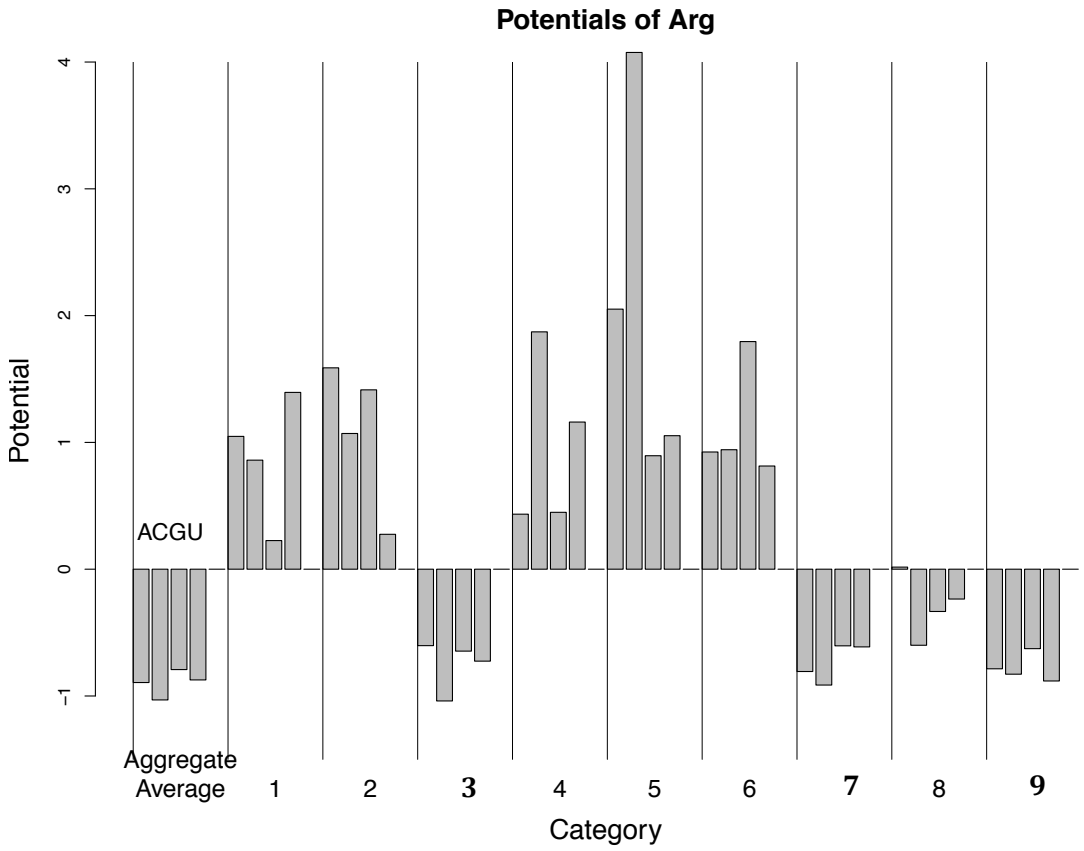

Figure 7. The potentials of Arg for four bases in each class. The bars in each category represent potentials of A, C, G, and U from the left (see the four bars in aggregate average). The potentials from missing data such as Arg-adenine in Category 1 were changed to zero in this plot for convenience. Bold column names indicate categories including RNA backbone.

Arg-uracil, respectively, in aggregate average (an approach with no structure classification). The other numbers labeled as abscissa represent Category 1 to 9.

The three categories involving backbone component of nucleotide (Categories 3, 7, and 9) have similar pattern, but potentials involving base component (Categories 1, 2, 4, 5, 6, and 8) look very different. The largest potential is Arg-cytosine in Category 1.

Potentials for Lysine have slightly different patterns (Figure 8). For example, in Category 2, Arg has preference for uracil but Lys prefers adenine. For both Lys and Arg, strongest interaction is with cytosine in Category 2, and they are of similar strength.

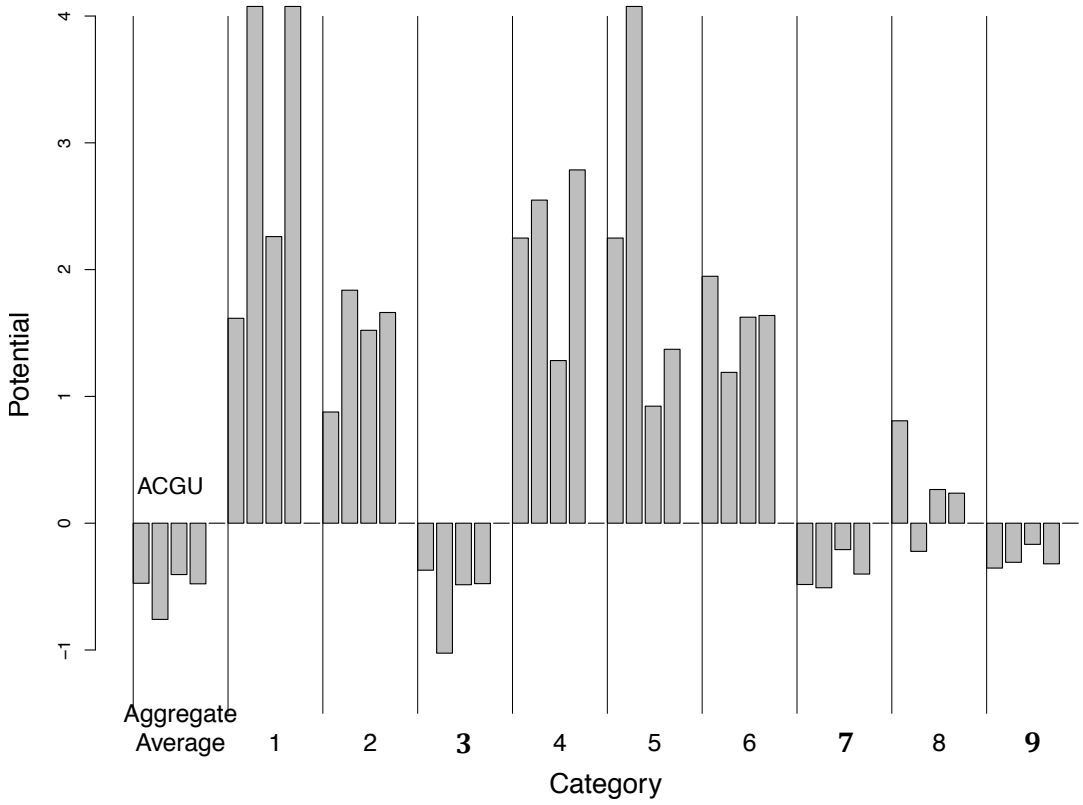

Figure 8. The potentials of Lys for four bases in each class. The bars in each class represent potentials of A, C, G, and U from the left (see the four bars in aggregate average). The potentials from missing data such as Lys-cytosine and Lys-uracil in Category 1 were changed to zero in this plot for convenience. Bold column names indicate categories including RNA backbone.

3.3.4. Potentials for Aggregate Average. The amino acid-nucleotide potentials for aggregate average (without classification by RNA structure) do not utilize RNA

structure classifiers. The potentials for aggregate average are shown in Figure 9.

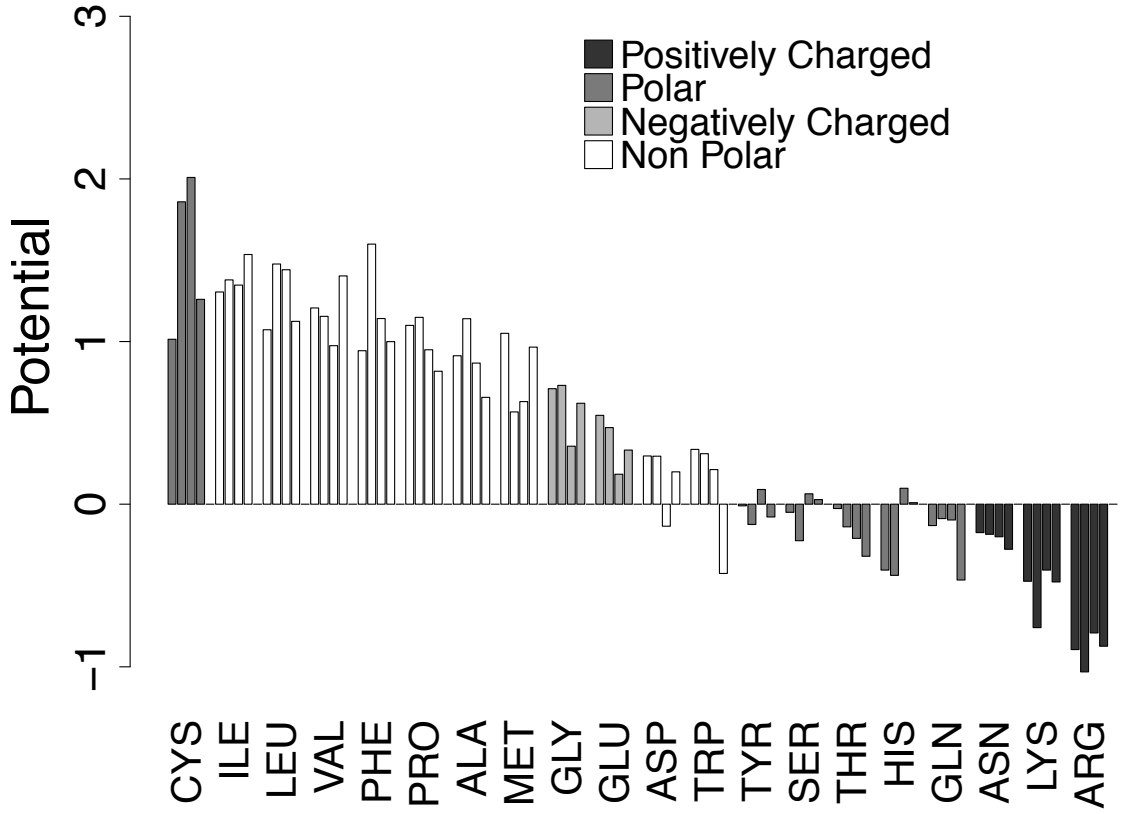

Figure 9. The potentials for aggregate average (potentials without classification by RNA structure). The bars in each class represent potentials of A, C, G, and U from the left. Amino acid is shown in descending order of the average potential for four bases. Background color for a box indicates the characteristic for the amino acid. For example, Arg, Lys, and His are positively charged residues and these boxes are painted in black.

Amino acids with at least one negative potential are Asp, Trp, Tyr, Ser, Thr, His, Gln, Asn, Lys, and Arg. These potentials can be clustered into five groups in the order of strong interactions: positively charged side chains (Arg, Lys), strongly polar side chains (Asn, Gln, His), polar side chains (Ser, Thr, Trp, and Tyr), and negatively charged side chains (Asp, Glu), and hydrophobic side chains. His in certain environments can be positively charged.[39]

3.4. Success Rate for the Current Method

Success rate is calculated among all twenty-one-test complexes to evaluate potentials. For instance, success rate is calculated for whether the native-like structure is identified as belonging in the threshold demarcated set of energy/structures. The success rates of the current, current redundant, and aggregate average are shown in the Figure 10.

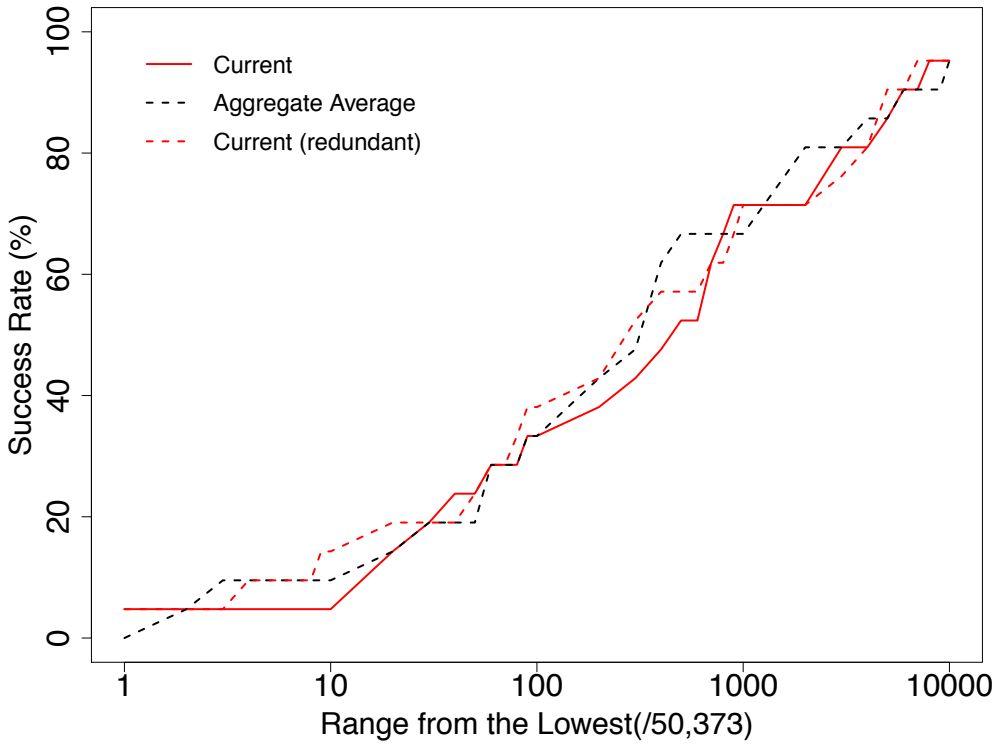

Figure 10. Success rates of current scenario (red line) and other methods (Perez-Cano (black dotted line), QUASI-RNP (blue dotted line), and DARS-RNP (red dotted line). The abscissa represents the increasing thresholds starting with the 1 lowest energy structure. The ordinate represents success rate at the threshold.

3.5. Success Rates for Other Scenarios

Success rates for current and QUASI-RNP, DARS-RNP, and Perez-Cano are compared in Figure 11A. The current scenario potentials compete well with others. Except for the middle range, current scenario performs best among the four sets of

potentials and achieves a 94.5% success rate at the threshold value 1000, which is best among the four potentials.

Comparison of the success rate from the aggregate average and QUASI-RNP, DARS-RNP, and Perez-Cano are shown in Figure 11B. All four approaches adopt reduced representation for atom types. The success rate for the aggregate average (black dotted line) is better than others except for the thresholds 50 to 200, and 600 to 3,000. At the threshold 10,000, the current scenario and aggregate average approach indicated a success rate of 95.24% where DARS-RNP and QUASI-RNP remained at 85.71%, and Perez-Cano remained at 80.95%. Our current scenario and aggregate average show at least comparable overall success rates.

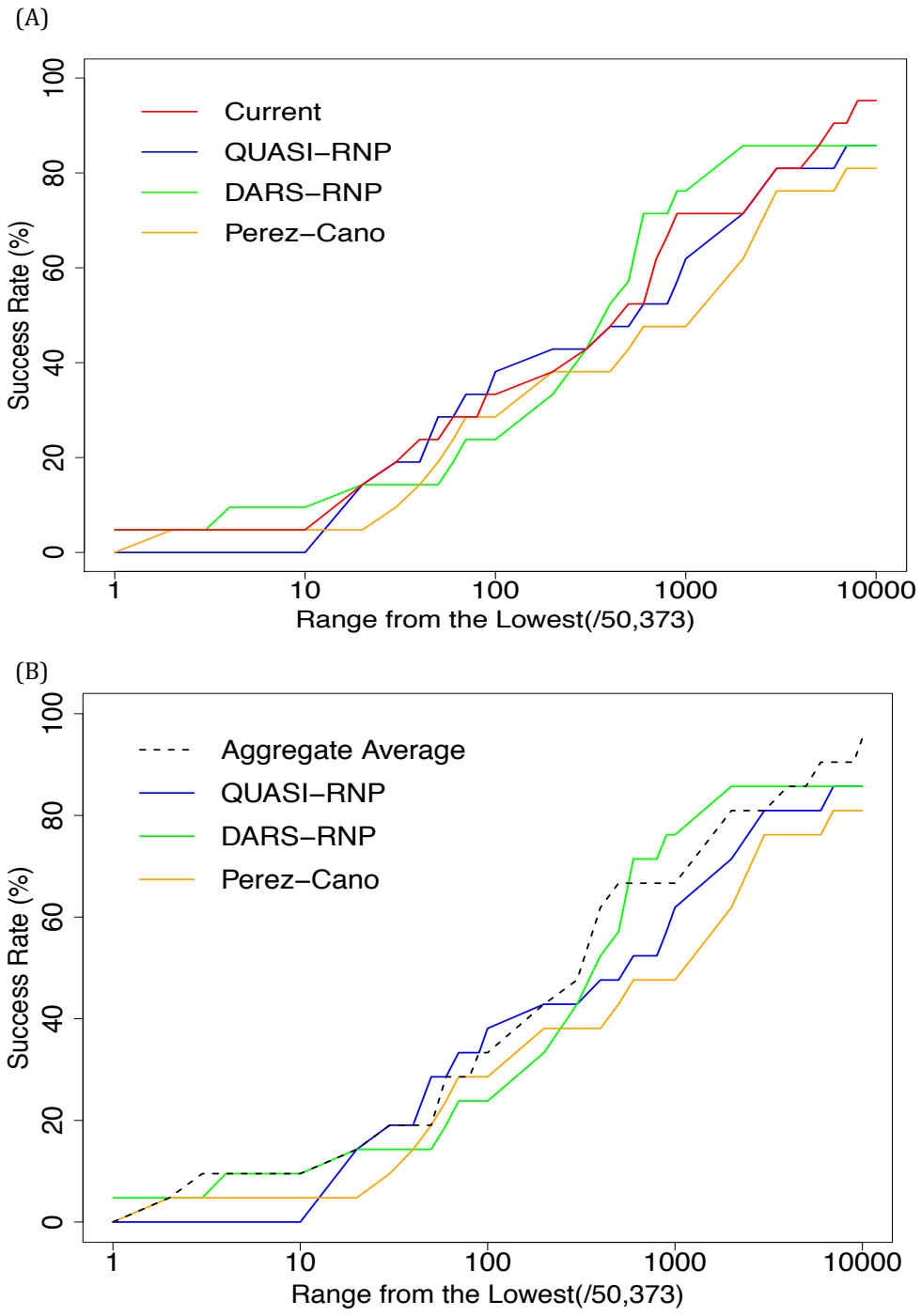

Figure 11. Plots of success rate as a function of threshold value. The abscissa represents the increasing thresholds starting with the 1 lowest energy structure and the ordinate represents success rate at the threshold. (A) Success rates of current scenario and others (Perez-Cano, DARS-RNP, QUASI-RNP), and (B) success rates of aggregate average and others (Perez-Cano, DARS-RNP, QUASI-RNP). The range is only from 1 to 10,000 among 50,373. A threshold 10,000, the right end, corresponds to 20% of the lowest energy structures.

3.6. Score versus RMSD Analysis

3.6.1. Analysis Guidelines. Investigating the discriminating ability of the scoring function or potentials by plotting score versus RMSD is typical.[32-34] Robertson and Varani (2007) employed five complexes (1CVJ, 1FXL, 1URN, 1EC6, and 1JID) as a bound test set, and generated poses with Rosetta (Chen et al., 2004). Also, they used twenty-one complexes and generated poses by MD (molecular dynamics). The docking coordinates are for bound chains, so both the RNA and protein chain belong to the same PDB entry. They examined the plot of score versus RMSD for poses of those five complexes. The R-squared value and Z-score were calculated only for the near native poses (RMSD < 3.0 Å). They compared their distance-dependent potentials and number of contacts with Coulomb potentials and the Rosetta HB potentials,[40] as well as the AMBER potentials.[41] R-squared values for the five bound complexes ranged from 0.15 to 0.46 (0.41 for 1CVJ, 0.20 for 1FXL, 0.15 for URN, 0.46 for 1EC6, 0.17 for 1JID, with p-values less than 0.05).

Tuszynska and Bujnicki (2011) tested the same poses as those in Chen et al. (2004) for five complexes (1CVJ, 1FXL, 1URN, 1EC6, and 1JID) and tested here are additional poses generated by GRAMM for eight pairs of unbound RNA and protein and five RNA-protein complexes. They compared their potentials (QUASI-RNP and DARS-RNP) with Varani potentials[40] and Perez-Cano potentials.[30] Also they analyzed the unbound docking set for eight complexes (3BSO, 1WPU, 2JEA, 1E7K, 2PXV, 1LNG, 2R8S, and 2RKJ) and bound docking set for five complexes (1CVJ, 1FXL, 1URN, 1EC6, and 1JID) using GRAMM for docking. They calculated correlation coefficients for poses with three different RMSD thresholds (5 Å, 10 Å, and 20 Å threshold).

Overall, the mean of the five correlation coefficients for DARS-RNP and QUASI-RNP was comparable to those involving the Perez-Cano potential. For example, the mean correlation coefficients for the unbound test set ranged from 0.23 to 0.37, while those for Varani and Perez-Cano ranged from -0.04 to 0.06 (at 10 Å threshold, p-value is greater than 0.05 in 2RKJ, 2R8S, 1LNG, and 2PXV).

Huang and Zou (2014) also used the bound docking set for the five Chen complexes.[40] Score versus RMSD plots were examined, and correlation coefficients for near-native poses were calculated with three different RMSD thresholds (5 Å, 10 Å, and 20 Å threshold). They also tested three other docking sets. They compared their potential ITScore-PR with DARS-RNP, QUASI-RNP, and dRNA.[31] Overall, for the five Chen complexes, at the RMSD threshold is 5 Å, the correlation coefficient for ITScore-PR was better (0.86) than others (dRNA 0.81, DARS-RNP 0.81, QUASI-RNP 0.80) with a p-value less than 0.05.

3.6.2. Analysis in the Current Scenario. Shown in Figure 12 are examples of three patterns involving plots of RMSD (abscissa) versus score in kcal/mol (ordinate). The rest of the plots are shown in Appendix C. Each pose as calculated by FTDock can be binary classified, with RMSD value less than 10 Å (red circle) or RMSD value greater than or equal to10 Å (black circle). Slopes and p-values for the regression lines are shown in Table 6, and each RNA-protein complex is classified into three types depending on the combination of the sign of the two slopes for the two groups. Type Ⅰ has two positive slopes for regression lines. Type Ⅱ has a positive slope for the low RMSD sets but is negative for the high

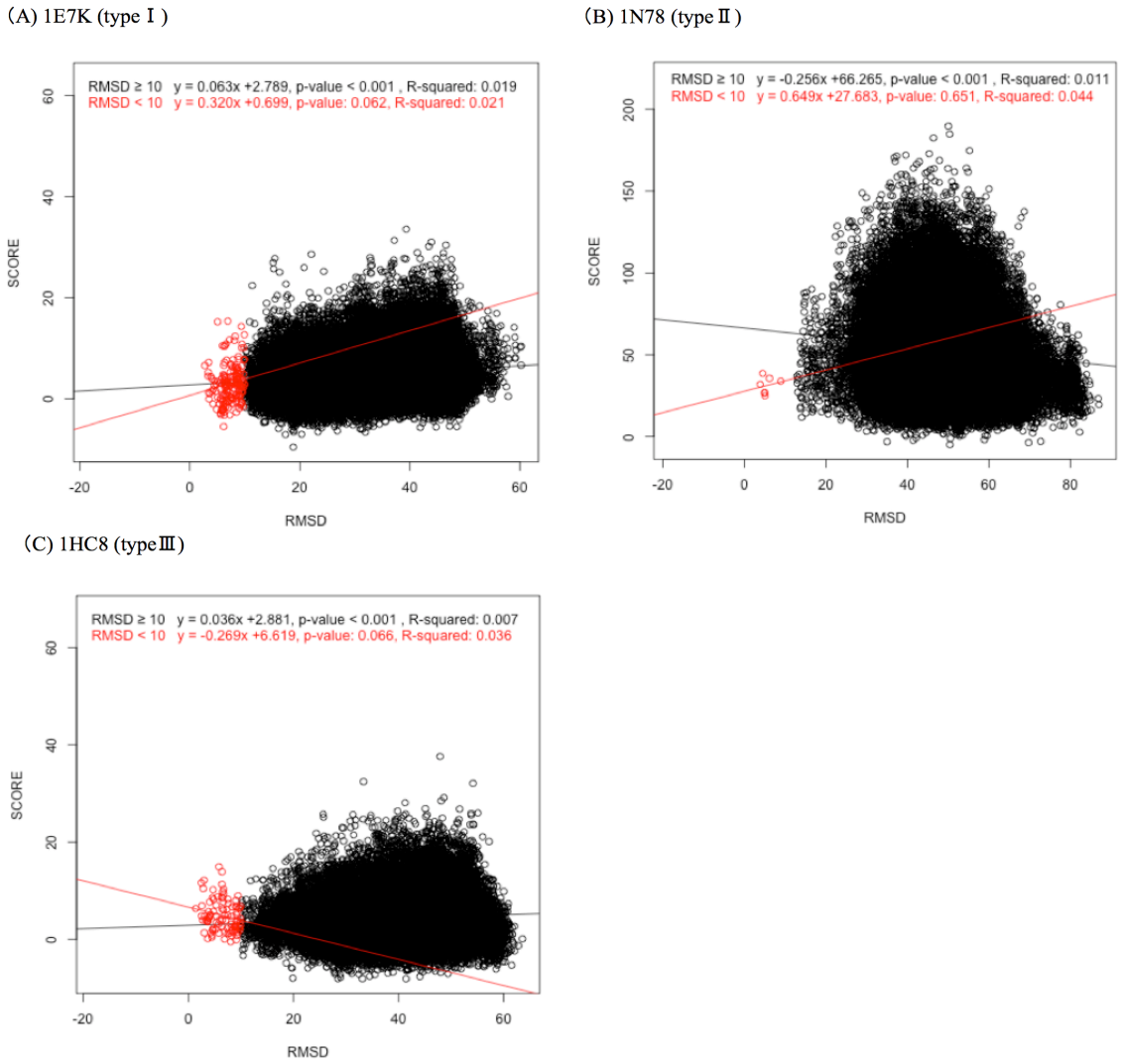

Figure 12. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario. (A) 1E7K (y=0.320x+0.699, R^2=0.021, p-value=0.062; y=0.063x+2.789, R^2=0.019, p-value<0.001), (B) 1N78 (y=0.649x+27.683, R^2=0.044, p-value=0.651; y=-0.256x+66.265, R^2=0.011, p-value<0.001), (C) 1HC8 (y=-0.269x+6.619, R^2=0.036, p-value<0.066; y=0.036x+2.881, R^2=0.007, p-value<0.001), which are the examples for type Ⅰ, Ⅱ, and Ⅲ, respectively. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or are more than 10 Å. The slopes and p-values for the regression lines for the red and black circles are shown above the scatter plots.

RMSD ones. Type Ⅲ has a negative slope for the low RMSD sets. For example, if both of the low and high RMSD sets have positive slopes for the regression lines, the complex is type Ⅰ (Figure 12). The current scenario set has ten complexes for type Ⅰ, three

complex for type Ⅱ, and seven complexes for type Ⅲ. Note that 1KOG cannot be assigned to any type because it has only one pose for low RMSD.

Current		RMSD < 10 Å				RMSD ≥ 10 Å			Type
		Number of Poses	Slope	p-value	R^2	Slope	p-value	R^2	
1	1E7K	22	0.320	0.062	0.021	0.063	< 0.001	0.019	Ⅰ
2	1EC6	200	0.048	0.870	< 0.001	-0.041	< 0.001	0.001	Ⅱ
3	1F7U	5	1.234	0.072	0.712	0.027	< 0.001	0.007	Ⅰ
4	1HC8	95	-0.269	0.066	0.036	0.036	< 0.001	0.007	Ⅲ
5	1JBR	74	-0.016	0.870	< 0.001	0.084	< 0.001	0.051	Ⅲ
6	1K8W	4	-6.605	0.813	0.035	-0.288	< 0.001	0.014	Ⅲ
7	1KOG	1	-	-	-	0.438	< 0.001	0.040	-
8	1LNG	28	-0.250	0.397	0.028	0.063	< 0.001	0.029	Ⅲ
9	1M8W	278	0.207	< 0.001	0.049	0.029	< 0.001	0.015	Ⅰ
10	1MFQ	3	-0.611	0.923	0.014	0.028	< 0.001	0.011	Ⅲ
11	1N78	7	0.649	0.651	0.044	-0.256	< 0.001	0.011	Ⅱ
12	1U0B	6	0.360	0.824	0.014	0.024	< 0.001	0.005	Ⅰ
13	1U63	48	-0.781	0.019	0.113	0.045	< 0.001	0.015	Ⅲ
14	1WPU	222	0.098	0.095	0.013	<0.001	0.970	< 0.001	Ⅰ
15	1WSU	465	0.164	0.239	0.003	0.017	< 0.001	0.008	Ⅰ
16	2BTE	14	6.348	< 0.001	0.765	0.007	< 0.001	0.001	Ⅰ
17	2FMT	9	-1.736	0.340	0.130	0.022	< 0.001	0.008	Ⅲ
18	2HW8	36	0.103	0.854	< 0.001	0.109	< 0.001	0.061	Ⅰ
19	2JEA	642	0.152	0.079	0.005	-0.018	< 0.001	0.004	Ⅱ
20	2PJP	141	0.737	0.039	0.030	0.117	< 0.001	0.096	Ⅰ
21	2QUX	17	1.017	0.004	0.431	0.052	< 0.001	0.011	Ⅰ

Table 6. Summary of regression lines for current scenario. Poses are separated into two groups, poses: the poses whose RMSD are less than 10 Å, and the poses whose RMSD values are greater than or equal to 10 Å. Regression line is drawn for each of the two groups, and the slope and the p-value for both of the regression lines are determined. Column 2 is the PDB entry for the twenty-one test complexes. Column 3 is the number of poses whose RMSD value is less than 10 Å, and Column 4 is the slope for the regression line. Column 5 is the p-value for the regression line. Columns 6 and 7 are the slope and p-value for the regression line drawn on the poses whose RMSD values are greater than or equal to 10 Å.

Type Ⅰ (ten complexes), identified out of assigned twenty RNA-protein complexes, is consistent with reasonably distributed decoys.[33] The slope for low RMSD is greater than 0.01, and the one for high RMSD is positive and smaller than the one for the low RMSD (except 2HW8). The most favorable case is the 2BTE whose slope for

low RMSD sets are 6.348, highest among all the slopes, with a p-value less than 0.001. Note that four among the ten complexes have p-values less than 0.05 (see Table 6). All R-squared values are less than 0.10 for the high RMSD sets. Low RMSD sets have slightly larger R-squared values (three complexes has R-squared values more than 0.1), and notably 2BTE has the largest R-squared value (0.765). Type Ⅰ is assumed preferable to other types.

Type Ⅱ has three complexes (1EC6, 1N78 and 2JEA), and has a negative slope for high RMSD sets and a positive one for low RMSD ones. Even though the slope for low RMSD sets is positive, this type is problematic. The p-value for the high RMSD sets is less than 0.001 in the three complexes.

TypeⅢ(seven complexes) has a negative slope for low RMSD sets. Especially, 1K8W have very steep regression lines (-6.605). R-squared values for high RMSD sets are all smaller than 0.06, and those for low RMSD ones are relatively large. All the p-values for low RMSD sets are larger than 0.05, and all the high RMSD sets have a small p-values less than 0.05 (except 1U63). Note that all complexes have a small number (<100) of poses for low RMSD sets.

3.6.3. Score versus RMSD Analysis. The analysis of energy score versus RMSD with regression lines for both the current showed that about half of the complexes are problematic because the slopes for the regression lines are almost zero or negative. This is in part due to the limited sampling by FTDock that includes rigid molecules in docking and no energy optimization. For instance, in the analysis for current scenario (Table 6), when the number of the native-like structures (RMSD < 10 Å) is greater than 100, the slope is always positive (1EC6, 1M8W, 1WPU, 1WSU, 2JEA, and 2PJP).

However, the slope can be negative when the number of the native-like structures is less than 100. An example of a negative slope for the low RMSD group can be seen in the literature, too.[33] For example, the regression line of poses for 1EC6 is almost a flat slope despite the fact that the docking pose was generated by ROSETTA which should provide a better sampling of poses.

4. DISCUSSION

4.1. Interactions for Arginine

Category 1 (A-form helix, major groove) has a strong preference for Arg-guanine, and Category 4 (not A-form helix, major groove, canonical base pair) also has a strong preference for Arg-guanine and Arg-adenine. The strong interactions of Arg-guanine are likely due to the ionic effect between the positively charged amino group of Arg and the partial negative charge on nitrogen or oxygen atoms of guanine and adenine.[42] However, no significant preference was indicated for categories 3, 7, and 9. Moreover, because Arg has an ability to make hydrogen-bonding networks with bases, phosphates, and sugars, it makes contacts with not only backbones but also with base edges.[43] Bidentate interactions for Arg-guanine also augment the potential.[44] Luscombe et al. observed four types of bidentate interactions between guanine and Arg. In addition, the longer side chain of Arg is suitable for the deep and narrow major groove for the A-form helix of RNA. Our results are consistent with the notion that guanine in the A-form helix for RNA has uniquely strong interactions with Arg.[29]

4.2. Comparison with Cutting-edge Density Function Potentials

A new scoring function called ITScore-PR was recently developed using a density function approach calculated from statistical mechanics.[32] It calculates all-atom and

distance-dependent atomic interaction potentials. The unique feature of the scoring function is that the potentials function is derived from the distribution function refined with poses generated by ZDock. Courtesy of Dr. Zou, we tested ITScore-PR[32] on our twenty-one test complexes, and its mean best rank in percentile on our twenty-one test complexes was 2.430%, which is better than our current scenario (5.352%). ITScore-PR defines atom types with reduced representation. For example, an alpha carbon of any amino acid is represented by "C3A," and "NZ" of Lys is represented as "N3+." Note that the alpha carbon is included in the contacts in their training set, which means their potential includes van der Waals force. The distance-dependent potentials are calculated using poses generated by ZDOCK in addition to 175 RNA-protein native complex structures. The number of classifications for ITScore-PR is 14,400 (12 RNA atom types, 20 protein atom types, and 60 bins for distance). However, their evaluation of the test set is more limited in its grouping of sets of atoms.

4.3. Redundancy in a Training Set

Current scenario without filtering and clustering had a better mean rank (3.706%) than current scenario (5.352%). The redundant training set (267,330 contacts) was more than fifteen times larger than the non-redundant training set (16,031 contacts). This result suggests that clustering and filtering allow a larger set of non-contact potentials and therefore are not allowing any relative scaling of those potentials with respect to amino acid type. However, preliminary results indicate that if we apply the 70% protein identity filter used in Perez-Cano, our ranking results for current scenario exceed those for Perez-Cano.

4.4. Limitations of FTDock and All-atom Potentials

The score versus RMSD analysis showed that seven of twenty-one test set complexes had negative slopes for regression lines in native-like structures (RMSD < 10 Å), indicating limited sampling by FTDock. Therefore, the comparison with other approaches may have drawbacks. This may come from the restricted ability for FTDock to generate realistic alternative poses. This problem in samplinig may be overcome by using a more sophisticated docking program such as Rosetta which adopts hierarchical modeling, flexible back bone docking and optimizing side chains.[33] However, when we apply our most recent all-atom potential to Rosetta,[20] we will not be able to use the potential in the first sampling step, the crucial step for good prediction, because the first step is too coarse-grained. This issue also may apply to using ITScore-PR.

5. CONCLUSIONS

This project has successfully developed pairwise nucleotide-amino acid potentials for protein-RNA binding and quantitatively analyzed interactions between RNA and protein, noting the following:

- The analysis of statistical potentials confirmed the strong preference for Arg-guanine in the major groove of A-form helices.
- Moreover, the dominant Arg interactions are involved in the backbone.
- We introduced a classifier of RNA structures, and it improved the prediction when we allowed redundancy and increased the size of the training set.
- Comparison with other approaches such as ITScore-PR may not be adequate because of the limited capability of FTDock to generate native-like binding poses.

6. FUTURE STUDIES

Continuation of these methods should include the following:

- Use docking programs such as Rosetta that include flexible docking and side chain repacking to fully evaluate our potentials.
- Incorporate our potentials into Rosetta (aggregate average for the first step and suitable all-atom potential to the second step).
- Explore the additional arguments such as distance and bonding angles to our potentials and evaluate them.
- Explore other potentials including van der Waals forces.
- Group together OP1 and OP2 (and NH1 and NH2), and recalculate potentials.
- In addition, better scaling for potentials with no actual hydrogen bonds identified should be explored.

REFERENCES

(1) Liu, B.; Diamond, J. M.; Mathews, D. H.; Turner, D. H. Fluorescence competition and optical melting measurements of RNA three-way multibranch loops provide a revised model for thermodynamic parameters. *Biochemistry* **2011**, *50*, 640-653.

(2) Cereda, M.; Pozzoli, U.; Rot, G.; Juvan, P.; Schweitzer, A.; Clark, T.; Ule, J. RNA motifs: prediction of multivalent RNA motifs that control alternative splicing. *Genome Biol*. **2014**, *15*, R20-2014-15-1-r20.

(3) Wang, Y. X.; Zuo, X.; Wang, J.; Yu, P.; Butcher, S. E. Rapid global structure determination of large RNA and RNA complexes using NMR and small-angle X-ray scattering. *Methods* **2010**, *52*, 180-191.

(4) Flores, J. K.; Kariawasam, R.; Gimenez, A. X.; Helder, S.; Cubeddu, L.; Gamsjaeger, R.; Ataide, S. F. Biophysical characterisation and quantification of nucleic acid-protein interactions: EMSA, MST and SPR. *Curr. Protein Pept. Sci.* **2015**, *16*, 727-734.

(5) Ponchon, L.; Catala, M.; Seijo, B.; El Khouri, M.; Dardel, F.; Nonin-Lecomte, S.; Tisne, C. Co-expression of RNA-protein complexes in Escherichia coli and applications to RNA biology. *Nucleic Acids Res*. **2013**, *41*, e150.

(6) Shi, X.; Huang, L.; Lilley, D. M.; Harbury, P. B.; Herschlag, D. The solution structural ensembles of RNA kink-turn motifs and their protein complexes. *Nat. Chem. Biol.* **2016**, *12*, 146-152.

(7) Gong, Z.; Schwieters, C. D.; Tang, C. Conjoined use of EM and NMR in RNA structure refinement. *PLoS One* **2015**, *10*, e0120445.

(8) Kligun, E.; Mandel-Gutfreund, Y. The role of RNA conformation in RNA-protein recognition. *RNA Biol*. **2015**, *12*, 720-727.

(9) Low, J. T.; Weeks, K. M. SHAPE-directed RNA secondary structure prediction. *Methods* **2010**, *52*, 150-158.

(10) Cao, S.; Chen, S. J. Physics-based de novo prediction of RNA 3D structures. J. *Phys. Chem. B* **2011**, *115*, 4216-4226.

(11) Kerpedjiev, P.; Honer Zu Siederdissen, C.; Hofacker, I. L. Predicting RNA 3D structure using a coarse-grain helix-centered model. *RNA* **2015**, *21*, 1110-1121.

(12) Lorenz, R.; Wolfinger, M. T.; Tanzer, A.; Hofacker, I. L. Predicting RNA secondary structures from sequence and probing data. *Methods* **2016***, 103*, 86-98.

(13) Biesiada, M.; Pachulska-Wieczorek, K.; Adamiak, R. W.; Purzycka, K. J. RNAComposer and RNA 3D structure prediction for nanotechnology. *Methods* **2016**, *103*, 120-127.

(14) Boudard, M.; Bernauer, J.; Barth, D.; Cohen, J.; Denise, A. GARN: Sampling RNA 3D structure space with game theory and knowledge-based scoring strategies. *PLoS One* **2015**, *10*, e0136444.

(15) Pusey, M. L.; Liu, Z. J.; Tempel, W.; Praissman, J.; Lin, D.; Wang, B. C.; Gavira, J. A.; Ng, J. D. Life in the fast lane for protein crystallization and X-ray crystallography. *Prog. Biophys. Mol. Biol.* **2005**, *88*, 359-386.

(16) Swails, J.; Zhu, T.; He, X.; Case, D. A. AFNMR: automated fragmentation quantum mechanical calculation of NMR chemical shifts for biomolecules. *J. Biomol. NMR* **2015**, *63*, 125-139.

(17) Bowie, J. U.; Luthy, R.; Eisenberg, D. A method to identify protein sequences that fold into a known three-dimensional structure. *Science* **1991**, *253*, 164-170.

(18) Baker, D.; Sali, A. Protein structure prediction and structural genomics. *Science* **2001**, *294*, 93-96.

(19) Kelley, L. A.; Sternberg, M. J. Protein structure prediction on the web: a case study using the Phyre server. *Nat. Protoc*. **2009**, *4*, 363-371.

(20) Guilhot-Gaudeffroy, A.; Froidevaux, C.; Aze, J.; Bernauer, J. Protein-RNA complexes and efficient automatic docking: expanding RosettaDock possibilities. *PLoS One* **2014**, *9*, e108928.

(21) Kaufmann, K. W.; Lemmon, G. H.; Deluca, S. L.; Sheehan, J. H.; Meiler, J. Practically useful: what the Rosetta protein modeling suite can do for you. *Biochemistry* **2010**, *49*, 2987-2998.

(22) Dill, K. A.; MacCallum, J. L. The protein-folding problem, 50 years on. *Science* **2012**, *338*, 1042-1046.

(23) Bale, J. B.; Gonen, S.; Liu, Y.; Sheffler, W.; Ellis, D.; Thomas, C.; Cascio, D.; Yeates, T. O.; Gonen, T.; King, N. P.; Baker, D. Accurate design of megadalton-scale two-component icosahedral protein complexes. *Science* **2016**, *353*, 389-394.

(24) Ke, A.; Doudna, J. A. Crystallization of RNA and RNA-protein complexes. *Methods* **2004**, *34*, 408-414.

(25) Cheng, Y. Single-Particle Cryo-EM at Crystallographic Resolution. *Cell* **2015**, *161*, 450-457.

(26) Tovchigrechko, A.; Vakser, I. A. GRAMM-X public web server for protein-protein docking. *Nucleic Acids Res.* **2006**, *34*, W310-4.

(27) Katchalski-Katzir, E.; Shariv, I.; Eisenstein, M.; Friesem, A. A.; Aflalo, C.; Vakser, I. A. Molecular surface recognition: determination of geometric fit between proteins and

their ligands by correlation techniques. *Proc. Natl. Acad. Sci. U. S. A.* **1992**, *89*, 2195-2199.

(28) Gabb, H. A.; Jackson, R. M.; Sternberg, M. J. Modelling protein docking using shape complementarity, electrostatics and biochemical information. *J. Mol. Biol.* **1997***, 272*, 106-120.

(29) Lustig, B.; Arora, S.; Jernigan, R. L. RNA base-amino acid interaction strengths derived from structures and sequences. *Nucleic Acids Res.* **1997**, *25*, 2562-2565.

(30) Perez-Cano, L.; Solernou, A.; Pons, C.; Fernandez-Recio, J. Structural prediction of protein-RNA interaction by computational docking with propensity-based statistical potentials. *Pac. Symp. Biocomput.* **2010**, 293-301.

(31) Wang, J.; Zhao, Y.; Zhu, C.; Xiao, Y. 3dRNAscore: a distance and torsion angle dependent evaluation function of 3D RNA structures. *Nucleic Acids Res*. **2015**, *43*, e63.

(32) Huang, S. Y.; Zou, X. A knowledge-based scoring function for protein-RNA interactions derived from a statistical mechanics-based iterative method. *Nucleic Acids Res*. **2014**, *42*, e55.

(33) Robertson, T. A.; Varani, G. An all-atom, distance-dependent scoring function for the prediction of protein-DNA interactions from structure. *Proteins* **2007**, *66*, 359-374.

(34) Tuszynska, I.; Bujnicki, J. M. DARS-RNP and QUASI-RNP: new statistical potentials for protein-RNA docking. *BMC Bioinformatics* **2011**, *12*, 348-2105-12-348.

(35) Burge, R. G.; Martinez-Yamout, M. A.; Dyson, H. J.; Wright, P. E. Structural characterization of interactions between the double-stranded RNA-binding zinc finger protein JAZ and nucleic acids. *Biochemistry* **2014**, *53*, 1495-1510.

(36) Lu, X. J.; Olson, W. K. 3DNA: a software package for the analysis, rebuilding and visualization of three-dimensional nucleic acid structures. *Nucleic Acids Res*. **2003**, *31*, 5108-5121.

(37) McDonald, I. K.; Thornton, J. M. Satisfying hydrogen bonding potential in proteins. *J. Mol. Biol.* **1994**, *238*, 777-793.

(38) Bailor, M. H.; Mustoe, A. M.; Brooks, C. L.,3rd; Al-Hashimi, H. M. 3D maps of RNA interhelical junctions. *Nat. Protoc*. **2011**, *6*, 1536-1545.

(39) Chi, Y. C.; Armstrong, G. S.; Jones, D. N.; Eisenmesser, E. Z.; Liu, C. W. Residue histidine 50 plays a key role in protecting alpha-synuclein from aggregation at physiological pH. *J. Biol. Chem.* **2014**, *289*, 15474-15481.

(40) Chen, Y.; Kortemme, T.; Robertson, T.; Baker, D.; Varani, G. A new hydrogen-bonding potential for the design of protein-RNA interactions predicts specific contacts and discriminates decoys. *Nucleic Acids Res*. **2004**, *32*, 5147-5162.

(41) Wang, J.; Wolf, R. M.; Caldwell, J. W.; Kollman, P. A.; Case, D. A. Development and testing of a general amber force field. *J. Comput. Chem.* **2004**, *25*, 1157-1174.

(42) Abraham, R. J.; Smith, P. E. Charge calculations in molecular mechanics 6: the calculation of partial atomic charges in nucleic acid bases and the electrostatic contribution to DNA base pairing. *Nucleic Acids Res*. **1988**, *16*, 2639-2657.

(43) Burd, C. G.; Dreyfuss, G. Conserved structures and diversity of functions of RNA-binding proteins. *Science* **1994**, *265*, 615-621.

(44) Luscombe, N. M.; Laskowski, R. A.; Thornton, J. M. Amino acid-base interactions: a three-dimensional analysis of protein-DNA interactions at an atomic level. *Nucleic Acids Res.* **2001**, *29*, 2860-2874.

APPENDICES

Appendix A

Table A. 1. Frequency of contacts for current scenario. The first row indicates categories. Columns 1 and 10 indicate amino acid-nucleotide pair.

	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9
ALA_A	0	0	5	0	0	0	27	6	21	LEU_A	0	0	7	0	1	0	11	13	7
ALA_C	1	0	8	0	0	1	22	4	6	LEU_C	0	0	11	0	0	0	10	13	2
ALA_G	0	1	11	11	0	12	28	10	13	LEU_G	0	1	8	0	0	0	18	18	4
ALA_U	0	0	12	0	0	0	22	7	28	LEU_U	0	0	1	0	0	0	8	7	5
ARG_A	5	2	82	62	4	27	508	113	440	LYS_A	2	7	58	3	3	5	308	31	222
ARG_C	20	14	500	6	0	29	674	161	237	LYS_C	0	4	512	2	0	20	356	89	103
ARG_G	60	8	263	98	46	10	584	129	212	LYS_G	2	7	210	25	46	14	313	49	102
ARG_U	3	20	109	15	18	27	303	104	311	LYS_U	0	2	75	1	11	7	222	49	126
ASN_A	1	3	18	3	4	10	64	43	44	MET_A	0	0	3	0	0	0	5	6	2
ASN_C	1	5	56	7	0	12	65	17	22	MET_C	0	1	9	0	0	2	18	4	4
ASN_G	0	13	52	8	2	28	88	30	21	MET_G	0	5	7	0	0	14	16	2	0
ASN_U	0	6	19	2	1	12	43	104	67	MET_U	0	0	1	0	0	3	6	2	2
ASP_A	0	0	8	0	2	1	13	30	25	PHE_A	0	0	0	0	1	0	2	7	19
ASP_C	1	1	20	4	0	2	18	20	28	PHE_C	0	0	2	0	0	0	2	3	3
ASP_G	0	25	22	0	6	36	35	59	16	PHE_G	0	1	6	0	0	0	2	15	4
ASP_U	0	0	13	0	0	0	19	35	19	PHE_U	0	0	1	0	0	0	2	12	5
CYS_A	0	0	0	0	0	0	4	3	1	PRO_A	0	0	1	0	0	1	13	4	8
CYS_C	0	0	0	1	0	0	1	0	0	PRO_C	0	2	5	0	0	0	10	4	5
CYS_G	0	0	1	0	0	0	1	0	0	PRO_G	3	4	10	3	0	11	8	1	7
CYS_U	0	0	0	0	0	0	0	2	2	PRO_U	0	0	3	0	0	0	16	6	8
GLN_A	2	8	19	1	1	2	39	33	41	SER_A	2	3	17	2	7	8	66	45	88
GLN_C	4	7	54	3	1	13	59	13	31	SER_C	2	13	79	9	1	20	129	44	38
GLN_G	1	21	56	4	1	58	92	19	17	SER_G	0	13	60	4	7	40	100	20	22
GLN_U	0	7	36	0	0	6	42	50	45	SER_U	3	0	14	2	3	6	48	30	54
GLU_A	1	0	10	0	1	0	21	33	19	THR_A	0	1	9	1	0	4	61	58	61
GLU_C	0	1	15	5	0	2	21	31	11	THR_C	0	4	90	0	0	4	83	37	30
GLU_G	0	10	26	3	2	22	34	88	25	THR_G	1	8	44	2	0	20	81	27	39
GLU_U	0	6	12	0	0	1	19	14	23	THR_U	1	1	29	0	3	0	52	20	65
GLY_A	0	2	14	0	3	2	60	18	62	TRP_A	0	0	1	0	0	0	7	2	11
GLY_C	4	2	40	3	0	3	84	17	16	TRP_C	0	0	4	1	0	0	9	5	4
GLY_G	11	27	47	6	0	33	271	13	44	TRP_G	0	3	2	0	0	1	24	3	2
GLY_U	0	0	21	2	0	0	48	36	37	TRP_U	0	0	3	0	1	1	49	1	3
HIS_A	0	2	23	0	1	9	33	10	44	TYR_A	0	3	6	0	3	3	89	65	76
HIS_C	1	3	36	0	0	7	40	22	21	TYR_C	0	1	135	2	2	0	103	9	19
HIS_G	2	4	46	3	7	12	50	14	34	TYR_G	0	3	27	0	2	11	49	19	30
HIS_U	0	2	15	0	1	3	29	22	38	TYR_U	0	1	3	1	0	1	29	14	44
ILE_A	0	0	4	0	0	0	4	25	3	VAL_A	0	0	1	0	0	0	14	14	6
ILE_C	1	0	1	0	0	0	4	7	6	VAL_C	0	0	20	0	0	0	10	6	4
ILE_G	0	0	4	0	0	0	9	7	6	VAL_G	0	3	13	0	0	2	33	12	7
ILE_U	0	0	0	0	0	0	8	10	7	VAL_U	0	0	3	0	0	0	8	3	5

Appendix B

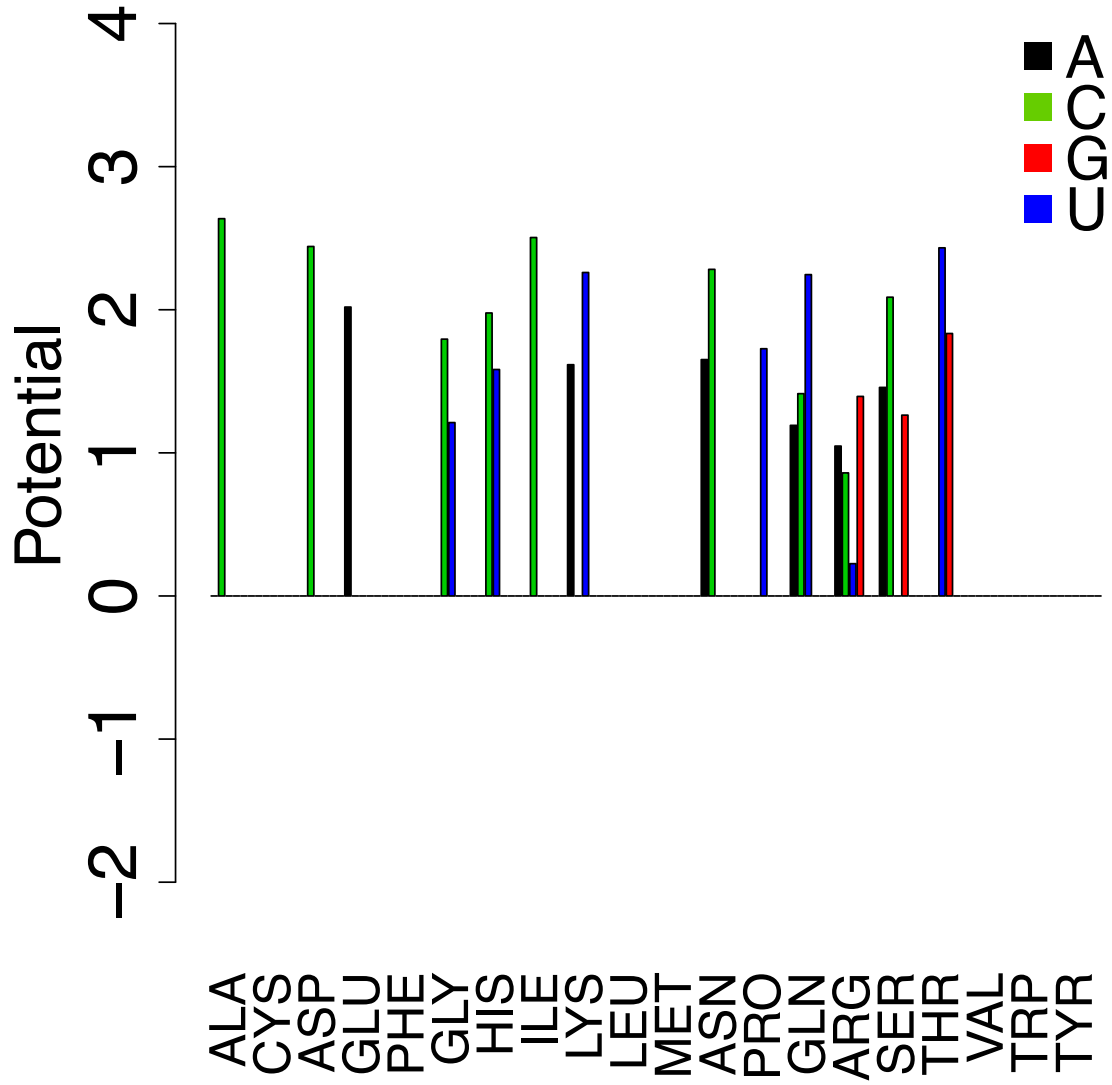

Figure B. 1. Potentials for current scenario Category 1 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

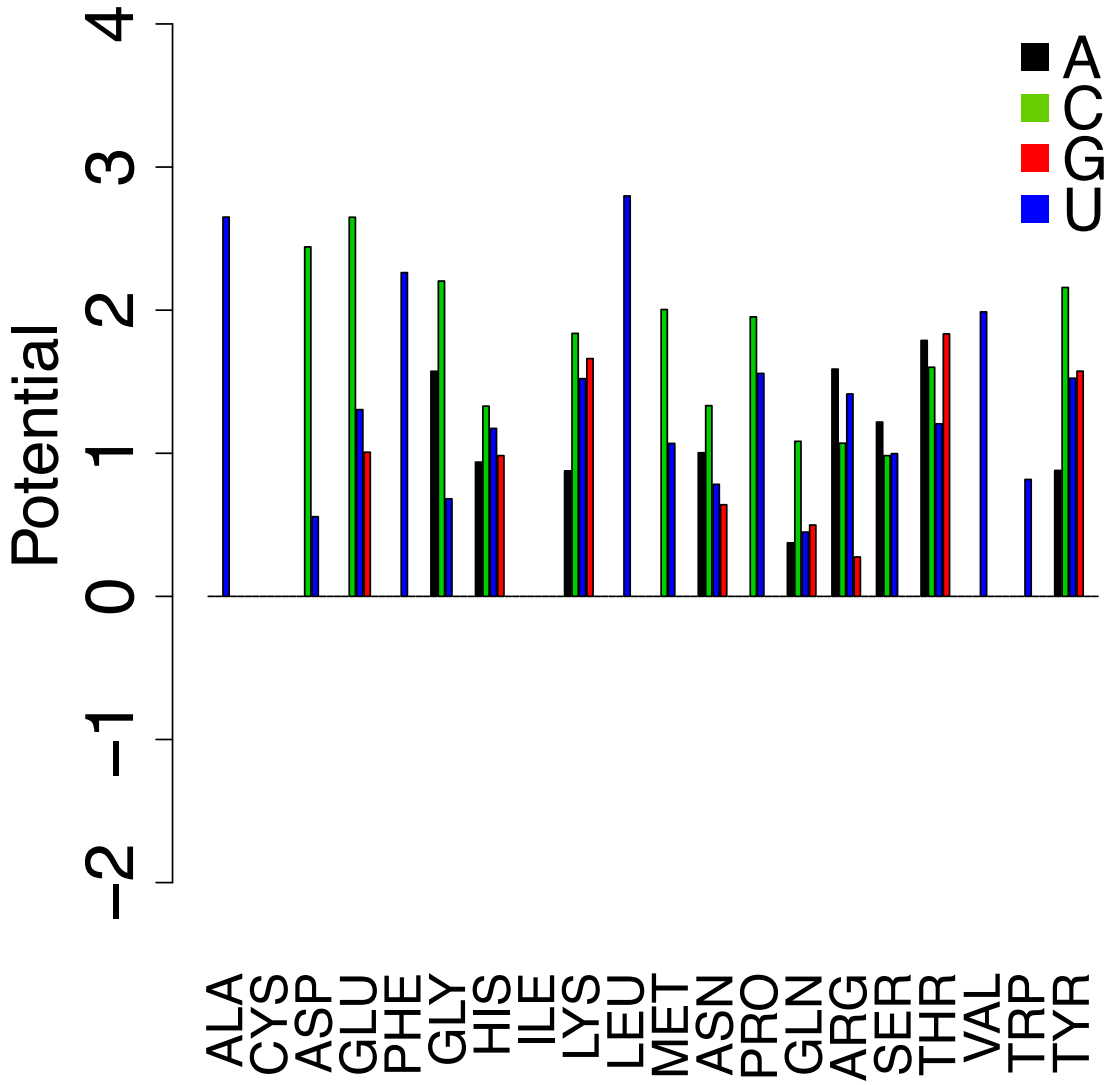

Figure B. 2. Potentials for current scenario Category 2 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

.

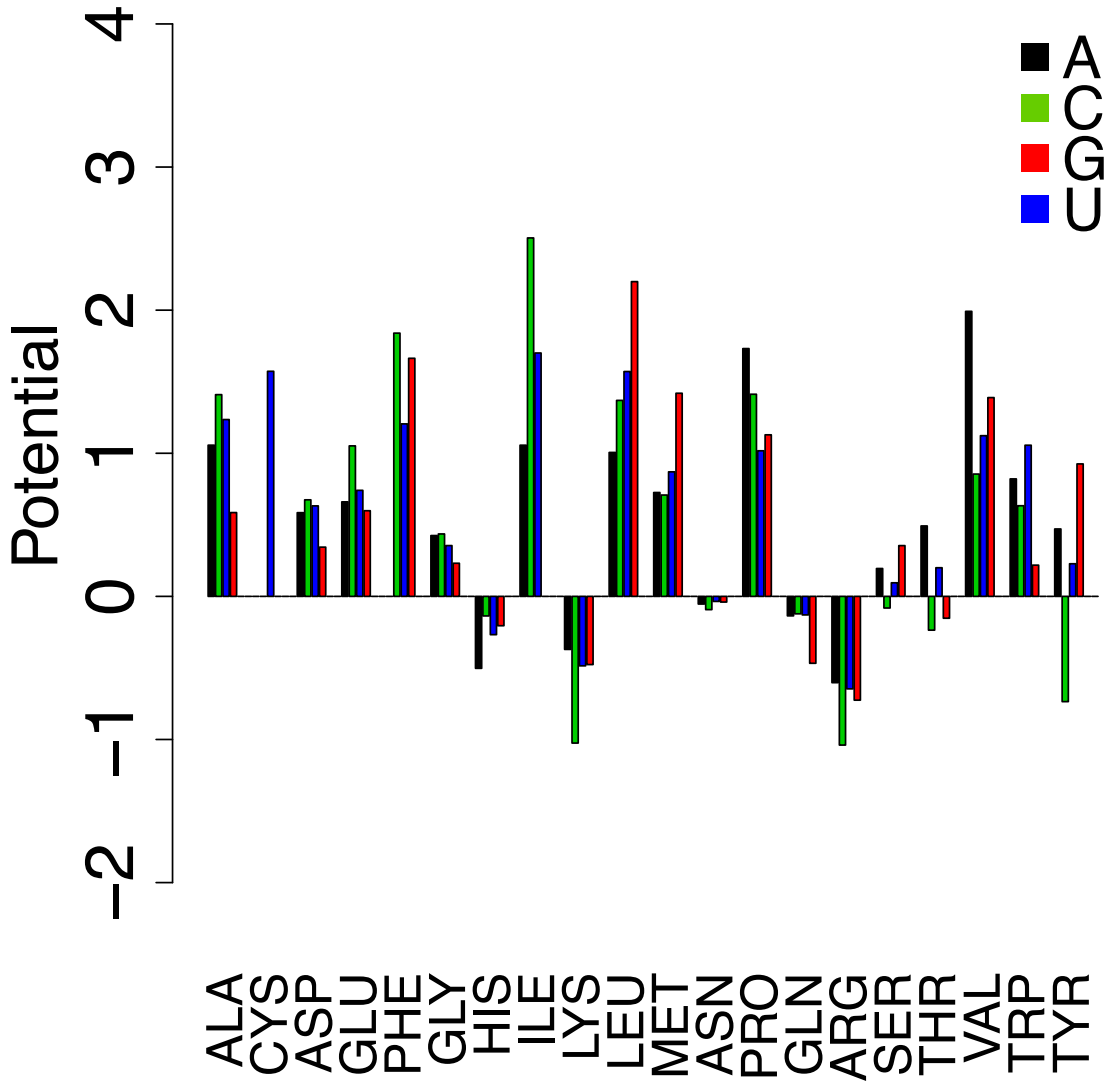

Figure B. 3. Potentials for current scenario Category 3 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

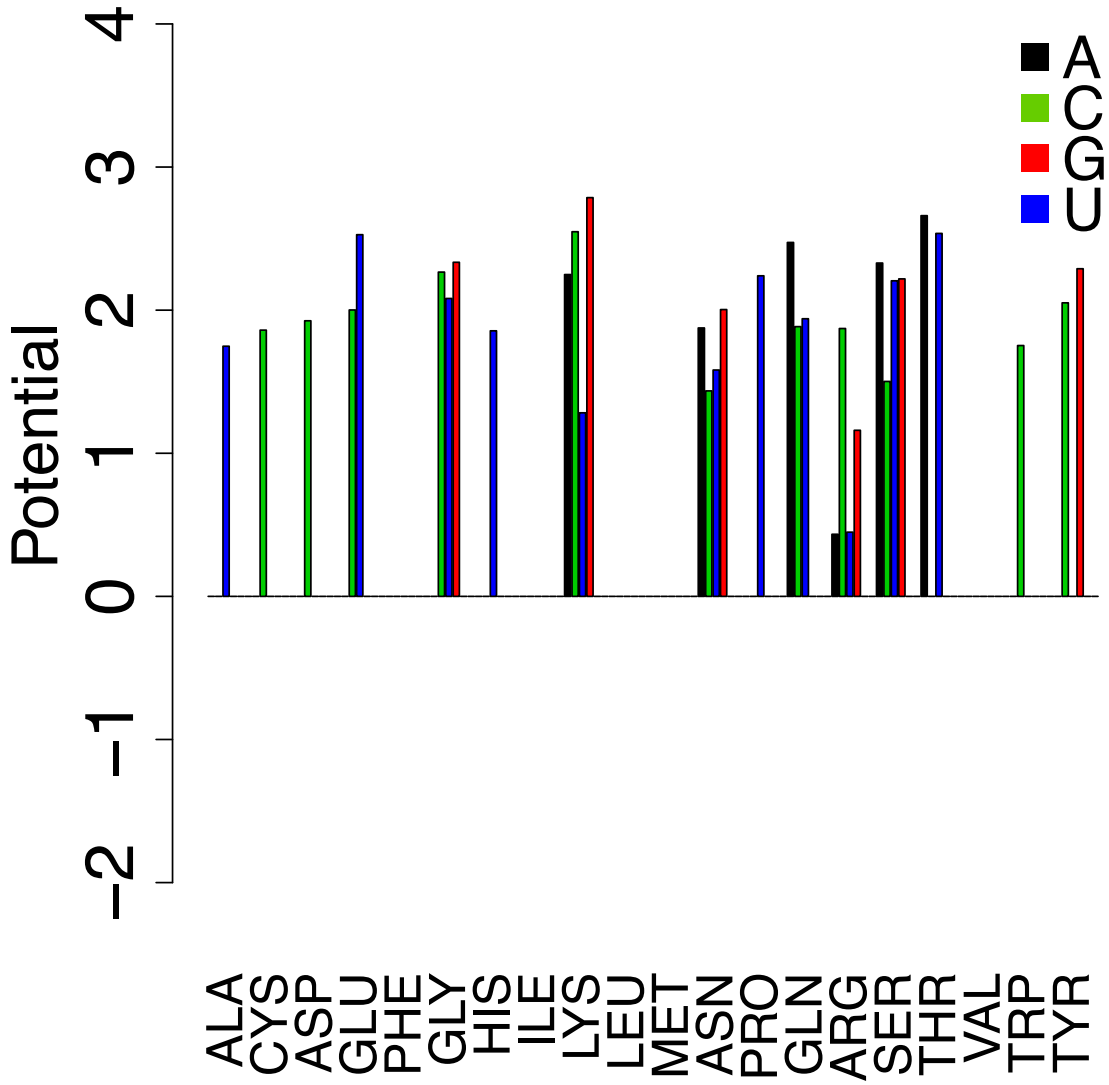

Figure B. 4. Potentials for current scenario Category 4 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

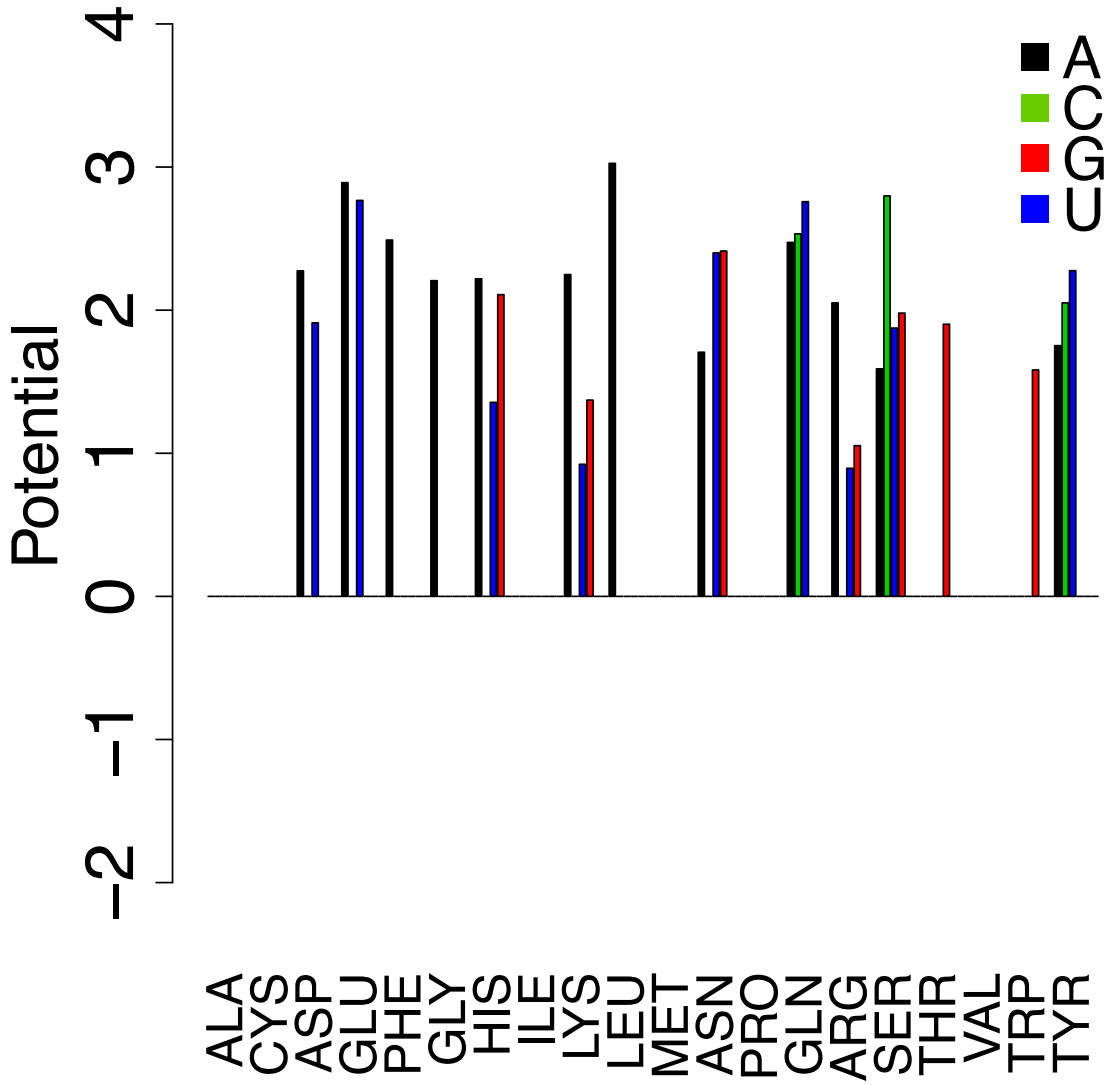

Figure B. 5. Potentials for current scenario Category 5 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

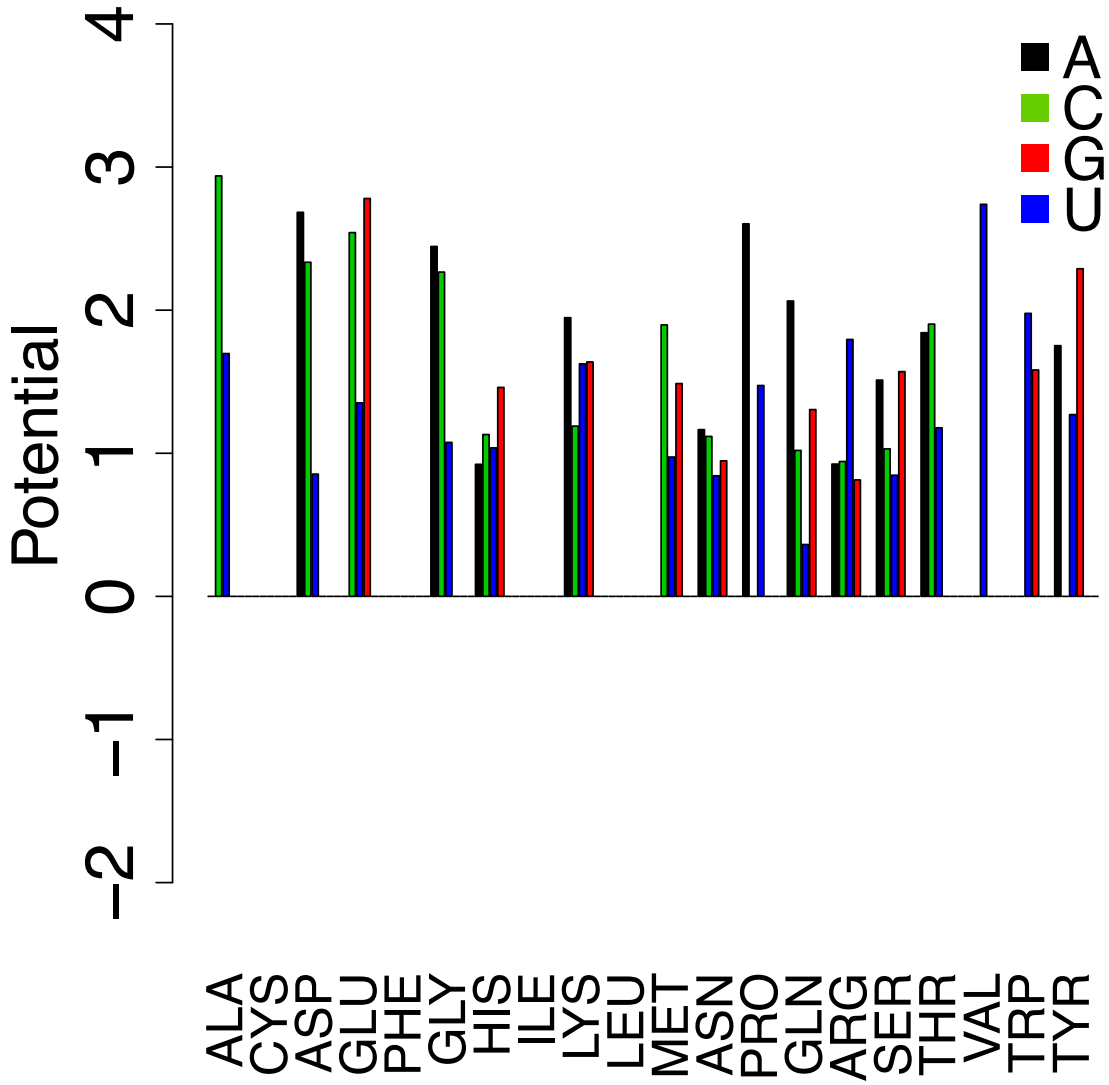

Figure B. 6. Potentials for current scenario Category 6 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

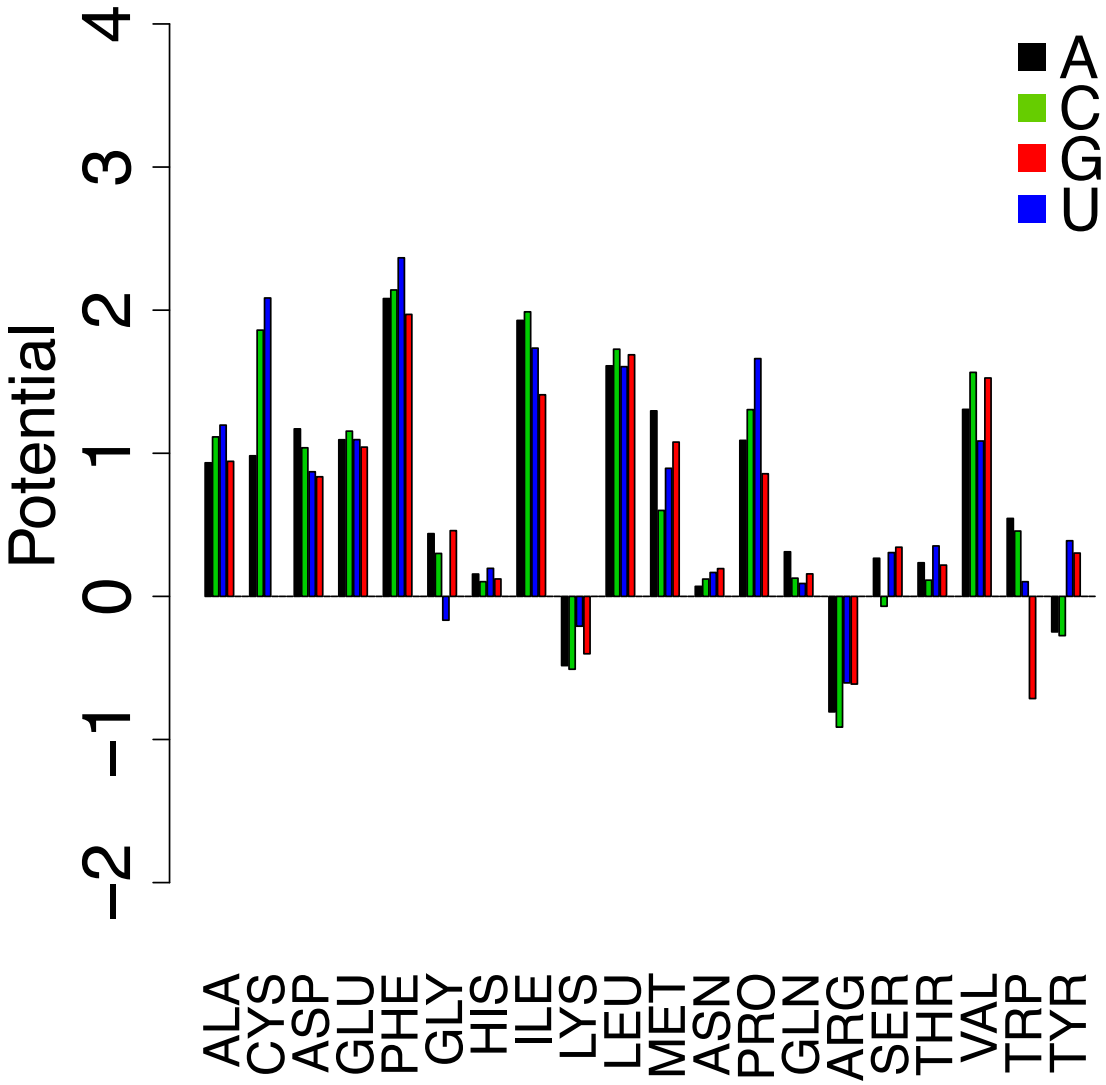

Figure B. 7. Potentials for current scenario Category 7 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

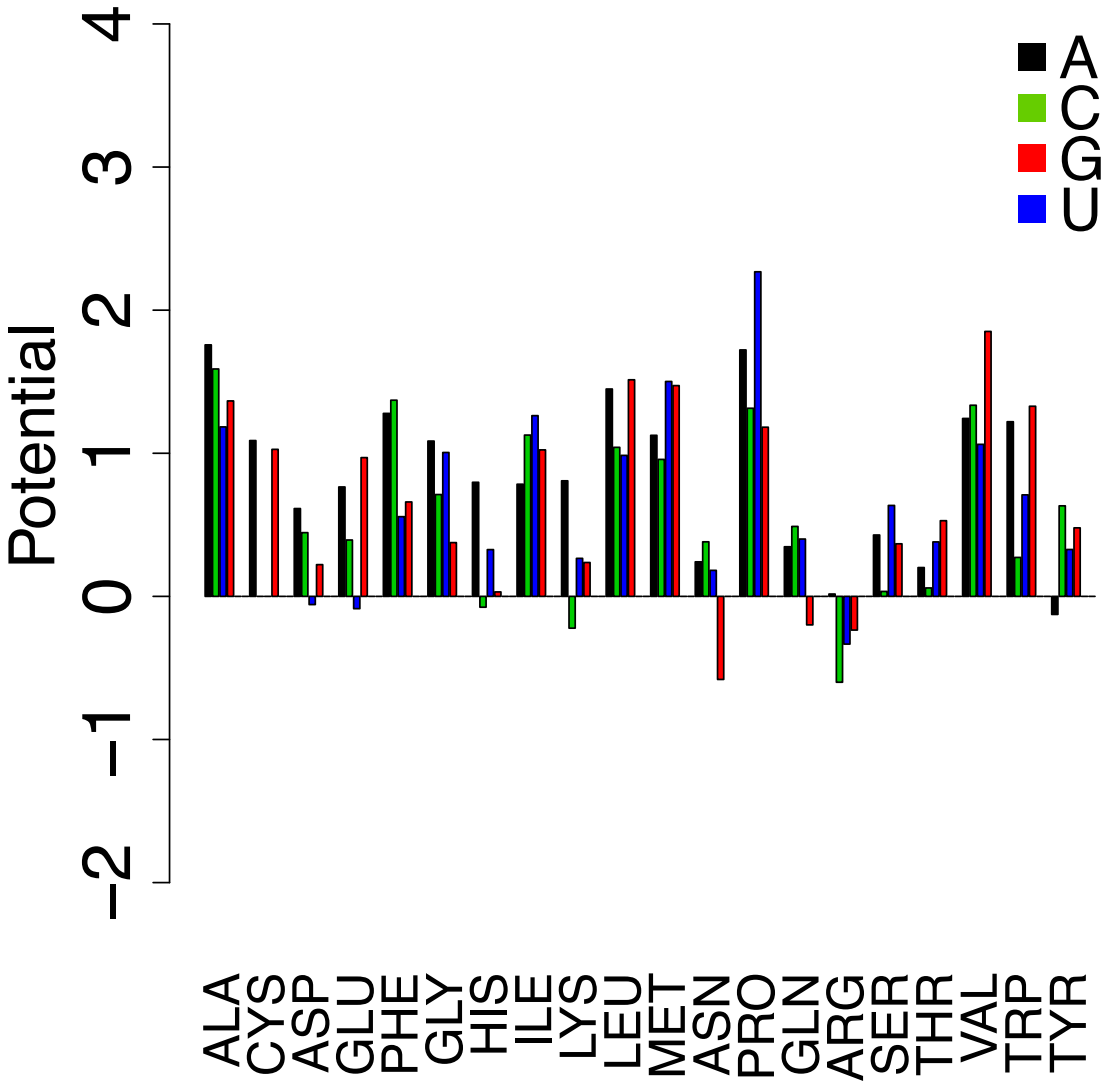

Figure B. 8. Potentials for current scenario Category 8 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

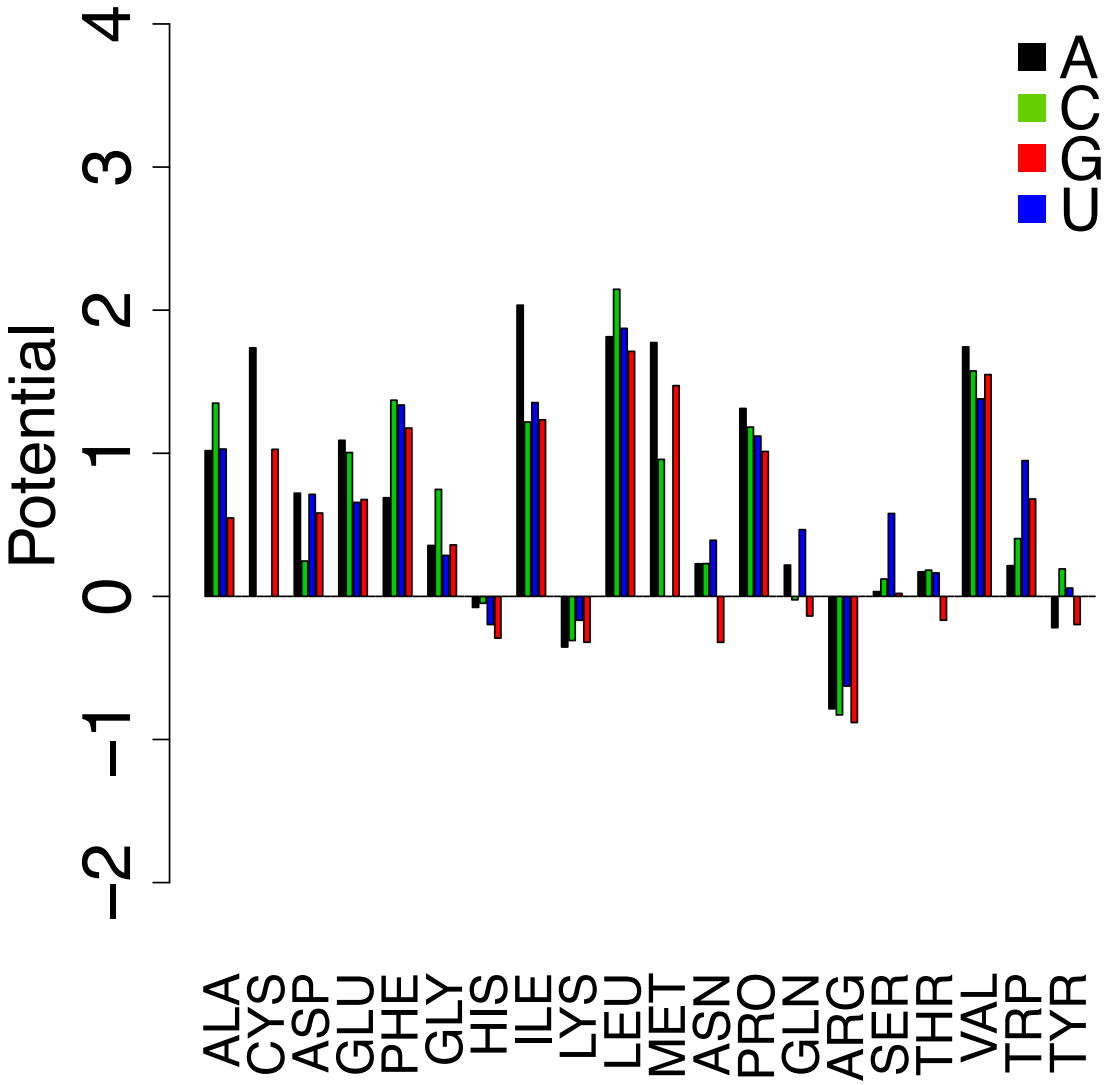

Figure B. 9. Potentials for current scenario Category 9 (potentials for missing data is set as 0). Potentials for A, C, G, and U are indicated, respectively, by black, green, red, and blue bars for each amino acid. Potentials for missing contacts are not shown in this plot. These potentials are calculated from the whole training set (without excluding any test set).

Appendix C

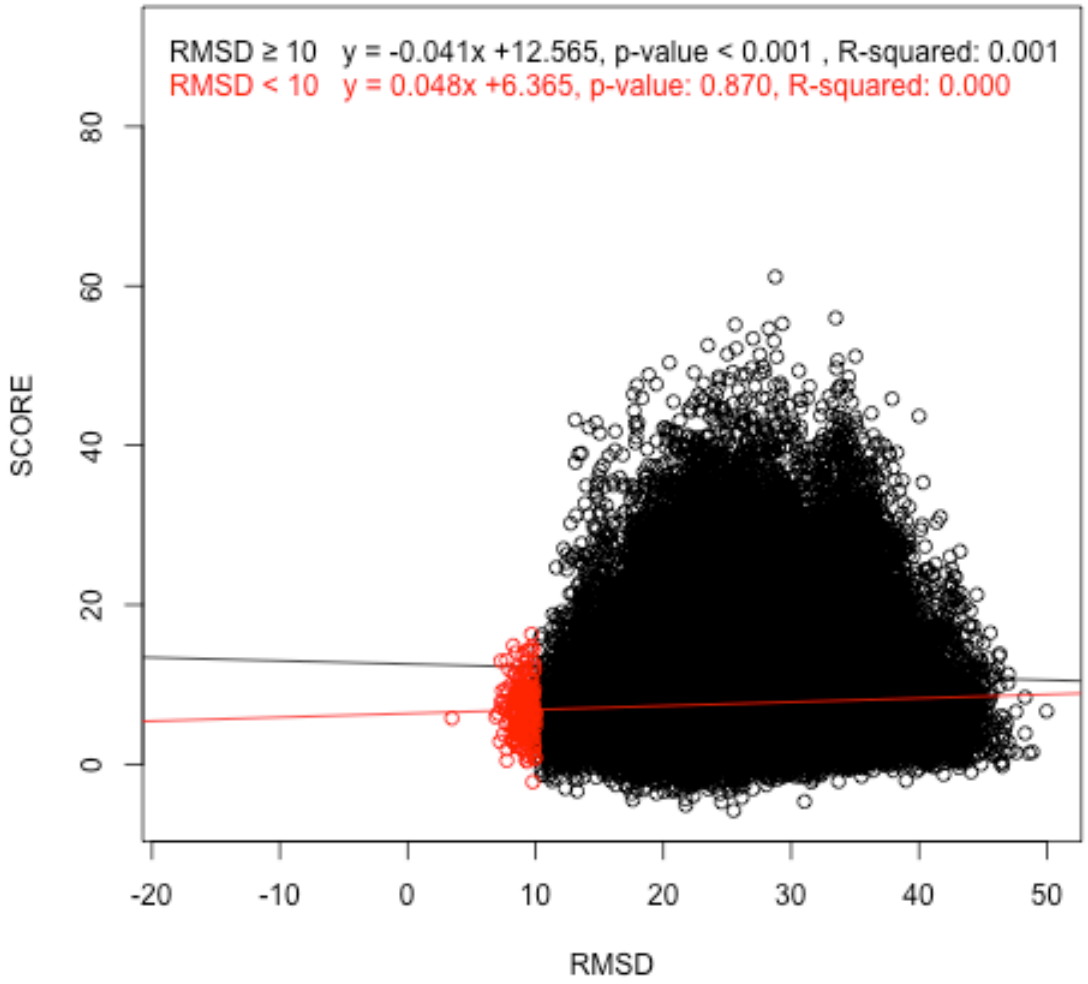

Figure C. 1. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1EC6. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

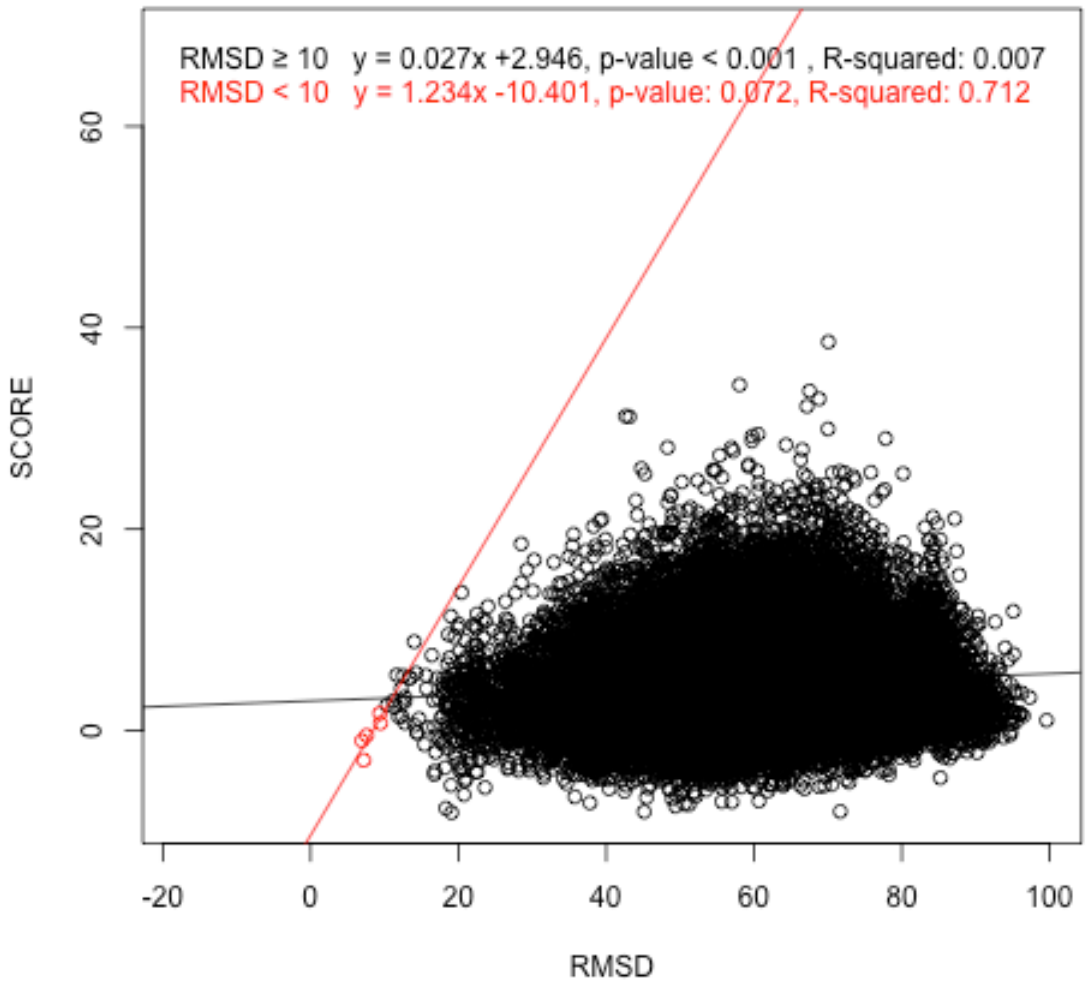

Figure C. 2. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1F7U. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

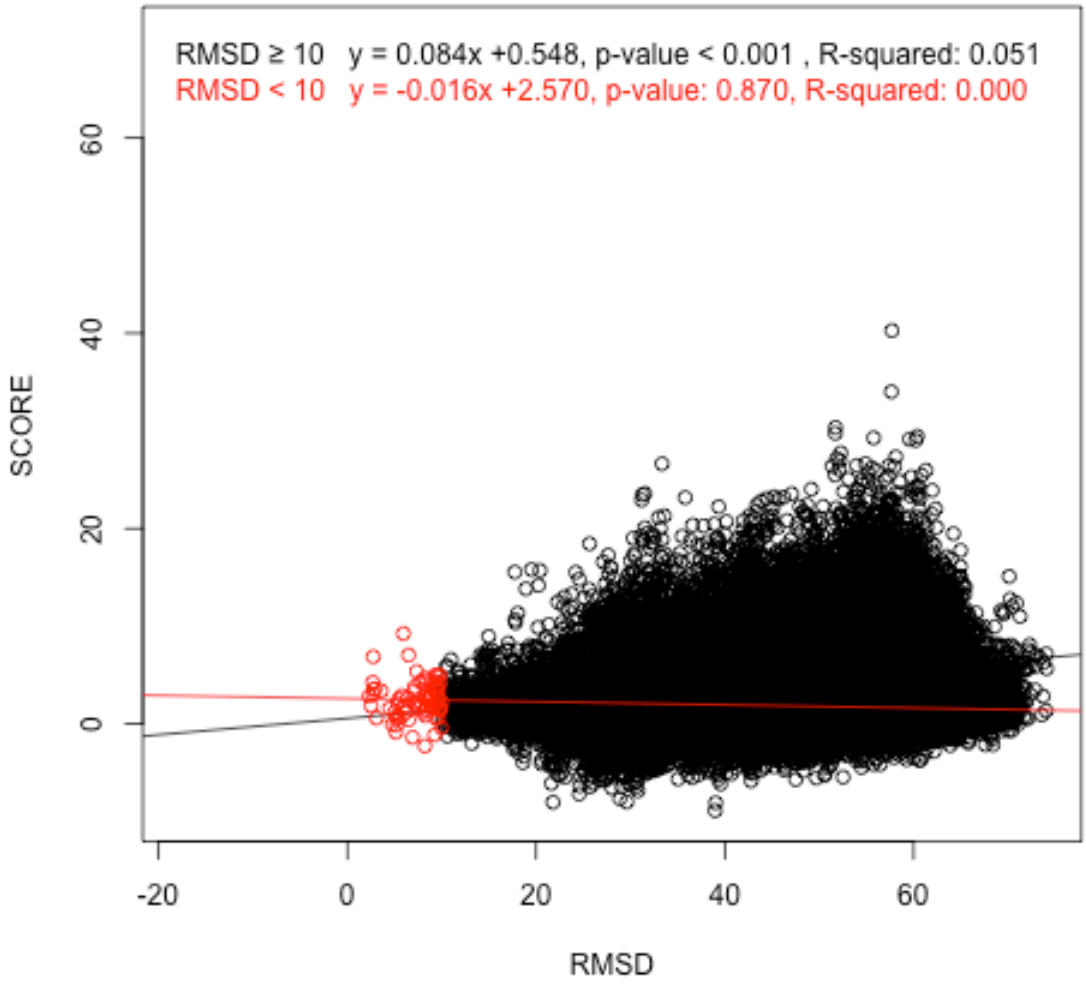

Figure C. 3. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1JBR. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

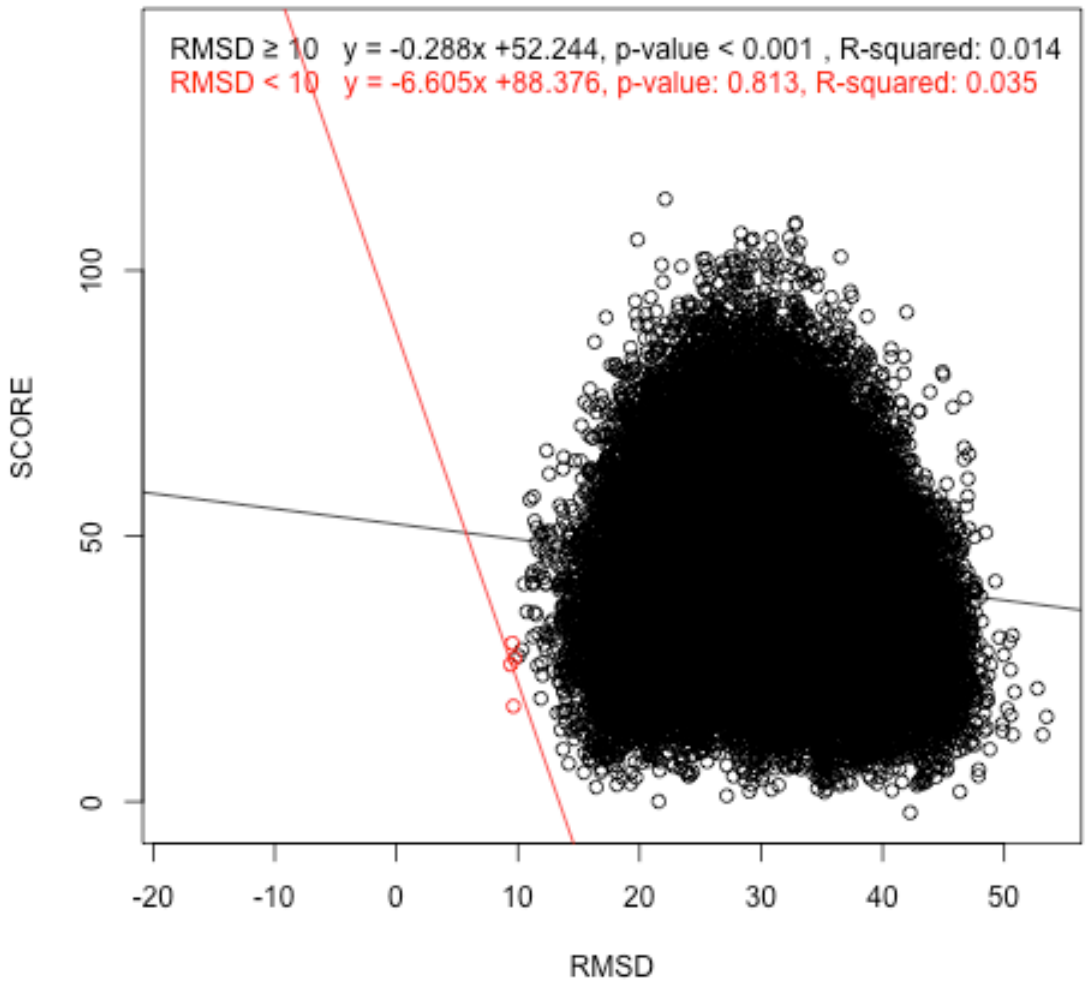

Figure C. 4. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1K8W. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

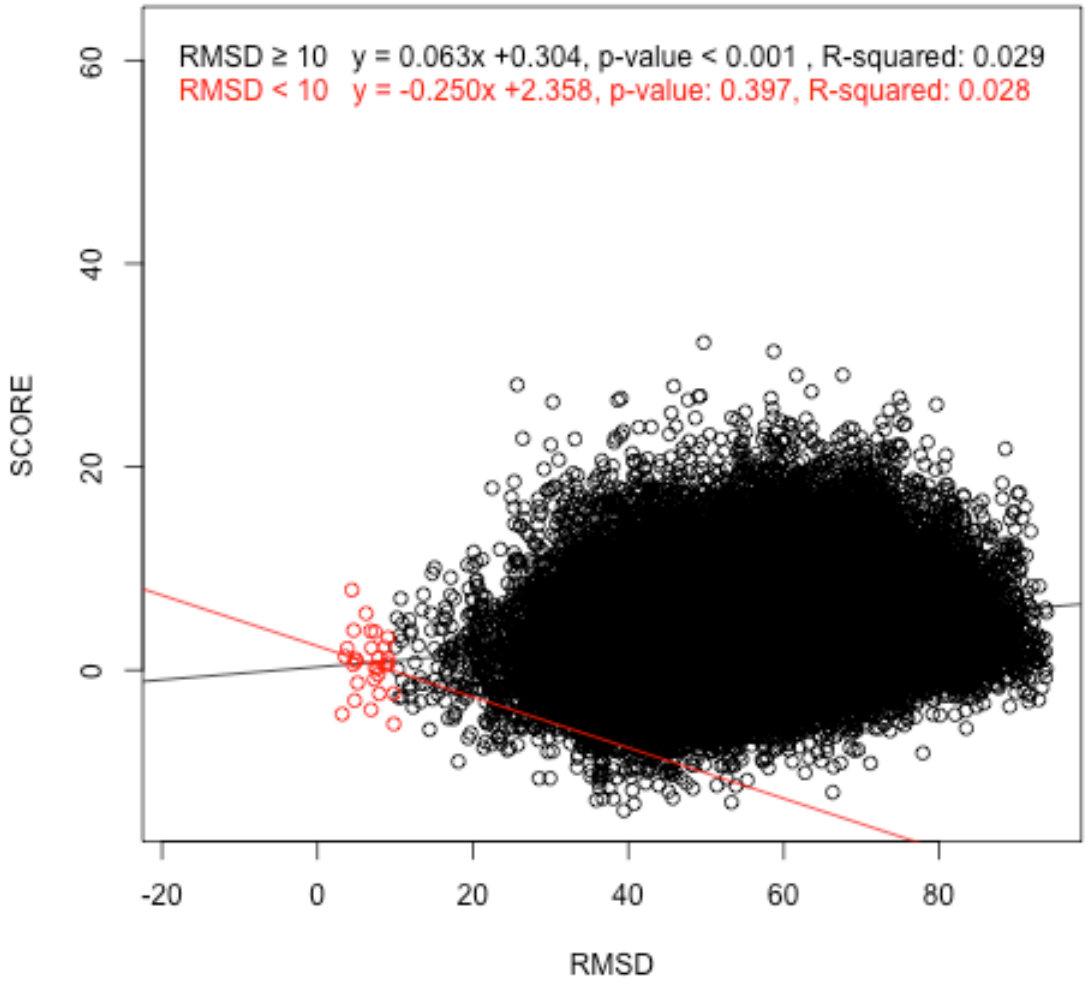

Figure C. 5. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1LNG. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

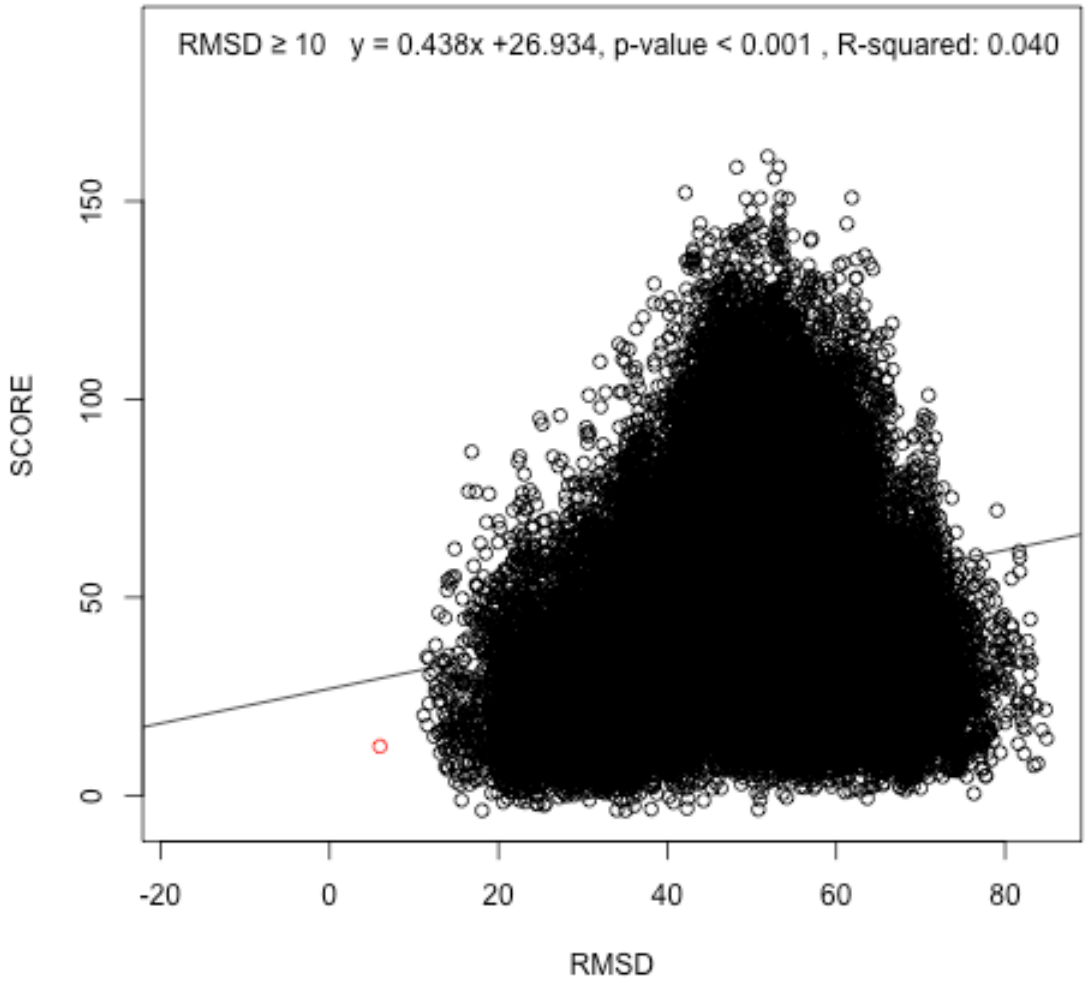

Figure C. 6. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1KOG. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

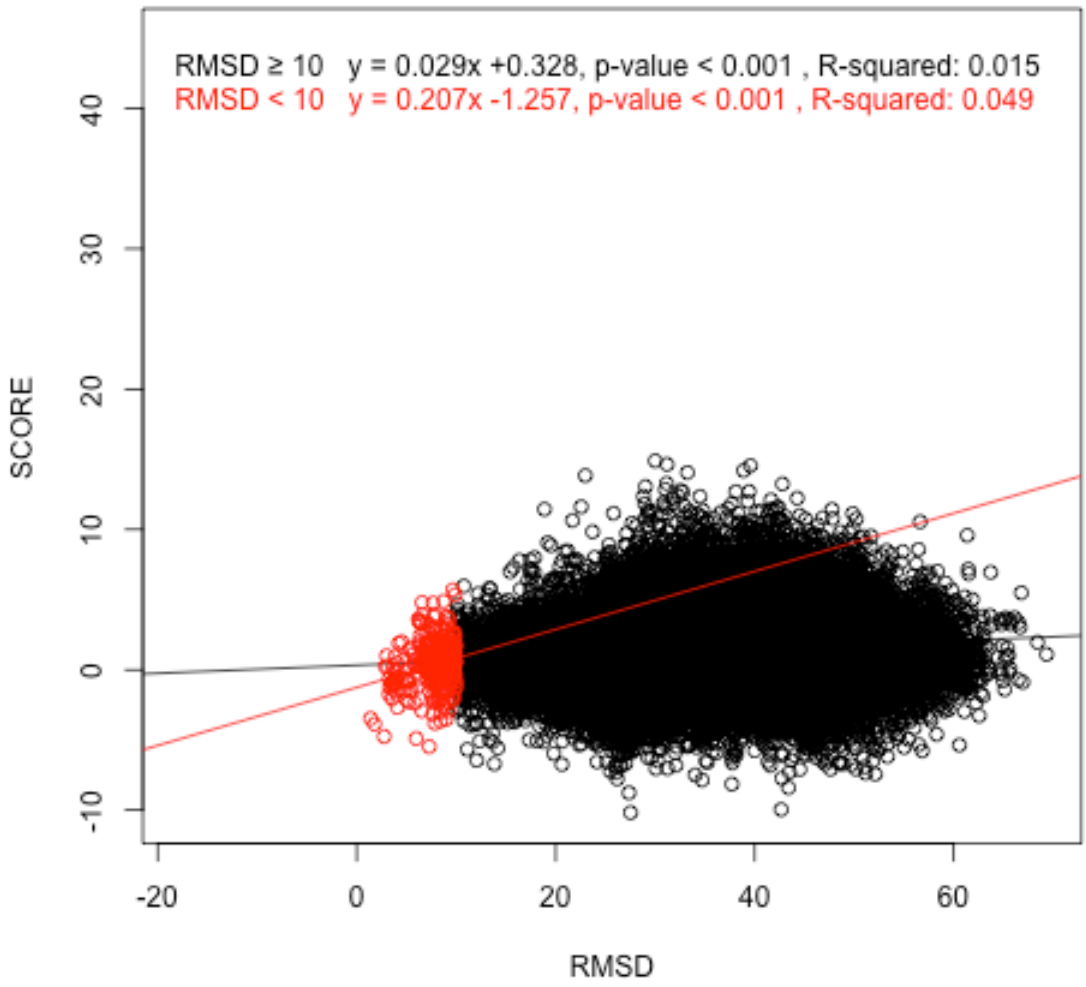

Figure C. 7. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1M8W. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

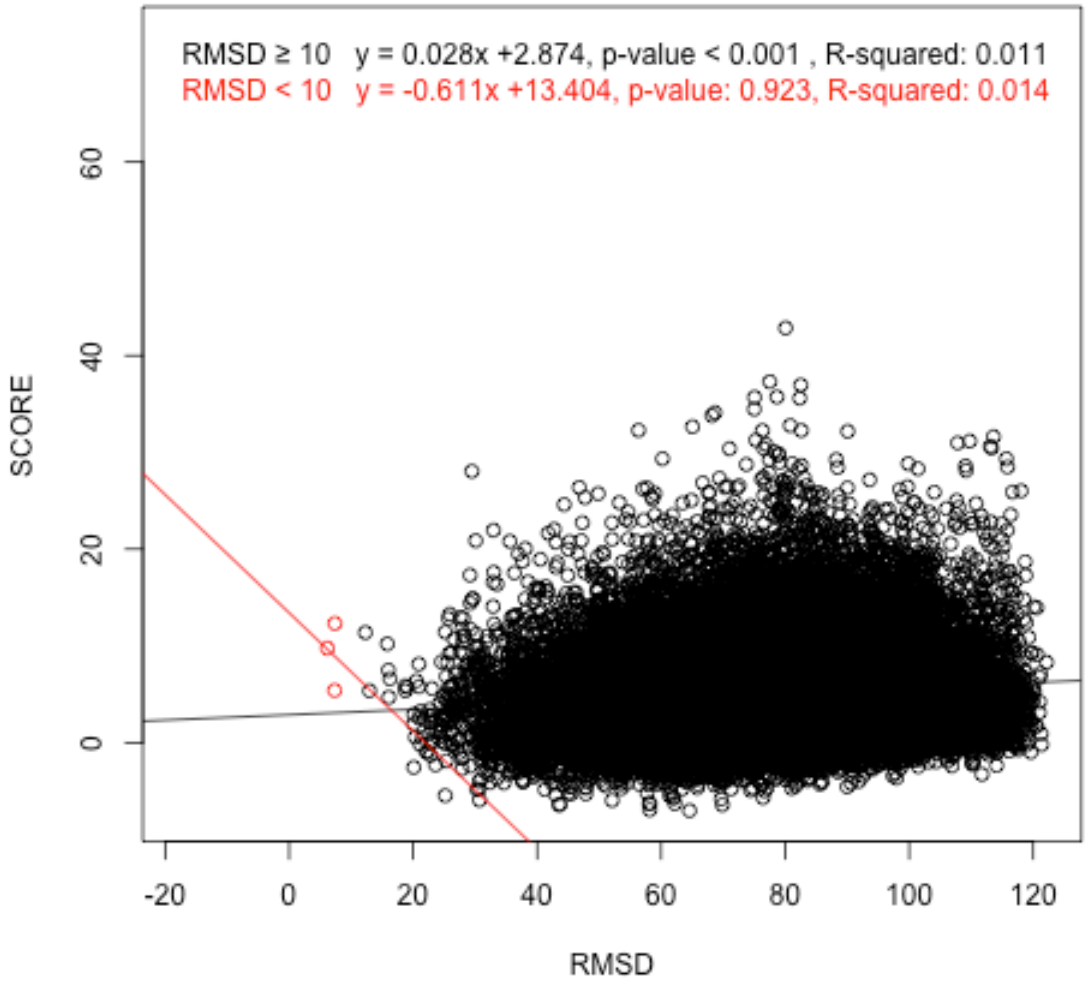

Figure C. 8. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1MFQ. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

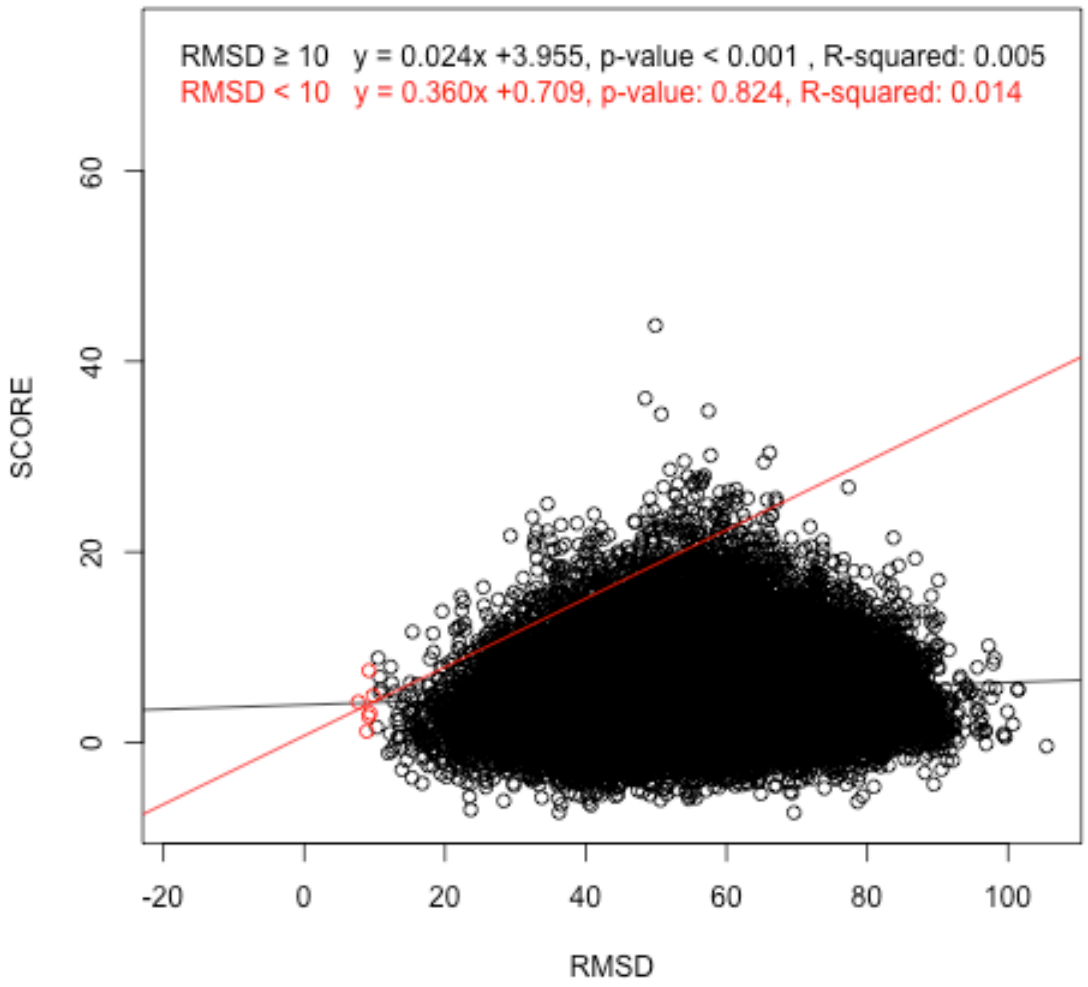

Figure C. 9. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1U0B. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

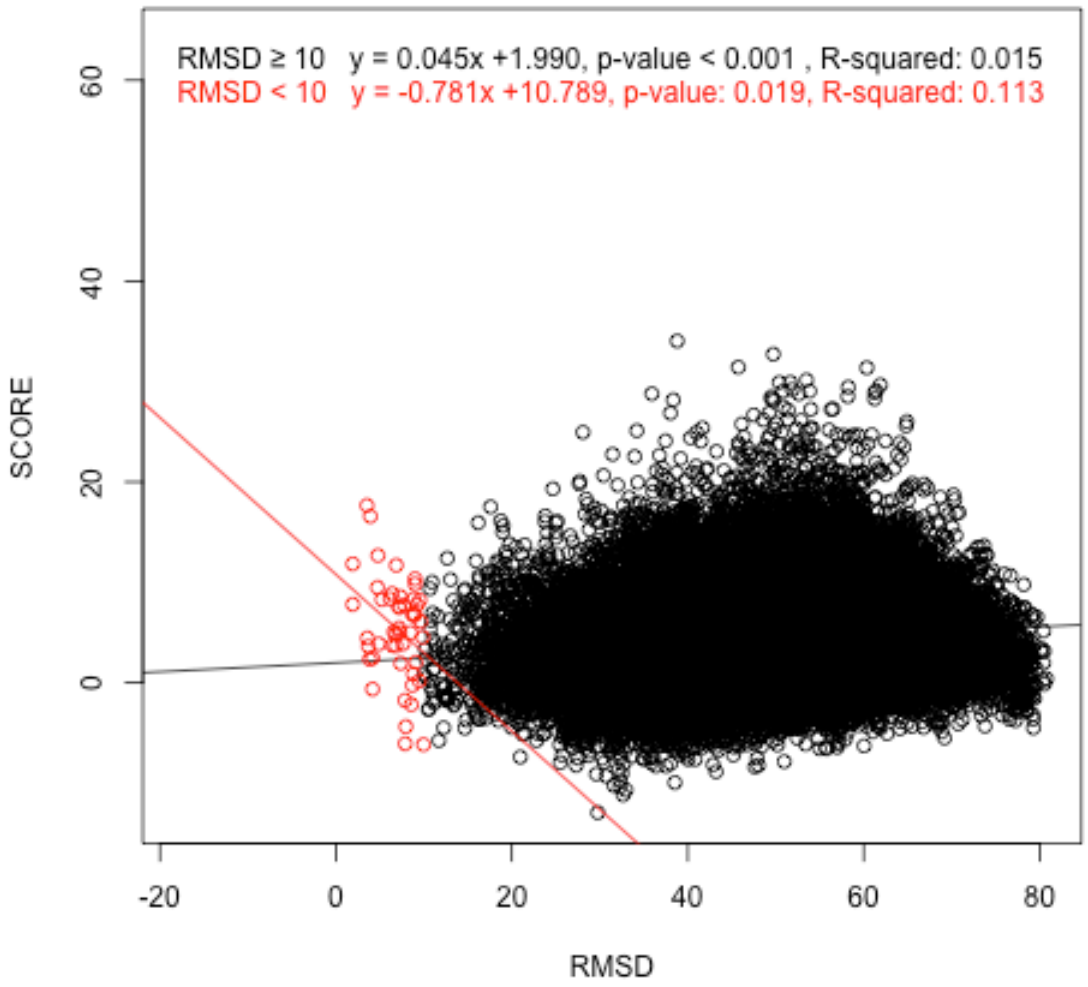

Figure C. 10. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1U63. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

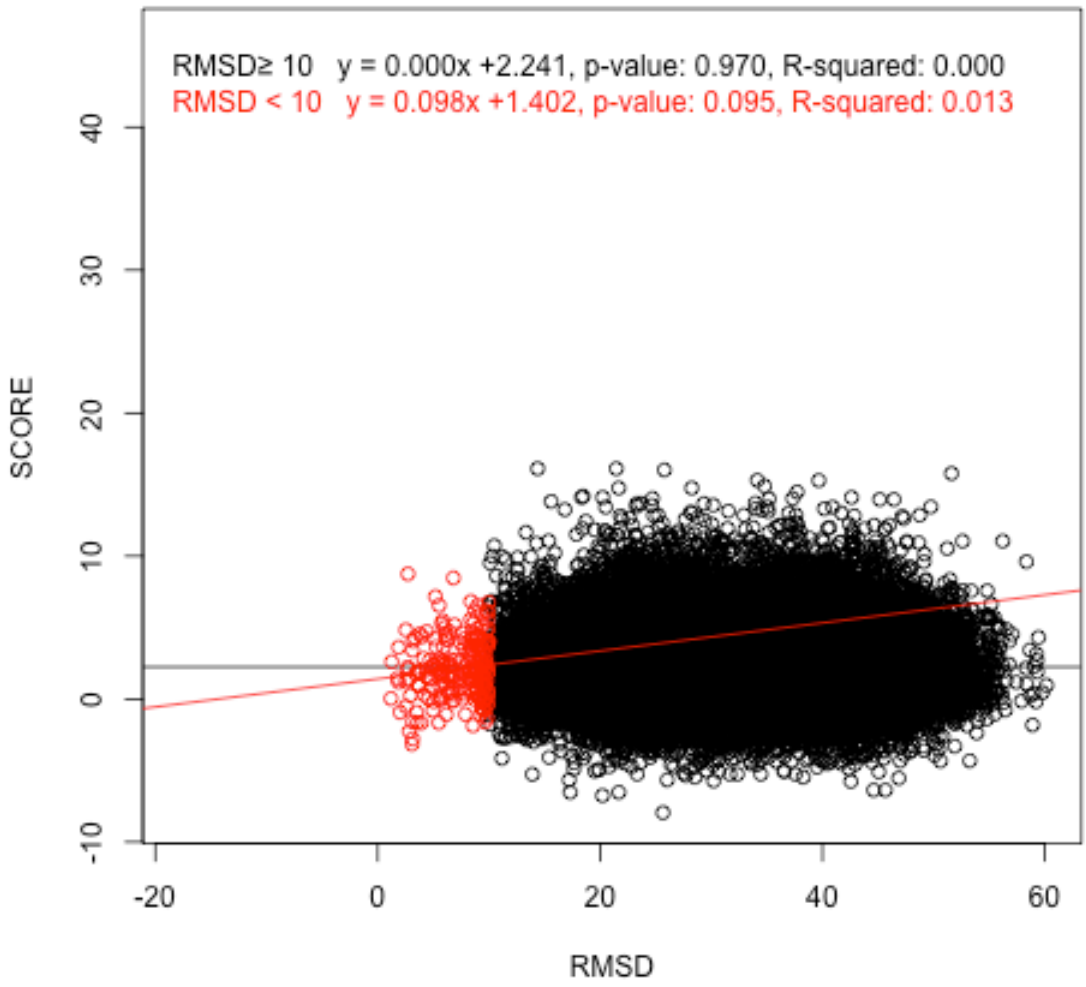

Figure C. 11. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1WPU. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

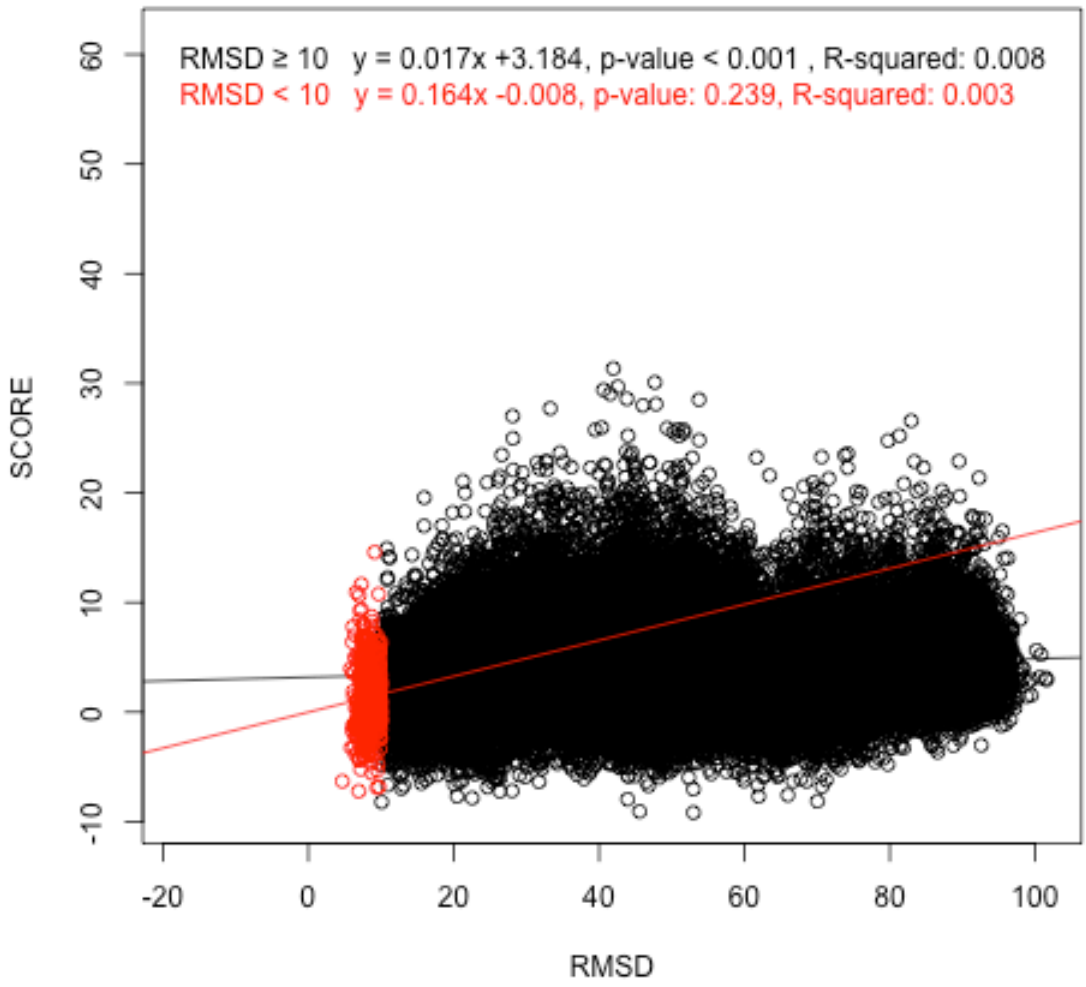

Figure C. 12. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 1WSU. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

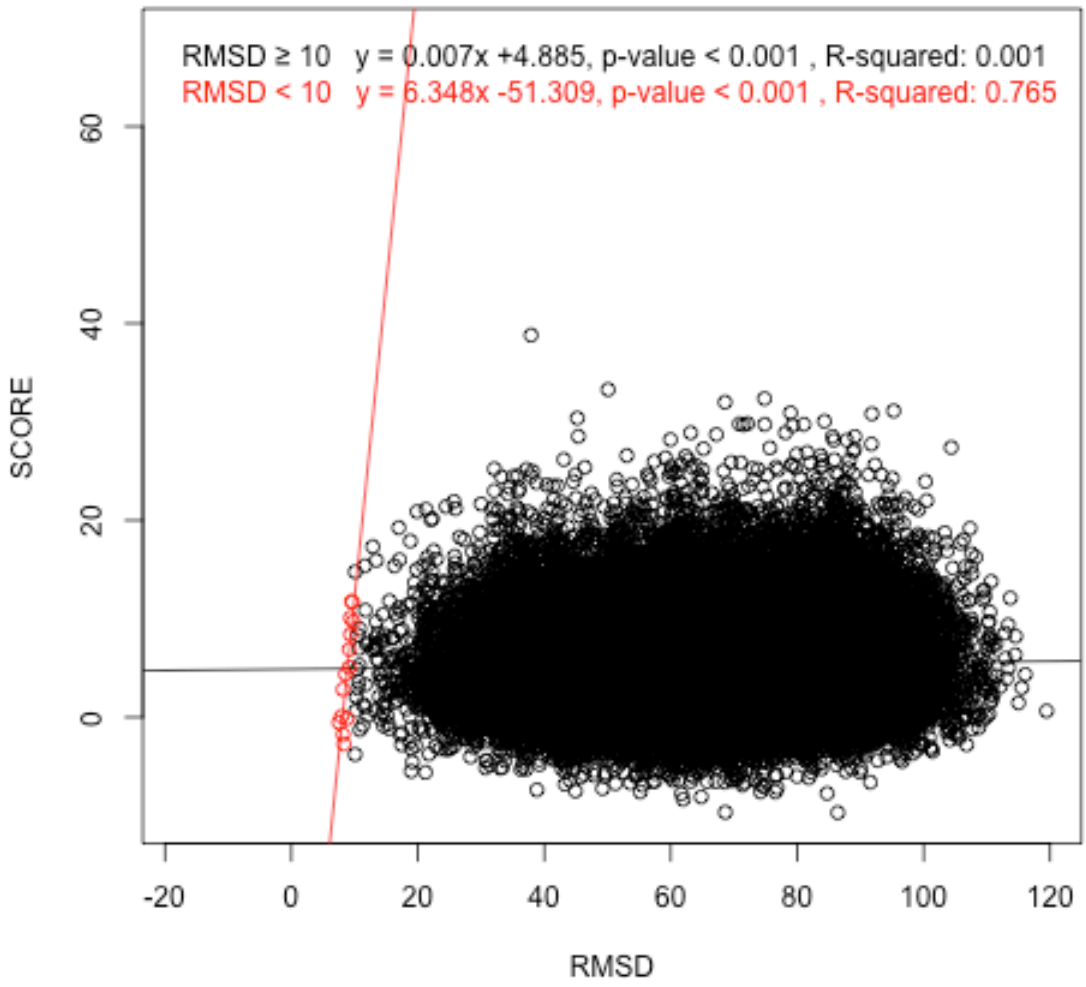

Figure C. 13. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 2BTE. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

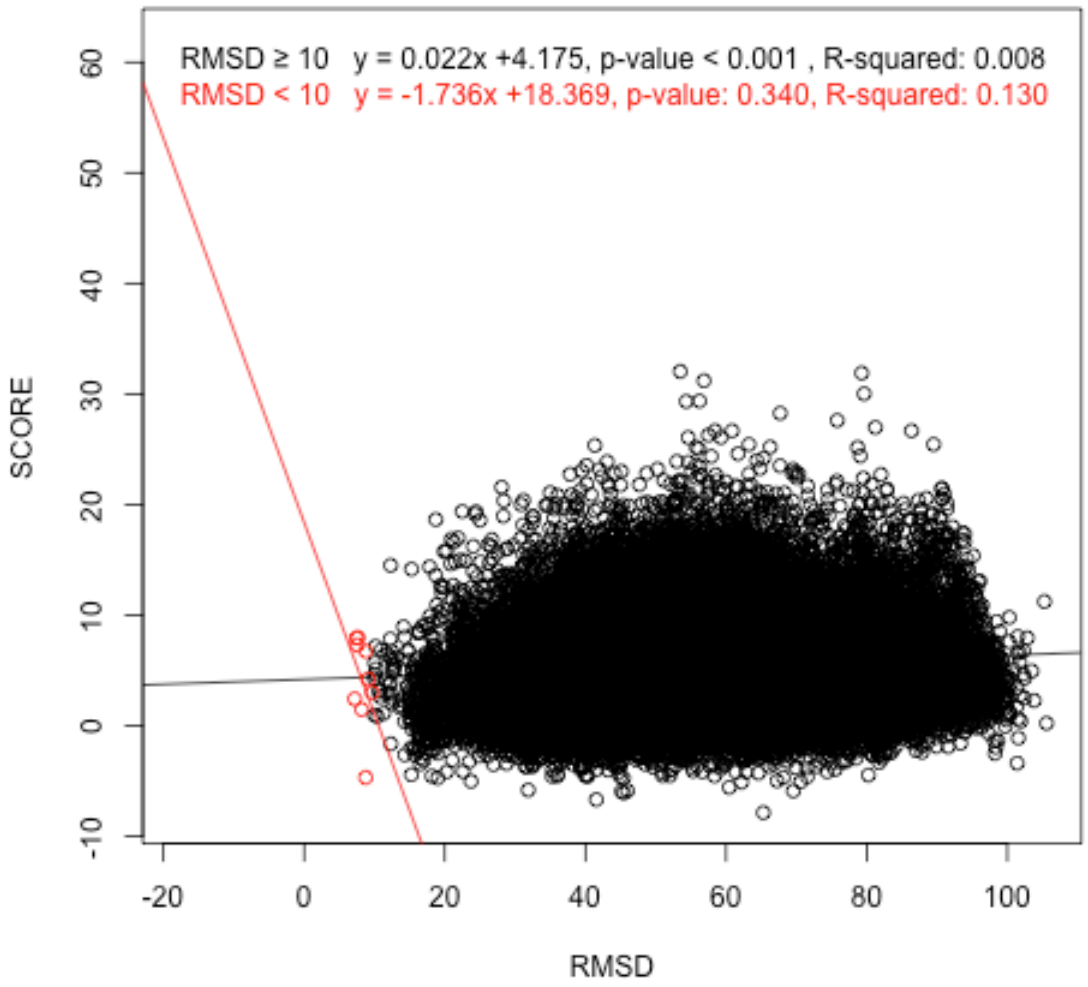

Figure C. 14. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 2FMT. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

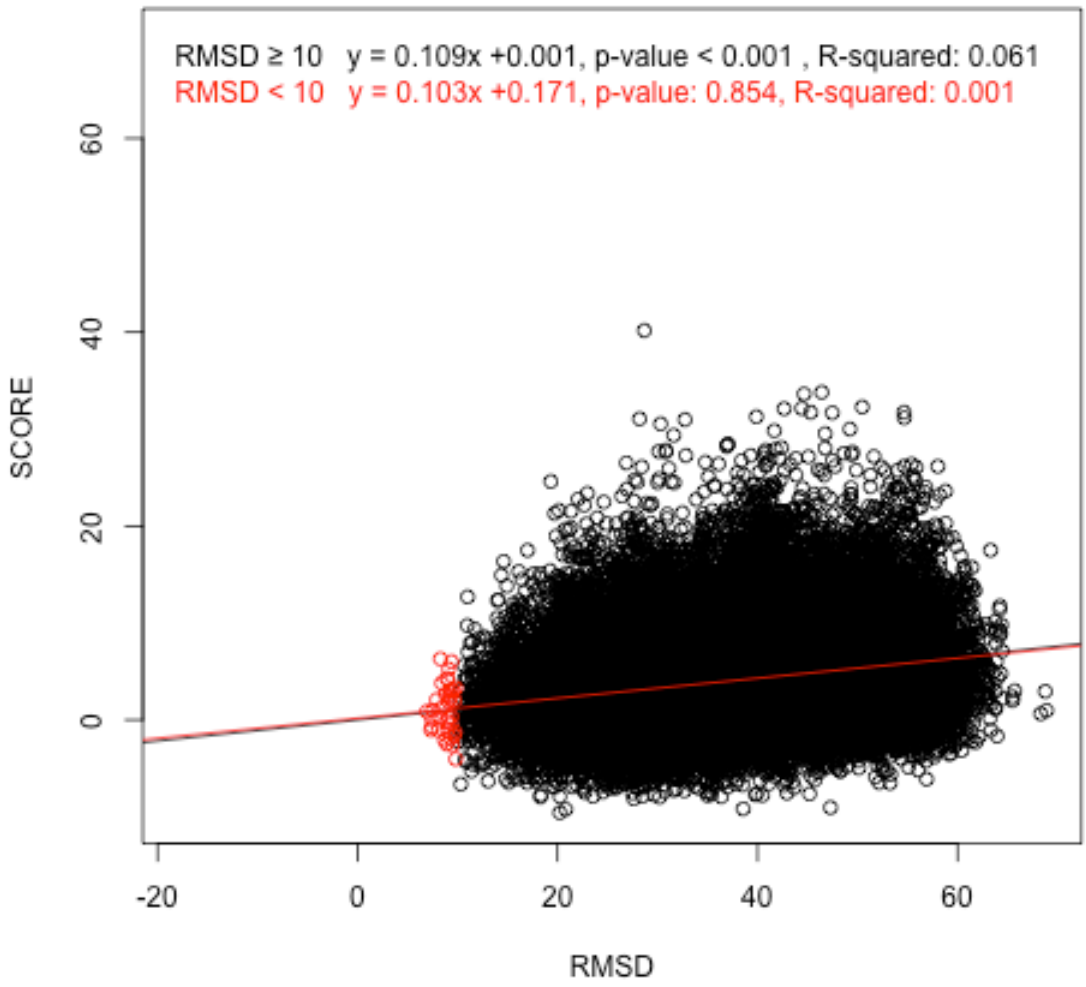

Figure C. 15. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 2HW8. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

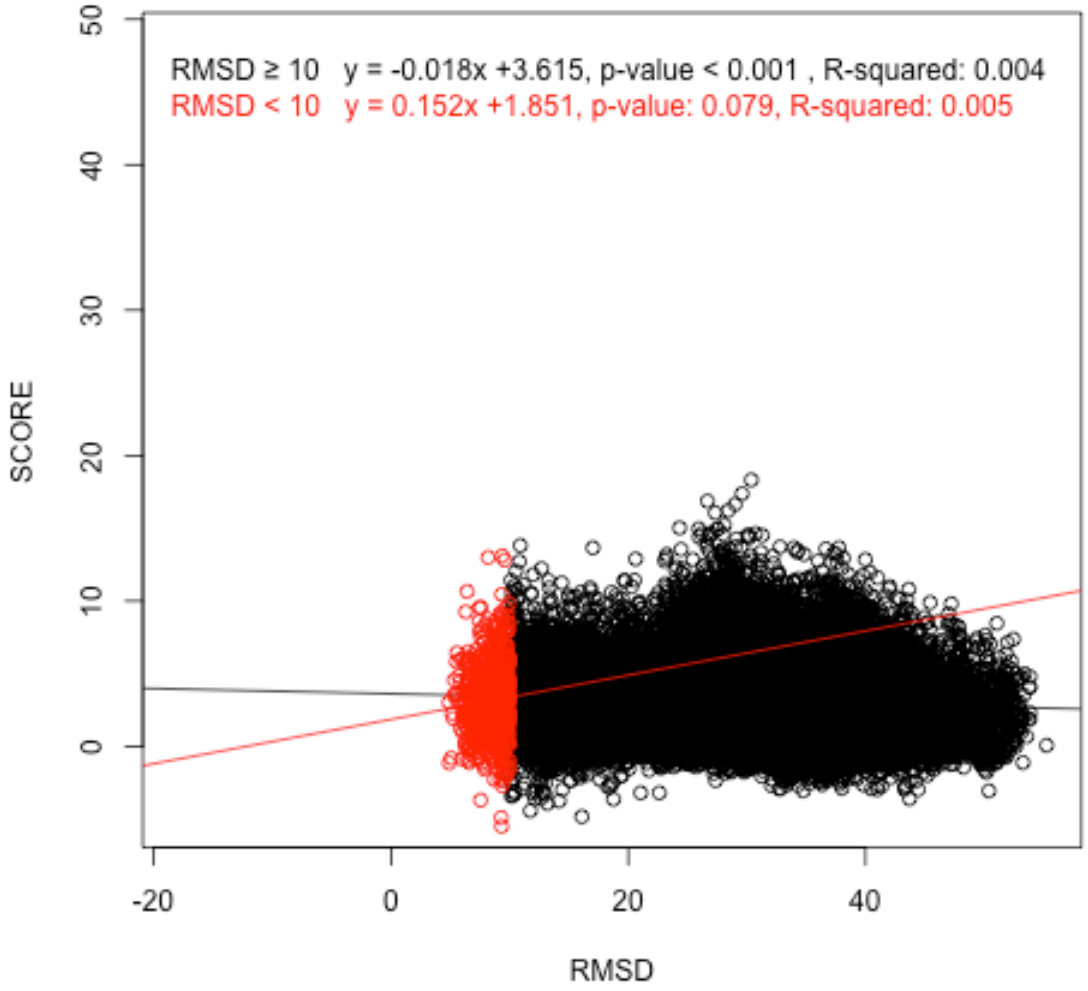

Figure C. 16. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 2JEA. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

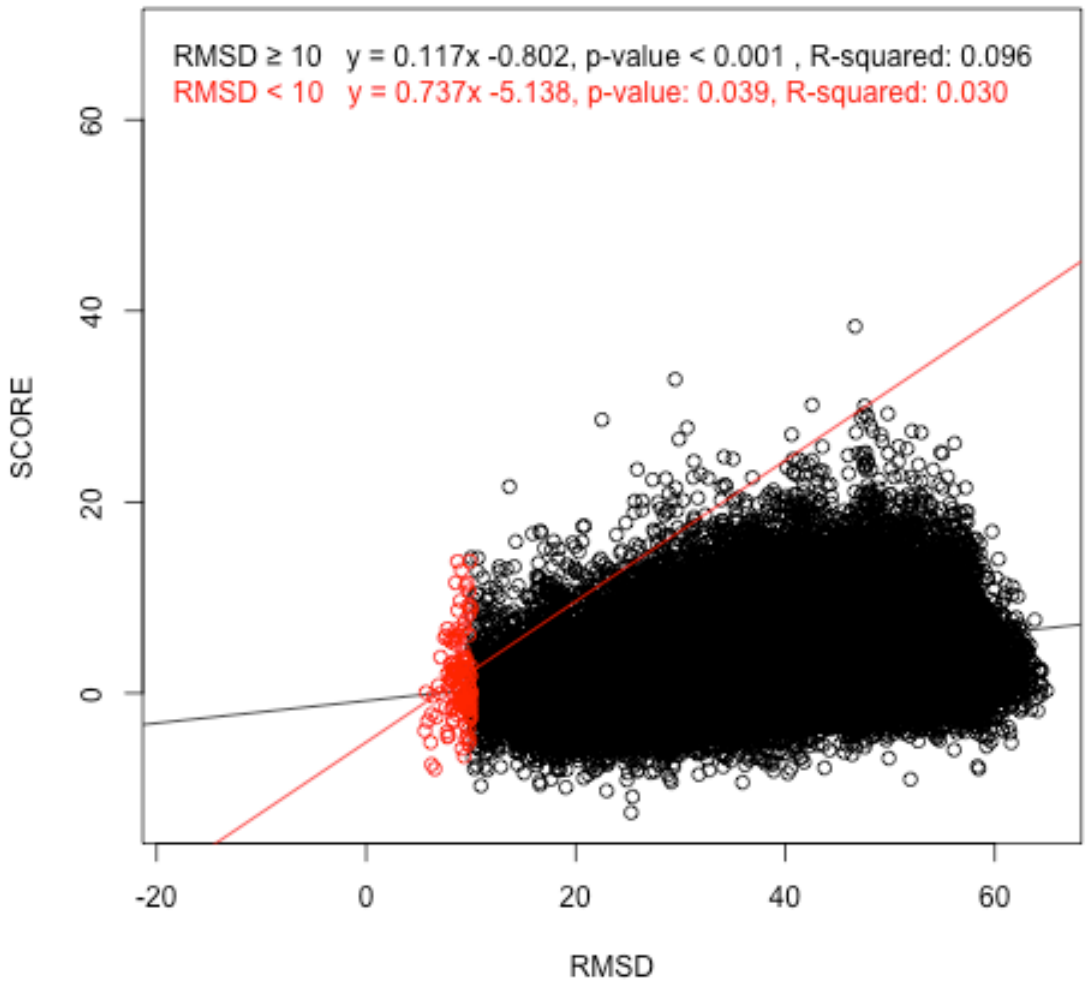

Figure C. 17. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 2PJP. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

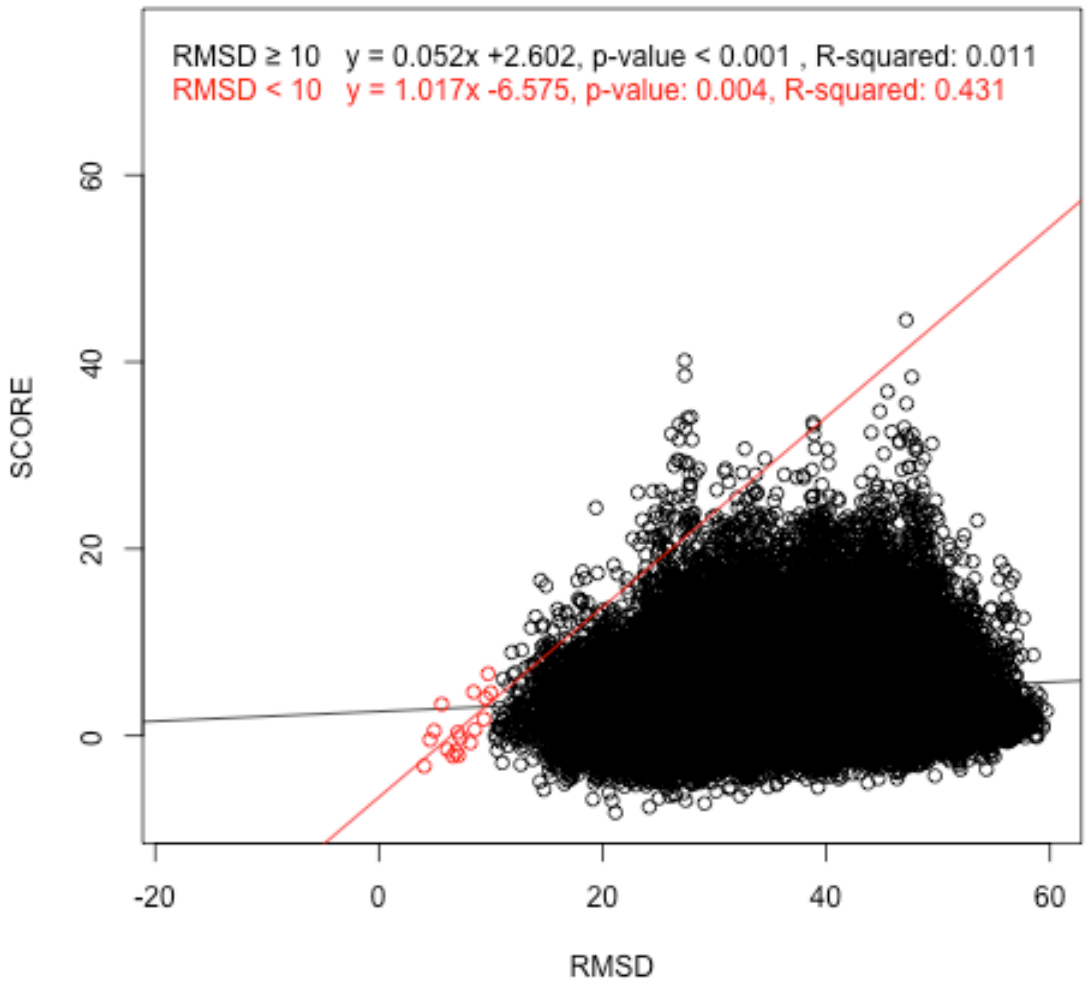

Figure C. 18. Scatter plots of score as a function of RMSD for 50,373 poses in current scenario of 2QUX. The red circles represent poses whose RMSD values are less than 10 Å and the black circles are the poses whose RMSD values equal or greater than 10 Å. The slopes, R-squared, and p-values for the regression lines for the red and black circles are shown above the scatter plots.

Appendix D Tables and Figures for Unpublished Results

Table D. 1. Classification summary of all scenarios including unpublished results

	Number of Classification						Total Number of Classifications	Mean Best Rank (%)
	Protein			RNA				
Scenario	Amino Acid Type	Atom Type	Structure	Base Type	Atom Type	Structure		
Current	20	-	-	4	-	9	720	5.352
PRat11	20	17	-	4	15	-	20,400	6.050
PRat17	20	17	-	4	15	7	142,800	11.972
PRat71	20	17	7	4	15	-	142,800	5.216
PRat77	20	17	7	4	15	7	999,600	5.872
PR77	20	-	7	4	-	7	3,920	7.275

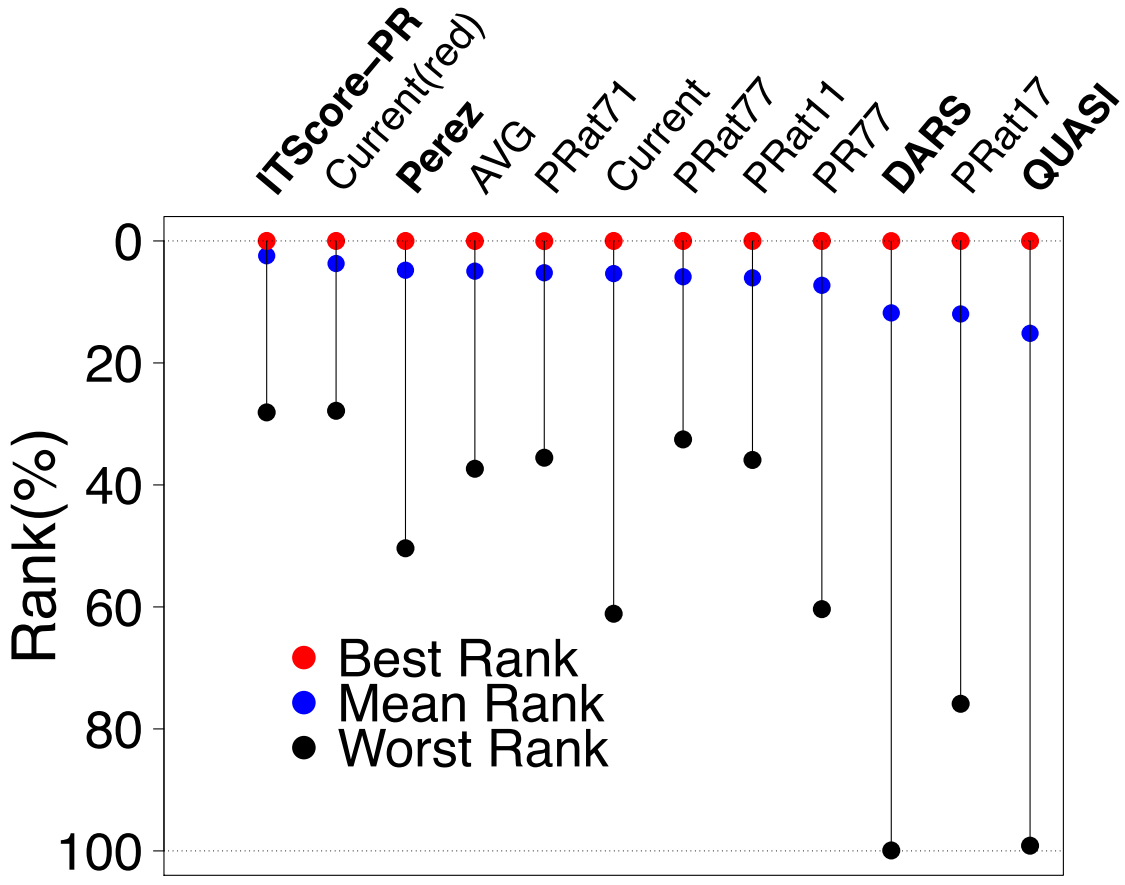

Figure D. 1. Best (red), mean (blue), and worst (black) rankings for all scenarios over twenty-one test complexes. For example, the mean rank for PRat11 (1.114%) is the average ranking among the twenty-one best rankings for the scenario.

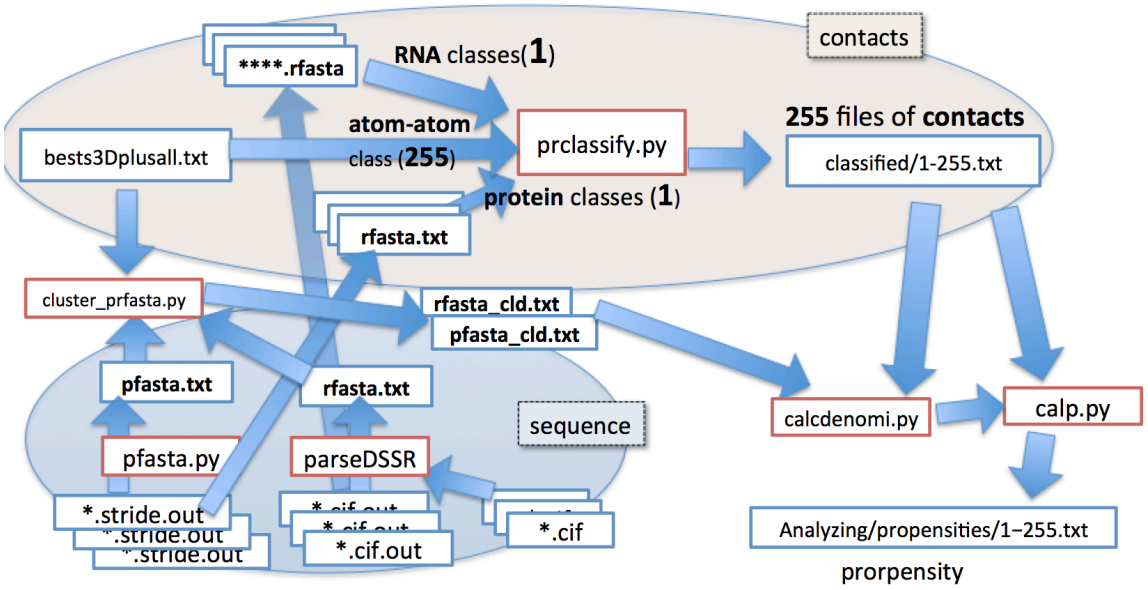

Figure D. 2. Chart Flow to Calculate Propensity for PRat11. The pfasta.txt and rfasta.txt are fasta format files that have structural identification numbers added by pfasta.py and parseDSSR.py. Protein structure was given by STRIDE.

$$\text{PRat77} \;:\; \frac{N^{C}(p1,p2,p3,r1,r2,r3)/N^{C}(all)}{N^{P}(p1,p2,p3)/N^{P}(all) \times N^{R}(r1,r2,r3)/N^{R}(all)}$$

N^{C} : Number of contacts
N^{P} : Number of amino acids (or atoms)
N^{R} : Number of bases (or atoms)

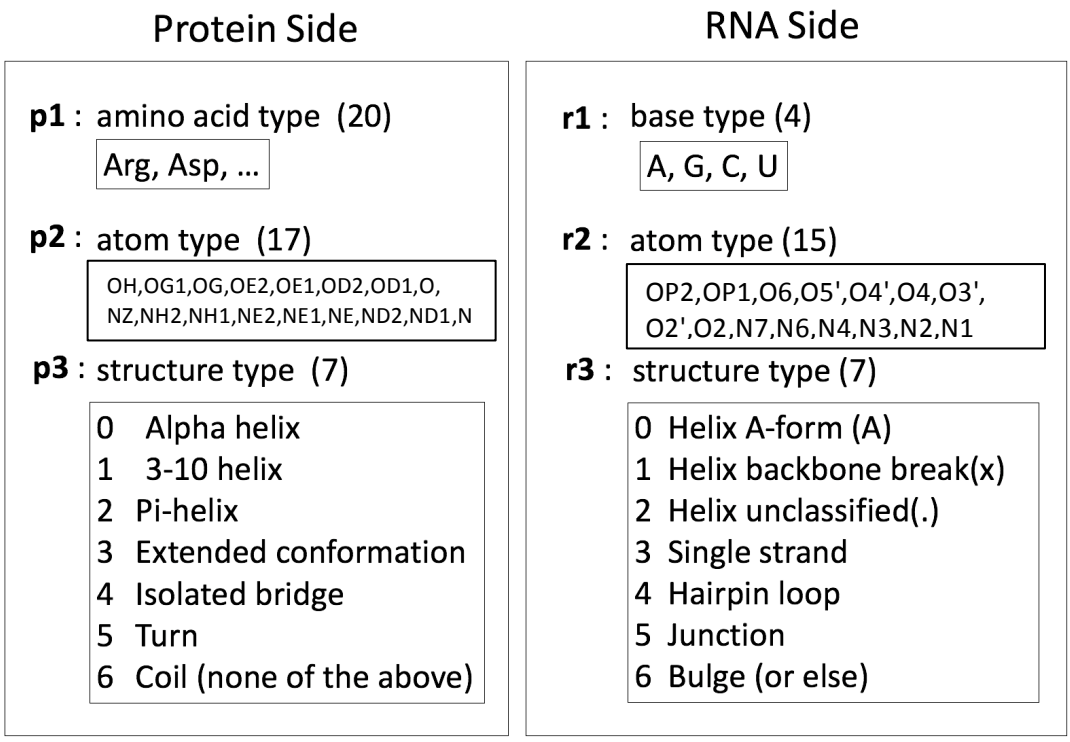

Figure D. 3. The equation to calculate propensity for PRat77 (top). N^{C}, N^{P}, and N^{R} indicate number of contacts, amino acids (or atoms in proteins), and bases (or atoms in RNA), respectively. Variables for the protein side(p1, p2, and p3) are described in the bottom-left box. Variables for the RNA side (r1,r2, and r3) are described in the bottom-right box. N^{C}(p1, p2, p3, r1, r2, r3) indicates the number of contacts classified by those six variables and N^{C} (all) indicates the number of all contacts in training set. N^{P} indicates the number of atoms in proteins classified by p1, p2 and p3. NR indicates the number of atoms in RNA classified by r1, r2, and r3.

$$\text{PRat17} \ : \ \frac{N^{C}(p1,p2,r1,r2,r3\,)/N^{C}(all)}{N^{P}\,(p1,p2\,)/N^{P}(all)\times\ N^{R}(r1,r2,r3)\,/N^{R}(all)}$$

$$\text{PRat71} \ : \ \frac{N^{C}(p1,p2,p3,r1,r2\,)/N^{C}(all)}{N^{P}\,(p1,p2,p3\,)/N^{P}(all)\times\ N^{R}(r1,r2)\,/N^{R}(all)}$$

$$\text{PRat11} \ : \ \frac{N^{C}(p1,p2,r1,r2\,)/N^{C}(all)}{N^{P}\,(p1,p2\,)/N^{P}(all)\times\ N^{R}(r1,r2)\,/N^{R}(all)}$$

$$\text{PR77} \ : \ \frac{N^{C}(p1,p3,r1,r3\,)/N^{C}(all)}{N^{P}\,(p1,p3\,)/N^{P}(all)\times\ N^{R}(r1,r3)\,/N^{R}(all)}$$

$$at77 \ : \ \frac{N^{C}(p2,p3,r2,r3\,)/N^{C}(all)}{N^{P}\,(p2,p3\,)/N^{P}(all)\times\ N^{R}(r2,r3)\,/N^{R}(all)}$$

Figure D. 4. Equations for scenarios (PRat17, PRat71, PRat11, PR77, and at77). Variables N^P and N^R indicate a number of atoms (atom name is denoted by p2 and r2) in the amino acid (p1) and the base (r1), respectively, but for PR77, N^P and N^R indicate a number of amino acids (p1) and bases (r1).

Appendix E Python Program Listings

Program: transformPISCES.py

```
path = '/Users/kimuratakayuki/Desktop/Thesis/Clustering/PISCES/30letter/'
previousID1,previousID2 = ","
i = 0

with open(path+'similarityNumed.txt','w+') as fo:
    with open(path+'similarity25rev.txt') as fi:
        for ilines in fi.readlines():
            ilinelist = ilines.split()

            # skip if same as the line above
            if previousID1 == ilinelist[1] and previousID2 == ilinelist[2]:
                continue

            # write both chainIDs if at the first line of the cluster
            elif previousID1 != ilinelist[1] or previousID1 == '':
                i += 1
                fo.writelines(str(i)+'\t'+ilinelist[1]+'\n')
                fo.writelines(str(i)+'\t'+ilinelist[2]+'\n')

            # if consecutive line, write only second chainID
```

```
            elif previousID1 == ilinelist[1]:
                fo.writelines(str(i)+'\t'+ilinelist[2]+'\n')

            previousID1 = ilinelist[1]
            previousID2 = ilinelist[2]
```

Program: GetClusterNum.py

```
# Based on chains in similarityNumed.txt that is from PISCES output,
# this code pulls resolution and method and add them to lines in similarityNumed.txt
def GetClusterNum(mode):

    import os
    clusterNum,reso,found = 0,0,0
    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    path1 = '/Users/kimuratakayuki/Desktop/Thesis/Clustering/PISCES/'

    if mode == 1:
        inputchaininfo = '5let_inputchains_valid.txt' # non redundant
    elif mode == 0:
        inputchaininfo = '5let_inputchains.txt'       # redundant

    # open the output file
    with open('/Users/kimuratakayuki/Desktop/PRat77/Clustering/chainsInfo.txt','w+') as
fo:
        with open(path1+inputchaininfo) as fr:
            for rline in fr.readlines():
                entry = rline[0:4]

                for filename in
os.listdir('/Users/kimuratakayuki/Desktop/PRat77/Clustering/mmCIF/'):
                    method = ''
                    reso = ''
                    if filename[0:4].upper() == entry and filename[-3:] == 'cif':
                        chainID = rline[0:5]

                        # open mmCif and obtain method and resolution
                        with
open('/Users/kimuratakayuki/Desktop/PRat77/Clustering/mmCIF/'+filename) as fcif:
                            for linecif in fcif.readlines():

                                # When the method is X-RAY
                                if 'X-RAY DIFFRACTION' in linecif:
                                    method = 'XRY'
                                    with
open('/Users/kimuratakayuki/Desktop/PRat77/Clustering/mmCIF/'+filename) as fcif2:
```

```
                        for linecif2 in fcif2.readlines():
                            if '_reflns.d_resolution_high' in linecif2:
                                resofull = (linecif2[27:]).strip()
                                reso = resofull[:3]
                                if reso == '?':
                                    with
open('/Users/kimuratakayuki/Desktop/PRat77/Clustering/mmCIF/'+filename) as fcif3:
                                        for linecif3 in fcif3.readlines():
                                            if '_refine.ls_d_res_high' in linecif3:
                                                resofull = (linecif3[23:]).strip()
                                                reso = resofull[:4]
                                                break
                                        break
                                break
                            elif '_refine.ls_d_res_high' in linecif2:
                                resofull = (linecif2[23:]).strip()
                                reso = resofull[:3]
                                break
                        break

                # obtain cluster#
                with open(path1+'30letter/similarityNumed.txt') as fp:
                    found = 0
                    for plines in fp.readlines():
                        plinelist = plines.split()
                        chainID = chainID.replace(':','')
                        if chainID.upper() == plinelist[1]:
                            clusterNum = plinelist[0]
                            found = 1
                            break

                if found == 1:

fo.writelines(chainID+'\t'+str(clusterNum)+'\t'+method+'\t'+str(reso)+'\n')
                elif found == 0:
                    fo.writelines(chainID+'\t'+'0'+'\t'+method+'\t'+str(reso)+'\n')
```

Program: combincontacts.py

```
# This code change the order of RNAchain and protein chain in DSSR output
# in this order.
# then from 'chainsinfo.txt', add resolution and cluster# to each contact

def combinecontacts(resolim,Cname):
    import os
    i = 0
```

```
    path = '/Users/kimuratakayuki/Desktop/Thesis/Clustering/'
    xpath = '/Volumes/Transcend/hbonds/'+Cname+'/'
    os.remove(path+'contacts.txt')

    with open(path+'contacts.txt','w+') as fo:
        with open(path+'chainsInfo.txt') as chainf:
            for chline in chainf.readlines():
                chlinelist = chline.split('\t')
                chainID = chlinelist[0]
                entry = chainID[0:4]
                for filename in
os.listdir('/Users/kimuratakayuki/Desktop/PRat77/Clustering/mmCIF/'):
                    if filename[0:4].upper() == entry.upper() and 'out' in filename:
                        with
open('/Users/kimuratakayuki/Desktop/PRat77/Clustering/mmCIF/'+filename) as fi:
                            for iline in fi.readlines():
                                # 0     1    2      3    4     5   6          7
                                # 937   634  #18    p    3.871 O/N O@P.PHE336 N4@R.C141
                                ilinelist = iline.split()
                                if ':' in iline and len(ilinelist) > 6:
                                    # c1 = [N@1:B, G0]  c2 = [O@1:A, PRO1]
                                    c1 = ilinelist[6].split('.')
                                    c2 = ilinelist[7].split('.')

                                    # nc1 = 'G'   nc2 = 'PRO'
                                    nc1 = c1[1].strip('0123456789')
                                    nc2 = c2[1].strip('0123456789')

                                    # if len(c1)+len(c2) == 4:
                                    if len(nc1) == 3 and len(nc2) == 1:
                                        tmp = ilinelist[7]
                                        ilinelist[7] = ilinelist[6]
                                        ilinelist[6] = tmp

                                    # c1 = [N@1:B, G0]  c2 = [O@1:A, PRO1]
                                    c1 = ilinelist[6].split('.')
                                    c2 = ilinelist[7].split('.')

                                    # nc1 = 'G'   nc2 = 'PRO'
                                    nc1 = c1[1].strip('0123456789')
                                    nc2 = c2[1].strip('0123456789')

                                    # d1 = [N@1, B]   d2 = [O@1, A]
                                    d1 = c1[0].split(':')
                                    d2 = c2[0].split(':')
                                    if resolim != 10:
```

```
                        if 'XRY' in chline:
                            # get cluster#, method, resolution from chainsInfo.txt **
protein **
                            if chainID == filename[:-8].upper()+d2[1]:
                                clusterN = chlinelist[1]
                                method = chlinelist[2]
                                resolution = chlinelist[3].strip('\n')
                                if float(resolution) < resolim:
                                    fo.writelines(str(i)+'\t'+filename[:-
8].upper()+'\t'+ilinelist[0]+'\t'+\

ilinelist[1]+'\t'+ilinelist[4]+'\t'+ilinelist[6]+'\t'+ilinelist[7]+\
                                        '\t'+clusterN+'\t'+method+'\t'+resolution+'\n')
                                        i += 1
                        elif resolim == 10:
                            if chainID == filename[:-8].upper()+d2[1]:
                                clusterN = chlinelist[1]
                                fo.writelines(str(i)+'\t'+filename[:-
8].upper()+'\t'+ilinelist[0]+'\t'+ilinelist[1]+'\t'+ilinelist[4]+\

'\t'+ilinelist[6]+'\t'+ilinelist[7]+'\t'+clusterN+'\t'+'any'+'\t'+'0'+'\n')
                                i += 1
```

Program: director_cluster.py

```
def director_cluster(potentials,complist,resolim,Cname):

    if potentials == 98 or potentials == 99:
        mode = 0
        # 1:non-redundant data   0:redundant data(filter with resolution, method, and cluster)
    elif potentials != 98 and potentials != 99:
        mode = 1

    from GetClusterNum import GetClusterNum
    GetClusterNum(mode)

    from combinecontacts import combinecontacts
    combinecontacts(resolim,Cname)

    if mode == 1:  # with clustering
        from choosebest_f import choosebest
        choosebest(complist,resolim)
        print('** non-redundant mode **')

        from getallredun_f import getallredun
```

```
        getallredun(complist)
        print('** allredun is made **')

    elif mode == 0:  # without clustering
        from choosebest_all import choosebest_all
        choosebest_all(complist)
        print('** redundant mode **')

        from getallredun_f_all import getallredun
        getallredun(complist)
        print('** allredun is made
```

Program: choosebest.py

```
def choosebest(complist,resolim):  # with clustering
    path = '/Users/kimuratakayuki/Desktop/Thesis/Clustering/'
    idlist = []
    i = 1
    amino =
['ALA','VAL','LEU','ILE','PHE','TRP','MET','PRO','ASP','GLU','GLY','SER','THR','CYS'
,'TYR','ASN','GLN','LYS','ARG','HIS']
    with open(path+'bests.txt','w+') as fo:
        with open(path+'contacts.txt') as fi:
            for line in fi.readlines():
                linelist = line.split('\t')

                #  0    1      2   3      4      5          6          7   8   9   10 : index
                #  24  1A34   1  2950   3.206  O5'@1:B.U11    N@1:A.THR13    6  XRY    1.8
1
                resi = linelist[6].split('.')
                resi2 = resi[1].strip('0123456789') # resi2 = THR or U
                resib = linelist[5].split('.')
                resib2 = resib[1].strip('0123456789') # resi2 = THR or U

                if linelist[1] in complist:
                    continue
                if len(resi2)+len(resib2) == 4:
                    if resolim != 10:
                        if resi2 in amino and linelist[8] == 'XRY':
                            if linelist[7] == '0':
                                fo.writelines(str(i)+'\t'+line)
                                i += 1
                            elif linelist[7] != 0:
                                id = linelist[7]
                                if id in idlist:
                                    continue
```

```
                elif id not in idlist:
                    idlist.append(id)
                    with open(path+'contacts.txt') as fi2:
                        bestline = ''
                        for line2 in fi2.readlines():
                            linelist2 = line2.split('\t')
                            id2 = linelist2[7]
                            if id2 == id:
                                # bestline[8]=method, bestline[9]=resolution
                                # compare current reso with bestreso
                                if bestline == '':
                                    if linelist2[8] == 'XRY':
                                        bestline = line2
                                else:
                                    if bestline[9] > line2[9]:
                                        bestline = line2
                                    else:
                                        continue

                        if bestline == '':
                            continue

                        fo.writelines(str(i)+'\t'+bestline)
                        i += 1
            elif resolim == 10:
                if resi2 in amino:
                    fo.writelines(str(i)+'\t'+line)
                    i += 1
```

Program: getallredun.py

```
# this code reads chainID like 1A1T_A from chosen contacts,
# and copy the line of the same chainID from contacts.txt to allredun.txt

def getallredun(complist):

    path = '/Users/kimuratakayuki/Desktop/Thesis/Clustering/'
    rlinelist,id6,id7,ridl,pidl,linelist,iid6,iid7,iridl,ipidl = [],[],[],[],[],[],[],[],[],[],[]
    chainid,ichainid,rid,pid,irid,ipid = ','','','','','
    i = 0

    with open(path+'allredun.txt','w+') as fo:
        with open(path+'bests.txt') as fr:
            for rline in fr.readlines():
                rlinelist = rline.split('\t')
```

```
            # 9    8  1A1T   312    1220   2.950  O6@1:B.G210    N@1:A.TRP37    0     0
            # 10   9  1A1T   312    1360   2.803  O6@1:B.G210    N@1:A.MET46    0     0
            # 22   21 1A1T   478    756    3.956  O4@1:B.U215    NH2@1:A.ARG7   0
any    0

            id6 = rlinelist[6].split(':')
            id7 = rlinelist[7].split(':')
            ridl = id6[1].split('.')
            pidl = id7[1].split('.')
            rid = ridl[0]
            pid = pidl[0]
            chainid = rlinelist[2]+'_'+rid+'_'+pid

            ires1 = ridl[1].strip('1234567890')
            ires2 = pidl[1].strip('1234567890')

            if rlinelist[2] in complist:
               print('remove a line of '+rlinelist[2])
               continue

            # write as is if the contact is not clustered
            elif rlinelist[8] == '0':
               if len(ires1) == 1 and len(ires2) == 3:

fo.writelines(str(i)+'\t'+chainid+'\t'+rlinelist[1]+'\t'+rlinelist[2]+'\t'+rlinelist[3]+\

'\t'+rlinelist[4]+'\t'+rlinelist[5]+'\t'+rlinelist[6]+'\t'+rlinelist[7]+'\t'+\
                              rlinelist[8]+'\t'+rlinelist[9]+'\t'+rlinelist[10])
                  i += 1

            # if clustered, obtain all contacts of the same chain combi ID
            else:
               with open(path+'contacts.txt') as fi:
                  for line in fi.readlines():
                     linelist = line.split('\t')

                     # line[1]=1A1T   [5]=OP2@1:B.A203   [6]=N@1:A.GLY4
                     iid6 = linelist[5].split(':')
                     iid7 = linelist[6].split(':')
                     iridl = iid6[1].split('.')
                     ipidl = iid7[1].split('.')
                     ichainid = linelist[1]+'_'+iridl[0]+'_'+ipidl[0]

                     res1 = iridl[1].strip('1234567890')
                     res2 = ipidl[1].strip('1234567890')
```

```
                    if chainid == ichainid:
                        if len(res1) == 1 and len(res2) == 3:
                            fo.writelines(str(i)+'\t'+ichainid+'\t'+line)
                            i += 1
```

Program: director_potential.py

```
# this is the director file that uses allredun.txt as an an input
# and output nine sets of potentials
def director_potential():
    nums = []

    from assignRNA3D import assignRNA3D
    assignRNA3D()
    print("2.assignRNA3D IS DONE")

    def calp_m1(k):  # k: 0-2
        from sepcalp1 import sepcalp1
        sepcalp1(k)

    import multiprocessing as mp1
    calp_processes = [mp1.Process(target=calp_m1, args=(k,)) for k in range(0,3)]

    for p in calp_processes:
        p.start()
    for p in calp_processes:
        p.join()

    from calcdenomi import calcdenomi
    r2p2 = calcdenomi()
    print("*** r2 and p2 were made ***")

    # Calculate potential set 1-9

    def calp_m(k,r2p2,nums):  # k: 0-5
        from sepcalp import sepcalp
        sepcalp(k,r2p2,nums)

    import multiprocessing as mp1
    calp_processes = [mp1.Process(target=calp_m, args=(k,r2p2,nums)) for k in
range(0,6)]

    for p in calp_processes:
        p.start()
    for p in calp_processes:
        p.join()
```

```
    from potenti import potenti
    potenti()

    print("15.potenti IS DONE")
    from overallpot import overallpot
    overallpot()

    print("16.overallpot IS DONE")
    print(nums)
    return nums
```

Program: assignRNA3Dall.py

```
# this code assign DSSR 3D structure to each RNA in a contact

def assignRNA3Dall():
    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i= 0
    with open(path+'Analyzing/bests3Dall.txt','w+') as fo:
        with open(path+'Clustering/allredun.txt') as fi:
            for iline in fi.readlines():
                ilinelist = iline.split()
                # 0 1        2  3    4  5    6     7          8          9 10  11
                # 3    1A34_B_A 26 1A34 14 2948 2.131 OP1@1:B.U11 N@1:A.GLY14 6
XRY 1.8

                sixlist = ilinelist[7].split(':')
                # query chain ID is like 'B.G668'
                qchID = sixlist[1]


fo.writelines(str(i)+'\t'+iline.strip('\n')+'\t'+'NA'+'\t'+'NA'+'\t'+'NA'+'\t'+'NA'+'\n')
```

Program: assignBorSall.py

```
# this code use best3D.txt and add 'bbone'/'bases' to each line

def assignBorSall():

    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    bbone = ["OP2","OP1","O5'","O4'","O3'","O2'"]
    bases = ['N1','O2','N3','O4','N6','N7','N9','N2','O6','N4','N']
    bbnum, basesnum,space, aform, uncon, uncla = 0,0,0,0,0,0

    with open(path+'Analyzing/bests3Dplusall.txt','w+') as fo:
```

```
        with open(path+'Analyzing/bests3Dall.txt') as fi:
            for iline in fi.readlines():
                ilinelist = iline.split()

                # 0 1 2       3  4    5   6    7     8         9           10 11  12  13   14   15 16
                # 1 4 1A34_B_A 27 1A34 345 2454 2.693 O2'@1:B.A5 OG1@1:A.THR36 6
XRY 1.8 left cW-W A  n
                # 0    3 1A34_B_A 26 1A34 14  2948    2.131 OP1@1:B.U11  N@1:A.GLY14
6  XRY 1.8 NA   NA  NA NA
                try:
                    # get base(='G') and RNAatom(='OP2')
                    preatom = ilinelist[8].split('@')
                    RNAatom = preatom[0]
                    prebase = preatom[1].split('.')
                    prebase2 = prebase[1]
                    base = prebase2[0]

                    # get residue and ratom
                    preresiatom = ilinelist[9].split('@')
                    resiatom = preresiatom[0]
                    preresi = preresiatom[1].split('.')
                    preresi2 = preresi[1]
                    residue = preresi2[0:3]

                    if 'Note: ' in iline:
                        break
                    if len(ilinelist) == 16:
                        ilinelist.append(' ')

                    # preatom = ["O2'", "1:B.A5"]
                    if preatom[0] in bases:

fo.writelines(ilinelist[1]+'\t'+ilinelist[2]+'\t'+base+'\t'+RNAatom+'\t'+residue+'\t'+resiato
m\
                                  +'\t'+ilinelist[13]+'\t'+ilinelist[14]+\

'\t'+ilinelist[15]+'\t'+ilinelist[16]+'\t'+'bases'+'\t'+str(basesnum)+'\n')

                    elif preatom[0] in bbone:

fo.writelines(ilinelist[1]+'\t'+ilinelist[2]+'\t'+base+'\t'+RNAatom+'\t'+residue+'\t'+resiato
m\
                                  +'\t'+ilinelist[13]+'\t'+ilinelist[14]+\

'\t'+ilinelist[15]+'\t'+ilinelist[16]+'\t'+'bbone'+'\t'+str(basesnum)+'\n')
                    # elif (preatom[0] not in bases) and (preatom[0] not in bbone):
```

```
            except IndexError:
                print(iline)
```

Program: overall.py

```
# this code calculates propensities for overall
def overall():
    def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
        newseq1,newseq2 = ","
        with open(reffile) as ff:
            linef = ff.readlines()
            for i in range(0, len(linef)):
                line = linef[i]
                # extend the RNA chain
                if (RNAchainID in line) and (sign == 'RNA'):
                    Rseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                Rseq = Rseq.strip('\n') + nextline
                            elif nextline[0] == '>':
                                newseq1 = oldseq+Rseq.strip('\n')
                                break

                        except IndexError:
                            newseq1 = oldseq+Rseq.strip('\n')
                            break
                    return newseq1

                # extend the protein chain
                elif (proteinchainID in line) and (sign == 'protein'):
                    pseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                pseq = pseq.strip('\n') + nextline
                            elif nextline[0] == '>':
                                newseq2 = oldseq + pseq.strip('\n')
                                break
```

```
                except IndexError:
                    newseq2 = oldseq + pseq.strip('\n')
                    break
            return newseq2
        return oldseq

    def calpropensity(RNAseq,proteinseq,outputfile,apair):

        aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
        bases = ['A','C','G','U']
        aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
              'GLN','ARG','SER','THR','VAL','TRP','TYR']
        # R = residic[ARG]
        with open(outputfile,'w+') as fo:
            for i in range(0,20):
                for j in range(0,4):
                    # avoid dividing by zero
                    if proteinseq.count(aminos[i])*RNAseq.count(bases[j])== 0:
                        fo.writelines(aminot[i]+'_'+bases[j]+'\t'+'0.0'+'\n')
                    else:
                        numerator = (apair.count(aminot[i]+'_'+bases[j]))/len(apair)
                        denominator1 = RNAseq.count(bases[j])/len(RNAseq)
                        denominator2 = proteinseq.count(aminos[i])/len(proteinseq)

fo.writelines(aminot[i]+'_'+bases[j]+'\t'+str(numerator/(denominator1*denominator2))+'\
n')

    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i,j,count = 0,0,0
    pre = ''
    # helix > a form > bases > mg > canonical

    with open(path+'Analyzing/bests3Dplusall.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","","

        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')
            # 15   1A9N_Q_B   U  O2 ARG    NE NA NA NA NA bases  0
            # 0 1       2 3   4   5   6     7
            # 4    1A4T_A_B C OP2 ARG NH2 bbone 0

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
```

```
            proteinchainID = elementlist[0]+':'+elementlist[2]

            # add this atom pair to list and extend the sequence
            atompair.append(ilinelist[4]+'_'+ilinelist[2])
            count += 1

            if ilinelist[1] not in combinations:
               combinations.append(ilinelist[1])
               RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','RNA',RNAseq)
               proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)


calpropensity(RNAseq,proteinseq,path+'Analyzing/propensities/overall.txt',atompair)
   print(count)

Program: assignRNA3D.py

# this code assign DSSR 3D structure to each RNA in a contact

def assignRNA3D():

   path = '/Users/kimuratakayuki/Desktop/Thesis/'
   i= 0
   wobblepair = ['GU','UG']

   with open(path+'Analyzing/bests3D.txt','w+') as fo:
      with open(path+'Clustering/allredun.txt') as fi:
         for iline in fi.readlines():
            ilinelist = iline.split()
            # 0 1       2  3    4  5    6     7          8          9 10  11
            # 3    1A34_B_A 26 1A34 14 2948 2.131 OP1@1:B.U11 N@1:A.GLY14 6
XRY 1.8

            sixlist = ilinelist[7].split(':')
            # query chain ID is like 'B.G668'
            qchID = sixlist[1]
            try:
               with
open('/Users/kimuratakayuki/Desktop/PRat77/Analyzing/DSSRout/'+ilinelist[3].lower()+
'.cif.out') as fm:
                  readstart,findstart,escape = 0,0,0
                  for mline in fm.readlines():
                     # search the keywords and start parsing lines
                     if mline[2:7] == 'helix':
```

```
                findstart = 1
            elif findstart == 1 and mline[6:16] == 'helix-form':
                readstart = 1
                stepline = mline
            elif readstart == 1:
                if (mline != '\n') and ('----' not in mline) and (mline[3] != ' '):
                    if '********' in mline:
                        break
                    row = mline.split()

                    #    row[1]='1:B.G201',row[2]='1:B.C220'
                    # or row[1]=  'B.G201',row[2]=  'B.C220'
                    base1 = row[1].split('.')
                    base2 = row[2].split('.')

                    chainid1pre = base1[0]
                    chainid1    = chainid1pre[-1:]
                    prebaseid1     = base1[1]
                    baseid1 = prebaseid1[0]

                    chainid2pre = base2[0]
                    chainid2    = chainid2pre[-1:]
                    prebaseid2     = base2[1]
                    baseid2 = prebaseid2[0]

                    if (baseid1 + baseid2) in wobblepair:
                        wobble = 'w'
                    else:
                        wobble = 'n'

                    if qchID == chainid1+'.'+prebaseid1:
                        i += 1
                        if stepline[17+int(row[0])] == '\n':
                            fo.writelines(str(i)+'\t'+iline.strip('\n')+'\t'+'left
'+'\t'+row[7]+'\t'+'e'+'\t'+wobble+'\n')
                        else:
                            fo.writelines(str(i)+'\t'+iline.strip('\n')+'\t'+'left
'+'\t'+row[7]+'\t'+stepline[17+int(row[0])]+'\t'+wobble+'\n')
                        break
                    elif qchID == chainid2+'.'+prebaseid2:
                        i += 1
                        if stepline[17+int(row[0])] == '\n':

fo.writelines(str(i)+'\t'+iline.strip('\n')+'\t'+'right'+'\t'+row[7]+'\t'+'e'+'\t'+wobble+'\n')
                        else:
```

```
fo.writelines(str(i)+'\t'+iline.strip('\n')+'\t'+'right'+'\t'+row[7]+'\t'+stepline[17+int(row[0])]
+'\t'+wobble+'\n')
                                break
            except IOError:
                print('IOError @ '+iline)
```

Program: assignBorS.py

```
# this code use best3D.txt and add 'bbone'/'bases' to each line
def assignBorS():

    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    bbone = ["OP2","OP1","O5'","O4'","O3'","O2'"]
    bases = ['N1','O2','N3','O4','N6','N7','N9','N2','O6',"]
    bbnum, basesnum,space, aform, uncon, uncla = 0,0,0,0,0,0

    with open(path+'Analyzing/bests3Dplus.txt','w+') as fo:
        with open(path+'Analyzing/bests3D.txt') as fi:
            for iline in fi.readlines():
                ilinelist = iline.split()

                # 0 1 2       3  4    5   6    7     8          9            10 11  12  13   14   15 16
                # 1 4 1A34_B_A 27 1A34 345 2454 2.693 O2'@1:B.A5 OG1@1:A.THR36 6
XRY 1.8 left cW-W A  n

                # get base(='G') and RNAatom(='OP2')
                preatom = ilinelist[8].split('@')
                RNAatom = preatom[0]
                prebase = preatom[1].split('.')
                prebase2 = prebase[1]
                base = prebase2[0]

                # get residue and ratom
                preresiatom = ilinelist[9].split('@')
                resiatom = preresiatom[0]
                preresi = preresiatom[1].split('.')
                preresi2 = preresi[1]
                residue = preresi2[0:3]

                if 'Note: ' in iline:
                    break
                if len(ilinelist) == 16:
                    ilinelist.append(' ')

                # preatom = ["O2'", "1:B.A5"]
                if preatom[0] in bases:
```

```
fo.writelines(ilinelist[0]+'\t'+ilinelist[2]+'\t'+base+'\t'+RNAatom+'\t'+residue+'\t'+resiato
m\
                        +'\t'+ilinelist[13]+'\t'+ilinelist[14]+\
                        '\t'+ilinelist[15]+'\t'+ilinelist[16]+'\t'+'bases'+'\t'+str(basesnum)+'\n')
                if ilinelist[16] == ' ':
                    space += 1
                elif ilinelist[16] == 'x':
                    uncon += 1
                elif ilinelist[16] == '.':
                    uncla += 1
                elif ilinelist[16] == 'A':
                    aform += 1

                basesnum += 1
            elif preatom[0] in bbone:

fo.writelines(ilinelist[0]+'\t'+ilinelist[2]+'\t'+base+'\t'+RNAatom+'\t'+residue+'\t'+resiato
m\
                        +'\t'+ilinelist[13]+'\t'+ilinelist[14]+\
                        '\t'+ilinelist[15]+'\t'+ilinelist[16]+'\t'+'bbone'+'\t'+str(basesnum)+'\n')
                bbnum += 1

            #
print(ilinelist[0]+'\t'+ilinelist[2]+'\t'+base+'\t'+RNAatom+'\t'+residue+'\t'+ratom\
            #                 +'\t'+ilinelist[13]+'\t'+ilinelist[14]+\
            #
'\t'+ilinelist[15]+'\t'+ilinelist[16]+'\t'+'bases'+'\t'+str(basesnum)+'\n')
```

Program: addmajor.py

```
# this code assign 'major' without considering RNA structure
# just from the base specimen and atom position
def addmajor():

    path = '/Users/kimuratakayuki/Desktop/Thesis/Analyzing/'

    Cmajor = ['N4']
    Gmajor = ['N7','O6',]
    Umajor = ['O4']
    Amajor = ['N6','N7']
    WUmajor = ['O4','N3']

    with open(path+'majoradded.txt','w+') as fo:
        with open(path+'bests3Dplus.txt') as fi:
            for ilines in fi.readlines():
```

```
            ilinelist = ilines.split('\t')

            major = 'no'
            if ilinelist[2] == 'G':
               if ilinelist[3] in Gmajor:
                  major = 'major'

            elif ilinelist[2] == 'C':
               if ilinelist[3] in Cmajor:
                  major = 'major'

            elif ilinelist[2] == 'A':
               if ilinelist[3]in Amajor:
                  major = 'major'

            elif ilinelist[2] == 'U':
               if (ilinelist[8] == 'w') and (ilinelist[3] in WUmajor):
                  major = 'major'
               elif (ilinelist[8] == 'n') and (ilinelist[3] in Umajor):
                  major = 'major'

            fo.writelines(ilines.strip('\n')+'\t'+major+'\n')
            major = ''
```

Program: extracthelix.py

```
def extractnothelix():
   path = '/Users/kimuratakayuki/Desktop/Thesis/Analyzing/'
   path2 = '/Users/kimuratakayuki/Desktop/Thesis/Clustering/'

   i = 0
   helix = []

   with open(path+'bests3D.txt') as f:
      for flines in f.readlines():
         flist = flines.split('\t')
         helix.append(flist[8]+flist[9])

   with open(path+'nothelix.txt','w+') as fo:
      with open(path2+'allredun.txt') as fa:
         for falines in fa.readlines():
            falineslist = falines.split('\t')
            if falineslist[7]+falineslist[8] not in helix:
               fo.writelines(str(i)+'\t'+falines)
```

Program: assignBorSnh.py

```
# this code use best3D.txt and add 'bbone'/'bases' to each line

def assignBorSnh():

    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    bbone = ["OP2","OP1","O5'","O4'","O3'","O2'"]
    bases = ['N1','O2','N3','O4','N6','N7','N9','N2','O6','']
    bbnum, basesnum,space, aform, uncon, uncla = 0,0,0,0,0,0

    with open(path+'Analyzing/nothelix2.txt','w+') as fo:
        with open(path+'Analyzing/nothelix.txt') as fi:
            for iline in fi.readlines():
                ilinelist = iline.split()

                # 0 1  2        3   4    5   6    7     8           9            10 11  12  13    14    15 16
                # 5    16 1A4T_A_B 176 1A4T 234 602  2.922 O3'@1:A.C11 ND1@1:B.HIS7
0  XRY 1.8
                # 1 4  1A34_B_A 27  1A34 345 2454 2.693 O2'@1:B.A5  OG1@1:A.THR36 6
XRY 1.8 left cW-W A  n

                # get base(='G') and RNAatom(='OP2')
                preatom = ilinelist[8].split('@')
                RNAatom = preatom[0]
                prebase = preatom[1].split('.')
                prebase2 = prebase[1]
                base = prebase2[0]

                # get residue and ratom
                preresiatom = ilinelist[9].split('@')
                resiatom = preresiatom[0]
                preresi = preresiatom[1].split('.')
                preresi2 = preresi[1]
                residue = preresi2[0:3]

                # preatom = ["O2'", "1:B.A5"]
                if preatom[0] in bases:

fo.writelines(ilinelist[0]+'\t'+ilinelist[2]+'\t'+base+'\t'+RNAatom+'\t'+residue+'\t'+resiato
m\
                              +'\t'+'bases'+'\t'+str(basesnum)+'\n')

                elif preatom[0] in bbone:

fo.writelines(ilinelist[0]+'\t'+ilinelist[2]+'\t'+base+'\t'+RNAatom+'\t'+residue+'\t'+resiato
```

```
m\
                        +'\t'+'bbone'+'\t'+str(basesnum)+'\n')
Program: calcdenomi.py

# this code calculates propensities for
# helix > a form > bases > mg > canonical
def calcdenomi():
    def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
        newseq1,newseq2 = ","
        with open(reffile) as ff:
            linef = ff.readlines()
            for i in range(0, len(linef)):
                line = linef[i]
                # extend the RNA chain
                if (RNAchainID in line) and (sign == 'RNA'):
                    Rseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                Rseq = Rseq.strip('\n') + nextline
                            elif nextline[0] == '>':
                                newseq1 = oldseq+Rseq.strip('\n')
                                break

                        except IndexError:
                            newseq1 = oldseq+Rseq.strip('\n')
                            break
                    return newseq1

                # extend the protein chain
                elif (proteinchainID in line) and (sign == 'protein'):
                    pseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                pseq = pseq.strip('\n') + nextline
                            elif nextline[0] == '>':
                                newseq2 = oldseq + pseq.strip('\n')
                                break
```

```
                except IndexError:
                    newseq2 = oldseq + pseq.strip('\n')
                    break
            return newseq2
        return oldseq

    path = '/Users/kimuratakayuki/Desktop/Thesis/'

    # helix > a form > bases > mg > canonical
    with open(path+'Analyzing/bests3Dplusall.txt') as fi:
        pairlen = 0

        for ilines in fi.readlines():
            pairlen += 1

    with open(path+'Analyzing/bests3Dplusall.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","","

        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
            proteinchainID = elementlist[0]+':'+elementlist[2]
            # add this atom pair to list and extend the sequence
            # 0     1       2 3  4   5 6     7    8 9 10    11   12
            # 11420    4D67_2_Y G O6 HIS N right tm+m x n bases 1783 major
            if ilinelist[1] not in combinations:
                combinations.append(ilinelist[1])
                RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','RNA',RNAseq)
                proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)
        rp = [len(RNAseq),len(proteinseq),pairlen]
        return rp

print(calcdenomi())

Program: sepcalp.py

def sepcalp(k,r2p2,nums):
    if k == 0:
        from calp1_2 import calp1_2
        calp1_2(nums,r2p2)
```

```
    elif k == 1:
        from calp3 import calp3
        calp3(nums,r2p2)
    elif k == 2:
        from calp4_5 import calp4_5
        calp4_5(nums,r2p2)
    elif k == 3:
        from calp6 import calp6
        calp6(nums,r2p2)
    elif k == 4:
        from calp7 import calp7
        calp7(nums,r2p2)
    elif k == 5:
        from calp8_9 import calp8_9
        calp8_9(nums,r2p2)
```

Program: calp1_2.py

```
# this code calculates propensities for
# helix > a form > bases > mg > canonical
def calp1_2(num1_2,r2p2):

    def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
        newseq1,newseq2 = ","
        with open(reffile) as ff:
            linef = ff.readlines()
            for i in range(0, len(linef)):
                line = linef[i]
                # extend the RNA chain
                if (RNAchainID in line) and (sign == 'RNA'):
                    Rseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                Rseq = Rseq.strip('\n') + nextline
                            elif nextline[0] == '>':
                                newseq1 = oldseq+Rseq.strip('\n')
                                break

                        except IndexError:
                            newseq1 = oldseq+Rseq.strip('\n')
                            break
                    return newseq1
```

```
            # extend the protein chain
            elif (proteinchainID in line) and (sign == 'protein'):
                pseq = ''
                for j in range(1,100):
                    try:
                        nextline = linef[i+j]

                        # if not > line, add the line to 'seq'
                        if nextline[0] != '>':
                            pseq = pseq.strip('\n') + nextline
                        elif nextline[0] == '>':
                            newseq2 = oldseq + pseq.strip('\n')
                            break

                    except IndexError:
                        newseq2 = oldseq + pseq.strip('\n')
                        break
                return newseq2
        return oldseq

    def calpropensity(RNAseq,proteinseq,outputfile,apair):

        aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
        bases = ['A','C','G','U']
        aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
              'GLN','ARG','SER','THR','VAL','TRP','TYR']
        # R = residic[ARG]
        with open(outputfile,'w+') as fo:
            for i in range(0,20):
                for j in range(0,4):
                    # avoid dividing by zero
                    if proteinseq.count(aminos[i])*RNAseq.count(bases[j])== 0:
                        fo.writelines(aminot[i]+'_'+bases[j]+'\t'+'0.0'+'\n')
                    else:
                        numerator = (apair.count(aminot[i]+'_'+bases[j]))/r2p2[2]
                        denominator1 = RNAseq.count(bases[j])/r2p2[0]
                        denominator2 = proteinseq.count(aminos[i])/r2p2[1]

fo.writelines(aminot[i]+'_'+bases[j]+'\t'+str(numerator/(denominator1*denominator2))+\

'\t'+'a1:'+'\t'+str(apair.count(aminot[i]+'_'+bases[j]))+'\t'+',a2:'+'\t'+str(r2p2[2])+\

'\t'+',rna1:'+'\t'+str(RNAseq.count(bases[j]))+'\t'+',rna2:'+'\t'+str(r2p2[0])+'\t'+',p1:'+\
```

```
'\t'+str(proteinseq.count(aminos[i]))+'\t'+',p2:'+'\t'+str(r2p2[1])+'\n')

    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i,j = 0,0
    count1, count2 =0,0

    # helix > a form > bases > mg > canonical
    with open(path+'Analyzing/majoradded.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","",""

        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
            proteinchainID = elementlist[0]+':'+elementlist[2]

            # add this atom pair to list and extend the sequence
            # 0     1        2 3  4   5 6     7    8 9 10    11   12
            # 11420    4D67_2_Y G O6 HIS N right tm+m x n bases 1783 major
            if (ilinelist[8] == 'A') and (ilinelist[10] == 'bases'):
                if ilinelist[12].strip('\n') == 'major':
                    atompair.append(ilinelist[4]+'_'+ilinelist[2])
                    count1 += 1
                    if ilinelist[1] not in combinations:
                        combinations.append(ilinelist[1])
                        RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_af.txt','RNA',RNAseq)
                        proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)

                elif ilinelist[12].strip('\n') != 'major':
                    atompair2.append(ilinelist[4]+'_'+ilinelist[2])
                    count2 += 1
                    if ilinelist[1] not in combinations2:
                        combinations2.append(ilinelist[1])
                        RNAseq2 =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_af.txt','RNA',RNAseq2)
                        proteinseq2 =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq2)

        calpropensity(RNAseq,proteinseq,path+'Analyzing/propensities/1_mg.txt',atompair)

calpropensity(RNAseq2,proteinseq2,path+'Analyzing/propensities/2_not_mg.txt',atompai
```

```
r2)
    print('1:'+str(count1))
    print('2:'+str(count2))
    num1_2.append(count1)
    num1_2.append(count2)
    return num1_2
```

Program: calp3.py

```
def calp3(nums,r2p2):

    def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
        newseq1,newseq2 = ","
        with open(reffile) as ff:
            linef = ff.readlines()
            for i in range(0, len(linef)):
                line = linef[i]
                # extend the RNA chain
                if (RNAchainID in line) and (sign == 'RNA'):
                    Rseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                Rseq = Rseq.strip('\n') + nextline
                            elif nextline[0] == '>':
                                newseq1 = oldseq+Rseq.strip('\n')
                                break

                        except IndexError:
                            newseq1 = oldseq+Rseq.strip('\n')
                            break
                    return newseq1

                # extend the protein chain
                elif (proteinchainID in line) and (sign == 'protein'):
                    pseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                pseq = pseq.strip('\n') + nextline
```

```
                    elif nextline[0] == '>':
                        newseq2 = oldseq + pseq.strip('\n')
                        break

                except IndexError:
                    newseq2 = oldseq + pseq.strip('\n')
                    break
            return newseq2
        return oldseq

    def calpropensity(RNAseq,proteinseq,outputfile,apair):

        aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
        bases = ['A','C','G','U']
        aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
            'GLN','ARG','SER','THR','VAL','TRP','TYR']
        # R = residic[ARG]
        with open(outputfile,'w+') as fo:
            for i in range(0,20):
                for j in range(0,4):
                    # avoid dividing by zero
                    if proteinseq.count(aminos[i])*RNAseq.count(bases[j])== 0:
                        fo.writelines(aminot[i]+'_'+bases[j]+'\t'+'0.0'+'\n')
                    else:
                        numerator = (apair.count(aminot[i]+'_'+bases[j]))/r2p2[2]
                        denominator1 = RNAseq.count(bases[j])/r2p2[0]
                        denominator2 = proteinseq.count(aminos[i])/r2p2[1]


fo.writelines(aminot[i]+'_'+bases[j]+'\t'+str(numerator/(denominator1*denominator2))+\

'\t'+'a1:'+'\t'+str(apair.count(aminot[i]+'_'+bases[j]))+'\t'+',a2:'+'\t'+str(len(apair))+\

'\t'+',rna1:'+'\t'+str(RNAseq.count(bases[j]))+'\t'+',rna2:'+'\t'+str(len(RNAseq))+'\t'+',p1:'+
\

'\t'+str(proteinseq.count(aminos[i]))+'\t'+',p2:'+'\t'+str(len(proteinseq))+'\n')

    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i,j = 0,0
    count = 0

    with open(path+'Analyzing/majoradded.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","","
```

```
        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
            proteinchainID = elementlist[0]+':'+elementlist[2]

            # add this atom pair to list and extend the sequence
            # 0     1       2 3  4   5 6     7    8 9 10    11   12
            # 11420    4D67_2_Y G O6 HIS N right tm+m x n bases 1783 major
            if (ilinelist[8] == 'A') and (ilinelist[10] == 'bbone'):
                atompair.append(ilinelist[4]+'_'+ilinelist[2])
                count += 1
                if ilinelist[1] not in combinations:
                    combinations.append(ilinelist[1])
                    RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_af.txt','RNA',RNAseq)
                    proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)


calpropensity(RNAseq,proteinseq,path+'Analyzing/propensities/3_aform_bb.txt',atompai
r)
    print('3:'+str(count))
    nums.append(count)
    return nums

Program: calp4_5.py

# this code calculates propensities for
# helix > a form > bases > mg > canonical
def calp4_5(nums,r2p2):

    def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
        newseq1,newseq2 = ","
        with open(reffile) as ff:
            linef = ff.readlines()
            for i in range(0, len(linef)):
                line = linef[i]
                # extend the RNA chain
                if (RNAchainID in line) and (sign == 'RNA'):
                    Rseq = ''
                    for j in range(1,100):
                        try:
```

```
                    nextline = linef[i+j]

                    # if not > line, add the line to 'seq'
                    if nextline[0] != '>':
                        Rseq = Rseq.strip('\n') + nextline
                    elif nextline[0] == '>':
                        newseq1 = oldseq+Rseq.strip('\n')
                        break

                except IndexError:
                    newseq1 = oldseq+Rseq.strip('\n')
                    break
            return newseq1

        # extend the protein chain
        elif (proteinchainID in line) and (sign == 'protein'):
            pseq = ''
            for j in range(1,100):
                try:
                    nextline = linef[i+j]

                    # if not > line, add the line to 'seq'
                    if nextline[0] != '>':
                        pseq = pseq.strip('\n') + nextline
                    elif nextline[0] == '>':
                        newseq2 = oldseq + pseq.strip('\n')
                        break

                except IndexError:
                    newseq2 = oldseq + pseq.strip('\n')
                    break
            return newseq2
    return oldseq

def calpropensity(RNAseq,proteinseq,outputfile,apair):

    aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
    bases = ['A','C','G','U']
    aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
          'GLN','ARG','SER','THR','VAL','TRP','TYR']
    # R = residic[ARG]
    with open(outputfile,'w+') as fo:
        for i in range(0,20):
            for j in range(0,4):
                # avoid dividing by zero
```

```
                if proteinseq.count(aminos[i])*RNAseq.count(bases[j])== 0:
                    fo.writelines(aminot[i]+'_'+bases[j]+'\t'+'0.0'+'\n')
                else:
                    numerator = (apair.count(aminot[i]+'_'+bases[j]))/r2p2[2]
                    denominator1 = RNAseq.count(bases[j])/r2p2[0]
                    denominator2 = proteinseq.count(aminos[i])/r2p2[1]


fo.writelines(aminot[i]+'_'+bases[j]+'\t'+str(numerator/(denominator1*denominator2))+\

'\t'+'a1:'+'\t'+str(apair.count(aminot[i]+'_'+bases[j]))+'\t'+',a2:'+'\t'+str(len(apair))+\

'\t'+',rna1:'+'\t'+str(RNAseq.count(bases[j]))+'\t'+',rna2:'+'\t'+str(len(RNAseq))+'\t'+',p1:'+
\

'\t'+str(proteinseq.count(aminos[i]))+'\t'+',p2:'+'\t'+str(len(proteinseq))+'\n')
    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i,j,count4,count5 = 0,0,0,0

    # helix > a form > bases > mg > canonical

    with open(path+'Analyzing/majoradded.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","",""

        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
            proteinchainID = elementlist[0]+':'+elementlist[2]

            if (ilinelist[8] != 'A') and (ilinelist[10] == 'bases'):
                if ilinelist[12].strip('\n') == 'major' and ilinelist[7] == 'cW-W':
                    atompair.append(ilinelist[4]+'_'+ilinelist[2])
                    count4 += 1
                    if ilinelist[1] not in combinations:
                        combinations.append(ilinelist[1])
                        RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_na.txt','RNA',RNAseq)
                        proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)

                elif ilinelist[12].strip('\n') == 'major' and ilinelist[7] != 'cW-W':
                    atompair2.append(ilinelist[4]+'_'+ilinelist[2])
```

```
                count5 += 1
                if ilinelist[1] not in combinations2:
                    combinations2.append(ilinelist[1])
                    RNAseq2 =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_na.txt','RNA',RNAseq2)
                    proteinseq2 =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq2)


calpropensity(RNAseq,proteinseq,path+'Analyzing/propensities/4_mglike_cano.txt',atom
pair)

calpropensity(RNAseq2,proteinseq2,path+'Analyzing/propensities/5_mglike_noncano.txt
',atompair2)
    print('4:'+str(count4))
    print('5:'+str(count5))
    nums.append(count4)
    nums.append(count5)
    return nums
```

Program: calp6.py

```
# this code calculates propensities for
# helix > a form > bases > mg > canonical
def calp6(nums,r2p2):

    def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
        newseq1,newseq2 = ","
        with open(reffile) as ff:
            linef = ff.readlines()
            for i in range(0, len(linef)):
                line = linef[i]
                # extend the RNA chain
                if (RNAchainID in line) and (sign == 'RNA'):
                    Rseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]

                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                Rseq = Rseq.strip('\n') + nextline
                            elif nextline[0] == '>':
                                newseq1 = oldseq+Rseq.strip('\n')
                                break
```

```
                except IndexError:
                    newseq1 = oldseq+Rseq.strip('\n')
                    break
            return newseq1

        # extend the protein chain
        elif (proteinchainID in line) and (sign == 'protein'):
            pseq = ''
            for j in range(1,100):
                try:
                    nextline = linef[i+j]

                    # if not > line, add the line to 'seq'
                    if nextline[0] != '>':
                        pseq = pseq.strip('\n') + nextline
                    elif nextline[0] == '>':
                        newseq2 = oldseq + pseq.strip('\n')
                        break

                except IndexError:
                    newseq2 = oldseq + pseq.strip('\n')
                    break
            return newseq2
    return oldseq

def calpropensity(RNAseq,proteinseq,outputfile,apair):

    aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
    bases = ['A','C','G','U']
    aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
          'GLN','ARG','SER','THR','VAL','TRP','TYR']
    # R = residic[ARG]
    with open(outputfile,'w+') as fo:
        for i in range(0,20):
            for j in range(0,4):
                # avoid dividing by zero
                if proteinseq.count(aminos[i])*RNAseq.count(bases[j])== 0:
                    fo.writelines(aminot[i]+'_'+bases[j]+'\t'+'0.0'+'\n')
                else:
                    numerator = (apair.count(aminot[i]+'_'+bases[j]))/r2p2[2]
                    denominator1 = RNAseq.count(bases[j])/r2p2[0]
                    denominator2 = proteinseq.count(aminos[i])/r2p2[1]


fo.writelines(aminot[i]+'_'+bases[j]+'\t'+str(numerator/(denominator1*denominator2))+\
```

```
'\t'+'a1:'+'\t'+str(apair.count(aminot[i]+'_'+bases[j]))+'\t'+',a2:'+'\t'+str(len(apair))+\

'\t'+',rna1:'+'\t'+str(RNAseq.count(bases[j]))+'\t'+',rna2:'+'\t'+str(len(RNAseq))+'\t'+',p1:'+
\

'\t'+str(proteinseq.count(aminos[i]))+'\t'+',p2:'+'\t'+str(len(proteinseq))+'\n')
    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i,j,count = 0,0,0

    # helix > a form > bases > mg > canonical

    with open(path+'Analyzing/majoradded.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","","

        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
            proteinchainID = elementlist[0]+':'+elementlist[2]

            # add this atom pair to list and extend the sequence
            # 0     1        2 3  4   5 6     7    8 9 10    11   12
            # 11420    4D67_2_Y G O6 HIS N right tm+m x n bases 1783 major
            if (ilinelist[8] != 'A') and (ilinelist[10] == 'bases'):
                if ilinelist[12].strip('\n') != 'major':
                    atompair.append(ilinelist[4]+'_'+ilinelist[2])
                    count += 1
                    if ilinelist[1] not in combinations:
                        combinations.append(ilinelist[1])
                        RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_na.txt','RNA',RNAseq)
                        proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)


calpropensity(RNAseq,proteinseq,path+'Analyzing/propensities/6_notmglike.txt',atompai
r)
    print('6:'+str(count))
    nums.append(count)
    return nums
```

Program: calp7.py

```
def calp7(nums,r2p2):

   def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
      newseq1,newseq2 = ","
      with open(reffile) as ff:
         linef = ff.readlines()
         for i in range(0, len(linef)):
            line = linef[i]
            # extend the RNA chain
            if (RNAchainID in line) and (sign == 'RNA'):
               Rseq = ''
               for j in range(1,100):
                  try:
                     nextline = linef[i+j]

                     # if not > line, add the line to 'seq'
                     if nextline[0] != '>':
                        Rseq = Rseq.strip('\n') + nextline
                     elif nextline[0] == '>':
                        newseq1 = oldseq+Rseq.strip('\n')
                        break

                  except IndexError:
                     newseq1 = oldseq+Rseq.strip('\n')
                     break
               return newseq1

            # extend the protein chain
            elif (proteinchainID in line) and (sign == 'protein'):
               pseq = ''
               for j in range(1,100):
                  try:
                     nextline = linef[i+j]

                     # if not > line, add the line to 'seq'
                     if nextline[0] != '>':
                        pseq = pseq.strip('\n') + nextline
                     elif nextline[0] == '>':
                        newseq2 = oldseq + pseq.strip('\n')
                        break

                  except IndexError:
                     newseq2 = oldseq + pseq.strip('\n')
                     break
```

```
                return newseq2
            return oldseq

    def calpropensity(RNAseq,proteinseq,outputfile,apair):

        aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
        bases = ['A','C','G','U']
        aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
                'GLN','ARG','SER','THR','VAL','TRP','TYR']
        # R = residic[ARG]
        with open(outputfile,'w+') as fo:
            for i in range(0,20):
                for j in range(0,4):
                    # avoid dividing by zero
                    if proteinseq.count(aminos[i])*RNAseq.count(bases[j])== 0:
                        fo.writelines(aminot[i]+'_'+bases[j]+'\t'+'0.0'+'\n')
                    else:
                        numerator = (apair.count(aminot[i]+'_'+bases[j]))/r2p2[2]
                        denominator1 = RNAseq.count(bases[j])/r2p2[0]
                        denominator2 = proteinseq.count(aminos[i])/r2p2[1]


fo.writelines(aminot[i]+'_'+bases[j]+'\t'+str(numerator/(denominator1*denominator2))+\

'\t'+'a1:'+'\t'+str(apair.count(aminot[i]+'_'+bases[j]))+'\t'+',a2:'+'\t'+str(len(apair))+\

'\t'+',rna1:'+'\t'+str(RNAseq.count(bases[j]))+'\t'+',rna2:'+'\t'+str(len(RNAseq))+'\t'+',p1:'+
\

'\t'+str(proteinseq.count(aminos[i]))+'\t'+',p2:'+'\t'+str(len(proteinseq))+'\n')
    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i,j,count = 0,0,0

    with open(path+'Analyzing/majoradded.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","","

        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
            proteinchainID = elementlist[0]+':'+elementlist[2]
```

```
        # add this atom pair to list and extend the sequence
        # 0     1       2 3  4   5 6     7    8 9 10    11   12
        # 11420    4D67_2_Y G O6 HIS N right tm+m x n bases 1783 major
        if (ilinelist[8] != 'A') and (ilinelist[10] == 'bbone'):
            atompair.append(ilinelist[4]+'_'+ilinelist[2])
            count += 1
            if ilinelist[1] not in combinations:
                combinations.append(ilinelist[1])
                RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_na.txt','RNA',RNAseq)
                proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)


calpropensity(RNAseq,proteinseq,path+'Analyzing/propensities/7_no_aform_bb.txt',atom
pair)
    print('7:'+str(count))
    nums.append(count)
    return nums
```

Program: calp8_9.py

```
# this code calculates propensities for
# helix > a form > bases > mg > canonical
def calp8_9(nums,r2p2):

    def extendtwochains(RNAchainID,proteinchainID,reffile,sign,oldseq):
        newseq1,newseq2 = ","
        aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
        bases = ['A','C','G','U']
        with open(reffile) as ff:
            linef = ff.readlines()
            for i in range(0, len(linef)):
                line = linef[i]
                # extend the RNA chain
                if (RNAchainID in line) and (sign == 'RNA'):
                    Rseq = ''
                    for j in range(1,100):
                        try:
                            nextline = linef[i+j]
                            # if not > line, add the line to 'seq'
                            if nextline[0] != '>':
                                if nextline[0] in bases:
                                    Rseq = Rseq.strip('\n') + nextline
                            elif nextline[0] == '>' :
                                newseq1 = oldseq+Rseq.strip('\n')
```

```
                    break

                except IndexError:
                    newseq1 = oldseq+Rseq.strip('\n')
                    break
                except EOFError:
                    newseq1 = oldseq+Rseq.strip('\n')
                    break

            return newseq1

        # extend the protein chain
        elif (proteinchainID in line) and (sign == 'protein'):
            pseq = ''
            for j in range(1,100):
                try:
                    nextline = linef[i+j]

                    # if not > line, add the line to 'seq'
                    if nextline[0] != '>':
                        if nextline[0] in aminos:
                            pseq = pseq.strip('\n') + nextline
                        elif nextline is None:
                            newseq2 = oldseq+pseq.strip('\n')
                            break
                    elif nextline[0] == '>':
                        newseq2 = oldseq + pseq.strip('\n')
                        break

                except IndexError:
                    newseq2 = oldseq + pseq.strip('\n')
                    break
            return newseq2
    return oldseq

def calpropensity(RNAseq,proteinseq,outputfile,apair):

    aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
    bases = ['A','C','G','U']
    aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
        'GLN','ARG','SER','THR','VAL','TRP','TYR']
    # R = residic[ARG]
    with open(outputfile,'w+') as fo:
        for i in range(0,20):
            for j in range(0,4):
```

```
                # avoid dividing by zero
                if proteinseq.count(aminos[i])*RNAseq.count(bases[j])== 0:
                    fo.writelines(aminot[i]+'_'+bases[j]+'\t'+'0.0'+'\n')
                else:
                    numerator = (apair.count(aminot[i]+'_'+bases[j]))/r2p2[2]
                    denominator1 = RNAseq.count(bases[j])/r2p2[0]
                    denominator2 = proteinseq.count(aminos[i])/r2p2[1]


fo.writelines(aminot[i]+'_'+bases[j]+'\t'+str(numerator/(denominator1*denominator2))+\

'\t'+'a1:'+'\t'+str(apair.count(aminot[i]+'_'+bases[j]))+'\t'+',a2:'+'\t'+str(len(apair))+\

'\t'+',rna1:'+'\t'+str(RNAseq.count(bases[j]))+'\t'+',rna2:'+'\t'+str(len(RNAseq))+'\t'+',p1:'+
\

'\t'+str(proteinseq.count(aminos[i]))+'\t'+',p2:'+'\t'+str(len(proteinseq))+'\n')
    path = '/Users/kimuratakayuki/Desktop/Thesis/'
    i,j = 0,0
    count8,count9 = 0,0
    # helix > a form > bases > mg > canonical

    with open(path+'Analyzing/nothelix2.txt') as fi:
        atompair,combinations,combinations2,atompair2 = [],[],[],[]
        proteinseq, RNAseq, proteinseq2, RNAseq2 = ","","","

        for ilines in fi.readlines():
            ilinelist = ilines.split('\t')

            # get chain ID
            elementlist = ilinelist[1].split('_')
            RNAchainID = elementlist[0]+':'+elementlist[1]
            proteinchainID = elementlist[0]+':'+elementlist[2]

            # add this atom pair to list and extend the sequence
            if ilinelist[6] == 'bases':
                atompair.append(ilinelist[4]+'_'+ilinelist[2])  # ARG_G
                count8 += 1
                if ilinelist[1] not in combinations:
                    combinations.append(ilinelist[1])
                    RNAseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_nh.txt','RNA',RNAseq)
                    proteinseq =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq)

            elif ilinelist[6] == 'bbone':
```

```
            atompair2.append(ilinelist[4]+'_'+ilinelist[2])
            count9 += 1
            if ilinelist[1] not in combinations2:
               combinations2.append(ilinelist[1])
               RNAseq2 =
extendtwochains(RNAchainID,proteinchainID,path+'fasta_nh.txt','RNA',RNAseq2)
               proteinseq2 =
extendtwochains(RNAchainID,proteinchainID,path+'fasta.txt','protein',proteinseq2)


calpropensity(RNAseq,proteinseq,path+'Analyzing/propensities/8_nothelix_bases.txt',ato
mpair)

calpropensity(RNAseq2,proteinseq2,path+'Analyzing/propensities/9_nothelix_bb.txt',ato
mpair2)
   print('8:'+str(count8))
   print('9:'+str(count9))
   nums.append(count8)
   nums.append(count9)
   return nums

Program: potenti.py

def potenti():
   import os,math

   path = '/Users/kimuratakayuki/Desktop/Thesis/Analyzing/'
   i = 1
   RT = 0.59
   pot = 0

   for filename in os.listdir(path+'propensities/'):
      if str(i)+'_' in filename:
         i += 1
         with open(path+'0/'+filename,'w+') as fo:
            with open(path+'propensities/'+filename) as fp:
               for plines in fp.readlines():
                  plist = plines.split('\t')
                  pro = float(plist[1])
                  if pro != 0:
                     pot = RT*math.log(pro)
                  elif pro == 0:
                     pro = 0.001
                     pot = RT*math.log(pro)

                  fo.writelines(plist[0]+'\t'+str(pot)+'\n')
```

Program: correct.py

```
def correct(rnafile,rchain,protfile,pchain,compfile,cchain1,cchain2):
    # this code delete unnecessary lines and change chain names
    # ex.)   rnafile  :  1ABC.pdb , rchain : a
    #        protfile :  2ADE.pdb , pchain : c
    #        compfile :  3EDF.pdb , cchain1: b, cchain2:t

    import os
    path = '/Users/kimuratakayuki/Desktop/Thesis/FTDock/structures/'

    # ATOM   2084  C   MET P   1      3.661   7.340  -5.524  1.00 12.96           M 13
    # ATOM   2085  O   MET P   1      3.663   6.397  -6.323  1.00 13.08           M 13

    # modify RNA file
    with open(path+rnafile[0:4]+'_r.pdb','w+') as rfo:
        with open(path+rnafile) as rf:
            start = 0
            for lines in rf.readlines():
                if lines[21].upper() in rchain.upper():
                    if lines[17:19] == '  ' and lines[0:4]=='ATOM':
                        rfo.writelines(lines[:21]+'R'+lines[22:])
                        start = 1
                    elif start == 1 and 'TER' in lines[0:3]:
                        break

    # modify protein file
    with open(path+protfile[0:4]+'_p.pdb','w+') as pfo:
        with open(path+protfile) as pf:
            start = 0
            for lines in pf.readlines():
                if lines[21].upper() == pchain.upper() and lines[17:19] != '  ' and
lines[0:4]=='ATOM':
                    # if int(lines[22:26]) > 800:
                    #     lines = lines[0:22]+str(int(lines[22:26])-800).rjust(4)+lines[26:]
                    pfo.writelines(lines[:21]+'P'+lines[22:])
                    start = 1
                elif start == 1 and lines[0:3] == 'TER':
                    break

    # modify comp file
    with open(path+compfile[0:4]+'_p_r.pdb','w+') as cfo:
        with open(path+compfile) as cf:
            tercount,rnaend,protend,rnastart,protstart = 0,0,0,0,0
            for lines in cf.readlines():
```

```
            if lines[21].upper() == cchain1.upper() and lines[17:19] != '  ' and
lines[0:4]=='ATOM' and rnaend == 0:
                # if int(lines[22:26]) > 800:
                #     lines = lines[0:22]+str(int(lines[22:26])-800).rjust(4)+lines[26:]
                cfo.writelines(lines[:21]+'P'+lines[22:])
                rnastart = 1
            elif rnastart == 1 and lines[0:3] == 'TER':
                rnaend = 1
                tercount += 1
                continue

            elif lines[21].upper() == cchain2.upper() and lines[17:19] == '  ' and
lines[0:4]=='ATOM' and protend == 0:
                cfo.writelines(lines[:21]+'R'+lines[22:])
                protstart = 1
            elif protstart == 1 and lines[0:3] == 'TER':
                tercount += 1
                protend = 1
                continue
```

Program: assign3D.py

```
# this code prepare 'master.txt' that has
# 3D structure info of RNA  (< DSSR.out, ligand.pdb)
def assign3D(rnafile):
    print('****  ASSIGN3D STARTED  ****')
    path = '/Users/kimuratakayuki/Desktop/Thesis/FTDock/'
    bbone = ["OP2","OP1","O5'","O4'","O3'","O2'"]
    bases = ['N1','O2','N3','O4','N6','N7','N9','N2','O6']
    Cmajor,Gmajor,Umajor,Amajor,WUmajor,wobblepair  =
['N4'],['N7','O6',],['O4'],['N6','N7'],['O4','N3'],['GU','UG']
    canonical = 0
    stepline = ''
    with open(path+'master.txt','w+') as fo:
        with open(path+'structures/'+rnafile) as fr:
            for rlines in fr.readlines():
                atomid = rlines[11:17].replace(' ','')
                baseID2 = (rlines[19]+rlines[22:26]).replace(' ','') #G140
                baseID = baseID2[:1]  # G
                findstart,readstart,category,helix,wobble,rownum,pairlinestart = 0,0,0,0,0,0,0
                with open(path+'DSSR.out') as fi:
                    # check whether the atom of the line is in helix
                    for lines in fi.readlines():
                        if findstart*readstart == 1:
                            if lines == '\n':
                                findstart,readstart = 0,0
```

```
            elif lines !='\n':
                row = lines.split()
                if len(row) >= 5:
                    base1 = row[1].split('.') # G139
                    base2 = row[2].split('.') # C123
                    if baseID2 == base1[1] or baseID2 == base2[1]:
                        helix = 1
                        canonical = lines[66:71]
                        rownum = str(lines[0:4]).replace(' ','')
                        if base1[1]+base2[1] in wobblepair:
                            wobble = 1
        elif findstart == 1 and lines[6:11] == 'helix':
            readstart = 1
            stepline = lines
        elif lines[2:7] == 'helix':
            findstart = 1

if helix == 0:
    if atomid in bases:
        category = 8
    elif atomid in bbone:
        category = 9
elif helix == 1:
    try:
        formtype = stepline[int(rownum)+17]
    except IndexError:
        formtype = ' '
    if formtype == 'A':
        if atomid not in bases:
            category = 3

        elif atomid in bases:
            if baseID == 'G' and atomid in Gmajor:
                category = 1
            elif baseID == 'C' and atomid in Cmajor:
                category = 1
            elif baseID == 'A' and atomid in Amajor:
                category = 1
            elif baseID == 'U':
                if atomid in Umajor and wobble == 0:
                    category = 1
                elif atomid in WUmajor and wobble == 1:
                    category = 1
                else:
                    category = 2
            else:
```

```
                    category = 2

            elif formtype != 'A':
                if atomid not in bases:
                    category = 7

                elif atomid in bases:
                    if (baseID == 'G' and atomid in Gmajor) or \
                      (baseID == 'C' and atomid in Cmajor)or \
                      (baseID == 'A' and atomid in Amajor):
                       if 'cW-W' in canonical:
                          category = 4
                       else:
                          category = 5
                    elif baseID == 'U':
                       if (atomid in Umajor and wobble == 0) or \
                         (atomid in WUmajor and wobble == 1):
                          if 'cW-W' in canonical:
                             category = 4
                          else:
                             category = 5
                       else:
                          category = 6
                    else:
                       category = 6
          fo.writelines(rlines[0:27]+str(category)+'\n')
print('****  ASSIGN3D COMPLETED  ****')

# For each atom in the RNA(****.pdb), assign 1-9 categories
# 0. assign baseID like 'G12'
# 1. helix or not helix
# 2. 1 -> if not helix, bases(8) or backbones(9)
# 3. 1 -> if helix, A-form or not
# 4. 3 -> if A-form, bases or backbone(3)
# 5. 4 -> if bases, major groove(1) or not(2)
# 6. 3 -> if not A-form, bases or backbone(7)
# 7. 6 -> if bases, mglike or not mglike(6)
# 8. 7 -> if mglike, canonical bp(4) or non canonical(5)
```

Program: RMSD.py

```
# this code calculates RMSD of each poses using pymol command 'align'
##  THIS CODE HAS TO BE RUN BY TYPING 'pymol RMSD.py' from terminal
## move to the directory you have RMSD.py (this file), then type
## ****  pymol -cqr RMSD.py >> ../../FTDock/RMSD.out  ***
#  *  change the chain ID before running this file
```

```
import __main__
__main__.pymol_argv = ['pymol', '-qc']
import pymol
import os

pymol.finish_launching()

resultlist = []
path = '/Users/kimuratakayuki/Desktop/Thesis/FTDock/'

# get complexname from a txt file
with open('/Users/kimuratakayuki/Desktop/Thesis/FTDock/threefiles.txt') as fi:
    fline = fi.readline()
    flist = fline.split('\t')
    complex = flist[2]

complexfile = path+'structures/'+complex

for filename in
os.listdir('/Users/kimuratakayuki/Desktop/Thesis/python/8.apply_potentials/'):
    if '.D' not in filename and 'Comp' in filename:
        posefile =
'/Users/kimuratakayuki/Desktop/Thesis/python/8.apply_potentials/'+filename
        pymol.cmd.do('load %s , pose' % posefile)
        pymol.cmd.do('load %s , complex' % complexfile)

        # align peptides
        pymol.cmd.do('align pose and name CA and chain P, complex and name CA and
chain P')
        pymol.cmd.do('select ou, /complex//R//')
        pymol.cmd.do('select co, /pose//R//')
        pymol.cmd.do('rms_cur ou,co')
        pymol.cmd.do('delete %s' % (posefile))
        pymol.cmd.do('delete %s'% (complexfile))
        pymol.cmd.do('delete complex')
        pymol.cmd.do('delete pose')
        pymol.cmd.do('delete ou')
        pymol.cmd.do('delete co')
```

Program: RMSD2.py

```
def RMSD2(xpath,cname):
    path = '/Users/kimuratakayuki/Desktop/Thesis/FTDock/'
    with open(xpath+'rmsd/'+cname+'/RMSD2.out','w+') as fo:
        infile = open(path+'RMSD.out','r')
```

```
        for line in infile:
            if ', pose' in line:
                flist = line.split('/')
                last = flist[-1]
                filenm = last.replace(' , pose','')
                filenm = filenm.replace('\n','')
            elif 'rms_cur' in line:
                nexline = next(infile)
                if 'r:Noato' in nexline[17:26].replace(' ','') :
                    continue
                rms = nexline[17:26].replace(' ','')
                fo.writelines(filenm.replace('\n','')+'\t'+str(rms)+'\n')

    print('****  RMSD2 COMPLETED  ****')
```

Program: calpot_multi.py

```
    # this code apply potentials to each pose
    # potential files + master.txt + pose files -> scores.txt
    # the output file would be like below
    # 1.pose filename  2.score  3.RMSD

def calpot(xpath,potcategory,rangenum,out_q,potmode,Cname):   # category 0-10, 10 is
overall, 0 is full set
    path = '/Users/kimuratakayuki/Desktop/Thesis/FTDock/'
    path2 = '/Users/kimuratakayuki/Desktop/Thesis/Analyzing/'
    # xpath = '/Volumes/Transcend/' or '/Users/kimuratakayuki/Desktop/PRat_asa/stored/'
    # xpath + 'hbonds/'+Cname+'/' or 'rmsd/' + Cname+'/'
    import os, shutil
    group,pre1,pre2,state = ","","","
    cate = [0,0,0,0,0,0,0,0,0,0]
    import math
    with open(path+str(rangenum[0])+'scores.txt','w+') as fo:
        for filename in os.listdir(xpath + 'hbonds/'+Cname+'/'):
            if 'Complex' in filename:
                filenum = int(filename.replace('Complex_','').replace('g.pdb.hb',''))
                if filenum in range(rangenum[0],rangenum[1]+1):
                    scoresum,combscore = 0,0
                    cate_native = [0,0,0,0,0,0,0,0,0,0]
                    rmsd = 0

                    with open(xpath + 'hbonds/'+Cname+'/'+filename) as fc:
                        for lines in fc.readlines():
                            if lines[0] != ' ':
                                continue
                            # pick the contact between RNA-protein
```

```
                pre_ele1,pre_ele2 = ","

                # 51    78  #50    p    3.106 O/N O@P.ALA384 N@P.GLN388
                linelist = lines.split()
                # 0      pre_elelist         '"           14           15
                # " 51    78  #50    p    3","106","O/N","O@P.ALA384",
"N@P.GLN388"
                atomID1 = linelist[0]
                atomID2 = linelist[1]
                atomnafl1 = linelist[6].split('.')
                atomnafl2 = linelist[7].split('.')
                atomna1 = atomnafl1[1]
                atomna2 = atomnafl2[1]
                distance = linelist[4]

                # pre_ele1 : G1  pre_ele2 : THR
                # remove numbers from 'G13'
                for i in range(0,10):
                   atomna1=atomna1.replace(str(i),'')
                   atomna2=atomna2.replace(str(i),'')

                if len(atomna1)+len(atomna2) == 4:
                   if len(atomna1) == 1:
                      atomna1,atomna2 = atomna2,atomna1
                      atomID = atomID1
                   else:
                      atomID = atomID2
                   pairtype = atomna1+'_'+atomna2
                   # get category# using atomID and baseID
                   # ATOM    198  P       U A   7 0
                   category = 0
                   with open(path+'master.txt') as fm:
                      for mlines in fm.readlines():
                         matomIDlist = mlines.split()
                         matomID = matomIDlist[1]
                         if matomID == atomID:
                            category = mlines[27]
                            if category != 0:
                               cate[int(category)-1] += 1
                               cate_native[int(category)-1] += 1

                   if potmode == 'ba':
                      if int(category) in [3,7,9]:
                         continue

                   elif potmode == 'bb':
```

```
                    if int(category) in [1,2,4,5,6,8]:
                        continue

                    # for filename2 in os.listdir(path2+str(potcategory)):
                    for filename2 in os.listdir(path2+'0'):
                        if potcategory == 0 or potcategory == 98:  # full set
                            if (str(category)+'_') in filename2:
                                with open(path2+'0/'+filename2) as fp:
                                    for plines in fp.readlines():
                                        # ASP_A    0.3593608045735999
                                        plist = plines.split('\t')
                                        if plist[0] == pairtype:
                                            scoresum += float(plist[1])
                                            # scoresum += (8103*math.exp(-
2.46*float(distance))/40)*float(plist[1])
                        elif potcategory == 11:  # propensity
                            if (str(category)+'_') in filename2:
                                with open(path2+'11/'+filename2) as fp:
                                    for plines in fp.readlines():
                                        # ASP_A    0.3593608045735999
                                        plist = plines.split('\t')
                                        if plist[0] == pairtype:
                                            scoresum += float(plist[1])
                        elif potcategory == 10:  # overall
                            with open(path2+'10/overall.txt') as fp:
                                for plines in fp.readlines():
                                    # ASP_A    0.3593608045735999
                                    plist = plines.split('\t')
                                    if plist[0] == pairtype:
                                        scoresum += float(plist[1])
                        elif potcategory == 99:  # overall
                            with open(path2+'99/overall.txt') as fp:
                                for plines in fp.readlines():
                                    # ASP_A    0.3593608045735999
                                    plist = plines.split('\t')
                                    if plist[0] == pairtype:
                                        scoresum += float(plist[1])
                        elif potcategory == 13:  # propensity
                            with open(path2+'13/amos.txt') as fp:
                                for plines in fp.readlines():
                                    # ASP_A    0.3593608045735999
                                    plist = plines.split()
                                    if plist[0] == pairtype:
                                        scoresum += float(plist[1])
                        elif potcategory != 0 and potcategory != 11:  # one set only
                            with open(path2+str(potcategory)+'/pot.txt') as fp:
```

```
                        for plines in fp.readlines():
                            # ASP_A    0.3593608045735999
                            plist = plines.split('\t')
                            if plist[0] == pairtype:
                                scoresum += float(plist[1])

                # calculate rank in ftdock scoring
                with open(path+'structures/ftdock_rpscored.dat') as rpf:
                    for lines in rpf.readlines():
                        if 'G_DATA' in lines:
                            list1 = lines.split()
                            if filename.replace('g.pdb.hb','').replace('Complex_','') == list1[1]:
                                combscore = float(list1[4])*(-1) + scoresum

                # get RMSD
                # Complex_1006g.pdb.rstd   13.7907018661  1460
                with open(xpath + 'rmsd/'+Cname+'/RMSD2.out') as fr:
                    for rlines in fr.readlines():
                        if filename.strip('.hb') in rlines:
                            rlist = rlines.split('\t')

                            rmsd = rlist[1]
                            if float(rmsd) < 10:
                                group = 'Native'
                                print(rlines)
                            else:
                                group = 'Non_native'


fo.writelines(filename+'\t'+str(scoresum)+'\t'+str(rmsd).replace('\n','')+'\t'+group+'\t'+str(c
ombscore)+'\t'+\

str(cate_native[0])+'\t'+str(cate_native[1])+'\t'+str(cate_native[2])+'\t'+str(cate_native[3])
+\

'\t'+str(cate_native[4])+'\t'+str(cate_native[5])+'\t'+str(cate_native[6])+'\t'+str(cate_native[
7])+\
                        '\t'+str(cate_native[8])+'\n')

    # make a list of native like poses and scores
    with open(path+str(rangenum[0])+'arrows.txt','w+') as fo2:
        with open(path+str(rangenum[0])+'scores.txt') as scof:
            for lines in scof.readlines():
                scolist = lines.split('\t')
                if scolist[3] == 'Native':
                    fo2.writelines(scolist[0]+'\t'+scolist[1]+'\t'+scolist[2]+'\t'+scolist[4]+'\n')
```

```
    result =  str(cate[0])+' '+str(cate[1])+' '+str(cate[2])+' '+str(cate[3])+' '+str(cate[4])+'
'+str(cate[5])+' '+\
    str(cate[6])+' '+str(cate[7])+' '+str(cate[8])
    out_q.put(result)
```

Program: nativerank.py

```
# this code makes nativerank.txt that has the best rank and
# the number of native like structures
# and then make result***.txt which is like a log

def nativerank(gridsize,compname,posemax,potentials,resolim,potmode):
    print('****  NATIVERANK STARTED  ****')
    import shutil
    path = '/Users/kimuratakayuki/Desktop/Thesis/FTDock/'
    comfile = open('/Users/kimuratakayuki/Desktop/Thesis/FTDock/bbrestore.txt','r')
    comlist = comfile.readline().split('\t')
    bbrestore = comlist[1]
    combrank = 0
    rank = 0
    state=''

    if bbrestore == '1':
        state = 'Rm'
    elif bbrestore == '0':
        state = 'Rf'

    with open(path+'nativeranks.txt','w+') as fo:
        with open(path+'arrows.txt') as fa:
            for lines in fa.readlines():
                if lines != '\n':
                    nlist = lines.split('\t')
                    score = nlist[1].replace(' ','')
                    combscore = nlist[3]
                    combcount = 0

                    # get the rank for each line in arrows.txt
                    with
open(path+'results/scores/'+compname.strip('.pdb')+'_'+potmode+'_'+str(gridsize)+'_'+str(
posemax)+'_3_'+str(potentials)+'_asa_0_'+str(resolim)+'.txt') as fr:
                        count = 0
                        for rlines in fr.readlines():
                            if rlines != '\n':
                                rlist = rlines.split('\t')
                                try:
```

```
                        if float(score) < float(rlist[1]):
                            count += 1
                        if float(combscore) < float(rlist[4]):
                            combcount += 1

                    except ValueError:
                        print('VALU ERROR  '+rlines)

                rank = count + 1
                combrank = combcount + 1
            if combrank == 0:
                combrank = posemax
            fo.writelines(lines.replace('\n','')+'\t'+str(rank)+'\t'+str(combrank)+'\n')

        # calculate the best rank in FTDock scoring from RMSD2.out
        with open(path+'RMSD2.out') as rmf:
            bestposeid = 50373
            for lines in rmf.readlines():
                linlist = lines.split('\t')
                if float(linlist[1]) < 10.0:
                    list2 = linlist[0].split('_')
                    list3 = list2[1].split('g.')
                    poseid = float(list3[0])
                    if poseid < bestposeid:
                        bestposeid = poseid
            fo.writelines('Best FTrank is : Complex_'+str(bestposeid)+'g.pdb and rank is
'+str(bestposeid))

        bestrank = 50373
        combbestrank = 50373
        # read the output and get the best ranking
        with open(path+'nativeranks.txt') as rankf:
            i = 0
            for ranline in rankf.readlines():
                if 'Best ' in ranline:
                    continue
                i += 1
                ranlist = ranline.split('\t')
                if int(ranlist[4]) < int(bestrank):
                    bestrank = ranlist[4]
                if int(ranlist[5]) < int(combbestrank):
                    combbestrank = ranlist[5]

            a1 = 'Best rank '+str(bestrank) + ' , number of native like '+str(i)
            a2 = 'FTDock best rank '+str(bestposeid)+', combined scoring rank best '+
str(combrank)
```

```
        nativemess = [a1, a2]

        # save native ranks with complex name in its filename

shutil.copy2(path+'nativeranks.txt',path+'results/natives/result_'+compname+'_'+state+'_'
+str(gridsize)\
                +'_'+str(posemax)+'_'+str(potentials)+'.txt')

        print('****  NATIVERANK COMPLETED  ****')
        return nativemess
```

Appendix F Main Python Scripts for Unpublished Results

Program: addPR_cate.py

```
def addPR_cate(patom, ratom):

    catelist = [[(1+i+17*j) for i in range(17)] for j in range(15)]  # column:protein,
row:RNA
    patomlist = ["OH", "OG1", "OG", "OE2", "OE1", "OD2", "OD1", "O", "NZ", "NH2",
"NH1", "NE2", "NE1", "NE", "ND2", "ND1","N"]
    ratomlist = ["OP2", "OP1", "O6", "O5'", "O4'", "O4", "O3'", "O2'", "O2", "N7", "N6",
"N4", "N3", "N2","N1"]
    #                              5                        10                       15
    try:
        pindex = patomlist.index(patom)
    except ValueError:
        return ''
    try:
        rindex = ratomlist.index(ratom)
    except ValueError:
        return ''
    return catelist[rindex][pindex]
```

Program: pfasta.py

```
def pfasta(path): # path = /--/PRat_asa/FTDock/
    import os
    aminos = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']
    aminot =
['ALA','CYS','ASP','GLU','PHE','GLY','HIS','ILE','LYS','LEU','MET','ASN','PRO',\
              'GLN','ARG','SER','THR','VAL','TRP','TYR']

    with open(path+'pfasta.txt','w') as fo2:
        print('')
```

```
    with open(path+'pfasta.txt','a') as fo:
        # pdb1j5e.ent.strideout
        # 5aor.pdb.strideout
        # 4v42-pdb-bundle.tar.strideout
        # ASG  THR A  134  122    E       Strand   -130.85    151.94     26.5     1A34
        # ASG  VAL A  135  123    E       Strand   -118.42    173.67     58.5     1A34
        for files in os.listdir(path+'structures/stride_out/'):
            if '.pdb.strideout' in files or '.ent.strideout' in files:
                with open(path + 'structures/stride_out/'+files) as fi:
                    pdb = files[0:4]
                    if files[0:3] == 'pdb':
                        pdb = files[3:7]
                    chainlist = []
                    for lines in fi.readlines():
                        # 0. AlphaHelix
                        # 1. 310Helix
                        # 2. Pi-helix
                        # 3. Strand
                        # 4. Bridge
                        # 5. Turn
                        # 6. Coil or else

                        # ASG  VAL P    2    2    C         Coil    -75.53    134.66     13.9     ~~~~
                        # ASG  VAL P    3    3    E       Strand   -133.52    116.40     11.0     ~~~~
                        try:
                            if lines[0:3] == 'ASG':
                                chain = lines[9]
                                if chain not in chainlist:
                                    chainlist.append(chain)
                                    fo.writelines(pdb.upper()+':'+chain+'\n')
                                if 'AlphaHelix' in lines.split()[6]:

fo.writelines(lines.split()[1]+'\t'+lines.split()[2]+'\t'+'0'+'\t'+lines.split()[3]+'\t'+'\n')
                                elif '310Helix' in lines.split()[6]:

fo.writelines(lines.split()[1]+'\t'+lines.split()[2]+'\t'+'1'+'\t'+lines.split()[3]+'\t'+'\n')
                                elif 'PiHelix' in lines.split()[6]:

fo.writelines(lines.split()[1]+'\t'+lines.split()[2]+'\t'+'2'+'\t'+lines.split()[3]+'\t'+'\n')
                                elif 'Strand' in lines.split()[6]:

fo.writelines(lines.split()[1]+'\t'+lines.split()[2]+'\t'+'3'+'\t'+lines.split()[3]+'\t'+'\n')
                                elif 'Turn' in lines.split()[6]:

fo.writelines(lines.split()[1]+'\t'+lines.split()[2]+'\t'+'4'+'\t'+lines.split()[3]+'\t'+'\n')
                                elif 'Bridge' in lines.split()[6]:
```

```
fo.writelines(lines.split()[1]+'\t'+lines.split()[2]+'\t'+'5'+'\t'+lines.split()[3]+'\t'+'\n')
                        else: # including Coil

fo.writelines(lines.split()[1]+'\t'+lines.split()[2]+'\t'+'6'+'\t'+lines.split()[3]+'\t'+'\n')
                    except ValueError:
                        continue

    for files in os.listdir(path.replace('/FTDock/','')+'/Clustering/mmCIF/'):
        if files[4:8] == '.cif' and len(files) == 8:   # files = '1a1t.cif'
            with open(path.replace('/FTDock/','')+'/pfasts/'+files[0:4].upper()+'.pfasta','w') as
fos:
                with open(path+'pfasta.txt') as fis:
                    start = 0
                    for lines in fis.readlines():
                        # 1A34:A
                        # T    A  3  13
                        # G    A  3  14
                        if start == 0:
                            if files[0:4].upper() in lines:
                                start = 1
                                fos.writelines(lines)
                        elif start == 1:
                            if ':' in lines:
                                if files[0:4].upper() not in lines:
                                    break
                                else:
                                    fos.writelines(lines)
                            else:
                                fos.writelines(lines)
pfasta('/Users/kimuratakayuki/Desktop/PRat77/FTDock/')

Program:  parseDSSR.py

def parseDSSR(): # this code generates rfasta.txt ******  MANUALLY RUN !!!

    def parseDS(dssrfile,ciffile,path):
        def sec_check(file,resi,type):  # resi -> Q.G344
            try:
                with open(file) as f:
                    start,found = 0,0
                    for lines in f.readlines():
                        if start == 0:
                            if 'List' in lines and type in lines:
                                start = 1
                        elif start == 1:
```

```
                    if '******' in lines:
                        break
                    elif resi in lines:
                        if resi+',' in lines:
                            found = 1
                            break
                        elif lines.split()[2].split(',')[-1] == resi:
                            found = 1
                            break
            return found
        except FileNotFoundError:
            pass

    with open(path+'rfasta.txt','a') as fo:
        with open(ciffile) as fr:  # make resilist in the cif file
            resilist,chlist = [],[]
            for rlines in fr.readlines():
                # ATOM 486   H H6     . C   A 1 15 ? 15.751  0.322   -6.067  1.00 0.00 ? ? ?
? ? ? 18 C   A H6     1
                # ATOM 487   N N      . ASN B 2 1  ? 10.545  2.262   5.673   1.00 0.00 ? ? ?
? ? ? 1  ASN B N      1
                # ATOM 488   C CA     . ASN B 2 1
                if rlines[0:4] == 'ATOM' and len(rlines.split()[5]) == 1:
                    resi = rlines.split()[23]+'.'+rlines.split()[5]+rlines.split()[21] # S.U608
                    if resi not in resilist and len(rlines.split()[23]) == 1:
                        resilist.append(resi)

        for resi in resilist: # classify the residues of the resilist
            category = ''
            if resi[0] not in chlist:
                chlist.append(resi[0])
                fo.writelines(dssrfile[-12:][0:4].upper()+':'+resi[0].upper()+'\n')

            # classify except helices : type -> 'hairpin loops','bulges','internal
loops','junctions',
            #
            # 0 Single Stranded       'single-stranded'
            # 1 Hairpin Loop          'hairpin loops'
            # 2 Bulge or Internal Loop 'bulges','internal loops'
            # 3 Junction              'junctions'
            # 4 Helix A-form
            # 5 Other Helix
            # 6 Other

            # assign category 0-3   *** priority ***
            if sec_check(dssrfile,resi,'single-stranded') == 1:   # S.U608
```

```
                category = 3

fo.writelines(resi.split('.')[1][0]+'\t'+resi[0].upper()+'\t'+str(category)+'\t'+resi.split('.')[1][
1:]+'\n')
                continue
            elif sec_check(dssrfile,resi,'hairpin loop') == 1:
                category = 4

fo.writelines(resi.split('.')[1][0]+'\t'+resi[0].upper()+'\t'+str(category)+'\t'+resi.split('.')[1][
1:]+'\n')
                continue
            elif sec_check(dssrfile,resi,'junction') == 1:
                category = 5

fo.writelines(resi.split('.')[1][0]+'\t'+resi[0].upper()+'\t'+str(category)+'\t'+resi.split('.')[1][
1:]+'\n')
                continue

            try:
                with open(dssrfile) as fi:
                    found,start,any = 0,0,0
                    for lines in fi.readlines():
                        if start == 0:
                            if lines[0:8] == '  helix#':
                                start,any = 1,1
                                signlist = []
                        elif start == 1:
                            if 'helix-form' in lines:
                                signlist = lines[18:]
                            elif lines[2:4] == '--':
                                start = 0
                            elif '********' in lines:
                                break
                            elif resi in lines:
                                found = 1
                                if signlist[int(lines.split()[0])-1] == 'A':
                                    category = 0

fo.writelines(resi.split('.')[1][0]+'\t'+resi[0].upper()+'\t'+str(category)+'\t'+resi.split('.')[1][
1:]+'\n')
                                    break
                                elif signlist[int(lines.split()[0])-1] == 'x':
                                    category = 1

fo.writelines(resi.split('.')[1][0]+'\t'+resi[0].upper()+'\t'+str(category)+'\t'+resi.split('.')[1][
1:]+'\n')
```

```
                        break
                    elif signlist[int(lines.split()[0])-1] == '.':
                        category = 2

fo.writelines(resi.split('.')[1][0]+'\t'+resi[0].upper()+'\t'+str(category)+'\t'+resi.split('.')[1][
1:]+'\n')
                        break
            except FileNotFoundError:
                pass

            if category == 0 or category == 1 or category == 2:
                continue
            else:
                category = 6

fo.writelines(resi.split('.')[1][0]+'\t'+resi[0].upper()+'\t'+str(category)+'\t'+resi.split('.')[1][
1:]+'\n')
                continue

    import os
    path0 = os.getcwd()
    path0 = path0.replace('/python/8.apply_potentials','')+'/' # path0 = /Users/---/PRat_asa/
    path = path0+'FTDock/'    # path = /--/PRat_asa/FTDock/
    with open(path0+'rfasta.txt','w'):  # renew rfasta.txt
        pass
    for files in os.listdir(path0+'Clustering/mmCIF/'):
        if files[4:8] == '.cif' and len(files) == 8:   # only cif files not '1a1t.cif.out'

parseDS(path0+'Analyzing/DSSRout/'+files[0:4]+'.cif.out',path0+'Clustering/mmCIF/'+fi
les[0:4]+'.cif',path0)

    for files in os.listdir(path0+'Clustering/mmCIF/'):
        if files[4:8] == '.cif' and len(files) == 8:   # files = '1a1t.cif'
            with
open(path.replace('/FTDock/','')+'/DSSRparsed/'+files[0:4].upper()+'.rfasta','w') as fos:
                with open(path0+'rfasta.txt') as fis:
                    start = 0
                    for lines in fis.readlines():
                        if start == 0:
                            if files[0:4].upper() in lines:
                                start = 1
                                fos.writelines(lines)
                        elif start == 1:
                            if ':' in lines:
                                if files[0:4].upper() not in lines:
                                    break
```

```
                        else:
                            fos.writelines(lines)
                    else:
                        fos.writelines(lines)

    print('**** DSSRparse complete ****')
parseDSSR()
```

Appendix G Shell Script

Program: DSSR.sh

```
#!/bin/bash
echo "****  DSSR STARTED  ****"
ifiles=/Users/kimuratakayuki/Desktop/Thesis/FTDock/structures/$1
outputdir=/Users/kimuratakayuki/Desktop/Thesis/FTDock/

for file in $ifiles;do
/usr/local/bin/x3dna-dssr --input=$file --output=$outputdir/DSSR.out &>dssr.log
done

echo "****  DSSR COMPLETED  ****"
```

Program: FTDock.sh

```
#!/bin/bash
path="/Users/kimuratakayuki/Desktop/Thesis/FTDock/structures/"
path1="/Users/kimuratakayuki/Desktop/Thesis/FTDock/"
path2="/Users/kimuratakayuki/Desktop/Beforeprelim/ftdock-2-dev/scripts-2.0.3/"

mfile=$path$1 # $1:protein like 1QUX_p.pdb
rfile=$path$2 # $2:rna

/opt/local/bin/perl $path1'preprocess-pdb.perl' -pdb $mfile shell=TRUE > prepro1.log
/opt/local/bin/perl $path1'preprocess-pdb.perl' -pdb $rfile shell=TURE > prepro2.log

alt=".parsed"
mf=${mfile%.pdb}$alt
rf=${rfile%.pdb}$alt

echo " ****  FTDOCK STARTED  **** "
# according to bbrestore($3), switch mobile and fixed
if [ "$3" -eq "1" ]; then
   mpirun -v -np 8 ftdock_mpi -mobile $rf -static $mf -calculate_grid $4 -dlim_static $9 -
dlim_mobile $9 -out $path'ftdock_global.dat' -surface $7 $8 -noelec > ftdock.log
elif [ "$3" -eq "0" ]; then
```

```
   mpirun -v -np 8 ftdock_mpi -mobile $mf -static $rf -calculate_grid $4 -dlim_static $9 -
dlim_mobile $9 -out $path'ftdock_global.dat' -surface $7 $8 -noelec > ftdock.log
fi
echo " ****  FTDOCK COMPLETED  **** "

echo " ****  RPSCORE STARTED  **** "
if [ "$6" -eq "1" ]; then  # default matrix
   /usr/local/bin/rpscore -in $path'ftdock_global.dat' -out $path'ftdock_rpscored.dat' -
matrix $path'best.matrix' > rpscore.log
elif [ "$3" -eq "0" ]; then   #  all 1 matrix
   /usr/local/bin/rpscore -in $path'ftdock_global.dat' -out $path'ftdock_rpscored.dat' -
matrix $path'allone.matrix' > rpscore.log
fi

echo " ****  RPSCORE COMPLETED  **** "
echo " ****  BUILD STARTED  **** "
cd /Users/kimuratakayuki/Desktop/Thesis/python/8.apply_potentials/restored
/usr/local/bin/build -b2 $5 -in $path'ftdock_rpscored.dat' > build.log

echo " ****  BUILD COMPLETED  **** "
```