A NEW APPROACH TO OPTIMIZE A PROTEIN ENERGY FUNCTION ON A FOLDING PATHWAY USING GŌ-LIKE POTENTIAL AND ALL-ATOM, *AB INITIO* MONTE CARLO SIMULATIONS

APPROVED BY SUPERVISORY COMMITTEE

A NEW APPROACH TO OPTIMIZE A PROTEIN ENERGY FUNCTION ON A FOLDING PATHWAY USING GŌ-LIKE POTENTIAL AND ALL-ATOM, *AB INITIO* MONTE CARLO SIMULATIONS

by

Alexandra Safronova

DISSERTATION

Presented to the Faculty of the Graduate School of Biomedical Sciences

The University of Texas Southwestern Medical Center at Dallas

in Partial Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

The University of Texas Southwestern Medical Center

Dallas, Texas

May, 2016

A NEW APPROACH TO OPTIMIZE A PROTEIN ENERGY FUNCTION ON A FOLDING PATHWAY USING GŌ-LIKE POTENTIAL AND ALL-ATOM, *AB INITIO* MONTE CARLO SIMULATIONS

Alexandra Safronova, M.S.

Nick Grishin, Ph.D.

Prediction of a protein structure is important for understanding the function of a protein. The process of protein structure prediction employs the approximation of a protein free energy that guides protein folding to the protein's native state. A function with a good approximation of the protein free energy should allow estimation of the structural distance of the evaluated candidate structure to the protein native state. Currently the energy optimization process relies on the correlation between the energy and the similarity to the native structure. The energy function is presented as a weighted sum of components which are designed by human experts with the use of statistical analysis of solved protein strictures. Values of the weights are derived through the procedure that maximizes the correlation between the energy and the similarity to the native structure measured by a root mean square deviation between coordinates of the protein backbone.

Two major components are required for a successful *ab initio* modelling: (1) an effective energy function that discriminates the native protein structure out of all possible decoy structures; (2) an efficient sampling algorithm that quickly searches for the low-energy states. In this dissertation a new method for energy optimization is proposed. The method relies on a fast sampling algorithm and targets successful protein folding. The weights for energy components are optimized on a found with the Gō potential energy fast folding pathway. The Lennard-Jones potential, the Lazaridis-Karplus solvation potential, hydrogen bonding potential are used in the optimization algorithm. The optimized weights successfully predict all α and α/β proteins.

The proposed strategy is conceptually different from the existing methods that optimize the energy on solved protein structures. The developed algorithm is a novel concept that allows the optimization of a more complex functional combination of the energy components that would improve the prediction quality.

Contents

List of Figures

List of Tables

List of Abbreviations

HB	Hydrogen bonding potential
LJ	the Lennard-Jones potential
LK	the Lazaridis-Karplus Potential
MC	Monte Carlo simulations
MD	Molecular Dynamics simulations
RMSD	Root Mean Square Deviation
SA	Simulated Annealing

Chapter 1

Introduction

1.1 Overview of the problem

Protein structure determination is central for understanding protein function (40). X-ray crystallography, nuclear magnetic resonance (NMR) and electron microscopy (EM) are common experimental techniques for protein structure determination. However, these methods are expensive and time consuming, leading to a large discrepancy between the number of known protein sequences and the number of solved protein structures (3, 4). Efficient computer-based algorithms capable of predicting 3D structures from sequences could reduce this gap (26).

Computational methods for protein structure prediction can be grouped into three categories: homology modeling (1, 82), threading (10, 34, 56, 94, 103), and *ab initio* methods (9, 12, 13, 25, 45, 55, 86, 98). Homology modeling methods assume structure resemblance from proteins that share sequence similarity. Threading approaches compare a target sequence against a set of known protein structures by using statistical knowledge of the relationship. If similar protein structures are not available, *ab initio* methods build the 3D models “from scratch”. Unlike homology modeling and threading approaches, *ab initio* algorithms are essential for understanding how and why a protein folds to its specific structure out of the large number of possibilities.

Ab initio modeling searches protein conformational space under the guidance of a designed energy function and generates a set of possible conformations from which final models are chosen. Three key components are required for successful *ab initio* modeling: (1) an effective energy function that discriminates the native protein structure out of all possible decoy structures (16, 53, 57, 86, 90); (2) an efficient sampling algorithm that quickly searches for the low-energy states; (3) a selection method for native-like models from a pool of decoys.

Typically, an energy function consists of a variety of energy terms that represent different structural features and the interplay between local and global interactions among amino acid residues. It includes Van Der Waals interactions, electrostatic interactions, hydrogen bonding, and solvation potential. If a sampling algorithm uses fixed backbone atoms and rotamer libraries, then the bonding energy terms often are ignored. The weights are used to balance the contribution of terms to the overall energy. The ability of an energy function to predict the structure of novel proteins is evaluated by its prediction accuracy on an independent set of test proteins. This approach aims to stabilize the correct structures and to destabilize incorrect ones by harnessing the theoretical argument that the native protein state is characterized by a large energy gap (92). Therefore, the best energy function would maximize the Z-score, defined as the difference between the mean energy of the native-like structures and the mean energy of the non-native structures divided by the standard deviation of the energy of the non-native structures (33, 54, 63).

1.2 Aims and Methodological Aspects

In this report, I describe a new method for energy weight optimization, inspired by the design of a funnel –shaped energy surface for folding of α, β and αβ proteins (51, 81, 91) This funnel sculpting method generates an energy function in an iterative manner until a random starting conformation folds into a native-like fold. In our algorithm, I outline the energy funnel shape by tuning a Monte Carlo based sampling algorithm (5) with a Gō potential (32) as a scoring function to simultaneously fold proteins from different SCOP classes (65). I used off-lattice all-atom protein models, including hydrogens atoms, and started folding simulations from a fully extended backbone chain. In general, a Gō potential is often viewed as an idealized energy landscape with a strong correlation between energy and structural distance from the native state, and, despite the criticism, is used to study folding kinetics (20, 70, 85, 104). Thus, I assumed that this strategy would be able to find fast-folding pathways (85) in which I optimized the weights for physics- and/or knowledge-based energy terms. The weight optimization routine employed a scoring function designed as a linear combination of the Gō potential and the three most frequently used potentials. During the algorithm's iterations, I minimize the Gō potential weight.

The method presented in this study is a novel concept. Although the potential I employed did not contain a sufficient number of energy terms to independently fold an arbitrary protein sequences, I demonstrated that the full conformational search of the ground states can be solved by the available sampling methods and standard computational resources.

Chapter 2

Background and Related Work

2.1 Amino acids and Proteins

Proteins are polymers that composed of 20 different amino acids (residues) encoded in DNA or RNA sequence. Each amino acid includes α-carbon (C_{α}) bonded to amino (NH) and carboxyl (COOH) groups and a side chain, which is different for each amino acid. The amino acids in the protein chain are linked with peptide bonds (CO-NH) formed between the amino and carboxyl groups (**Figure 2.1**). These bonds are formed during the polymerization process when a molecule of water is lost. The connected carbon, oxygen and nitrogen atoms form a protein backbone. This backbone can adapt repeating local structures called secondary structure elements: α helices, β sheets or loops (**Figure 2.1**). The secondary structure elements and their spatial interrelations form the tertiary structure of a protein. A part of a protein folded into a distinct structural region comprises a protein domain. Under physiological conditions proteins spontaneously fold into a particular shape named as its native state. The native state geometry of a protein defines its behavior and function.

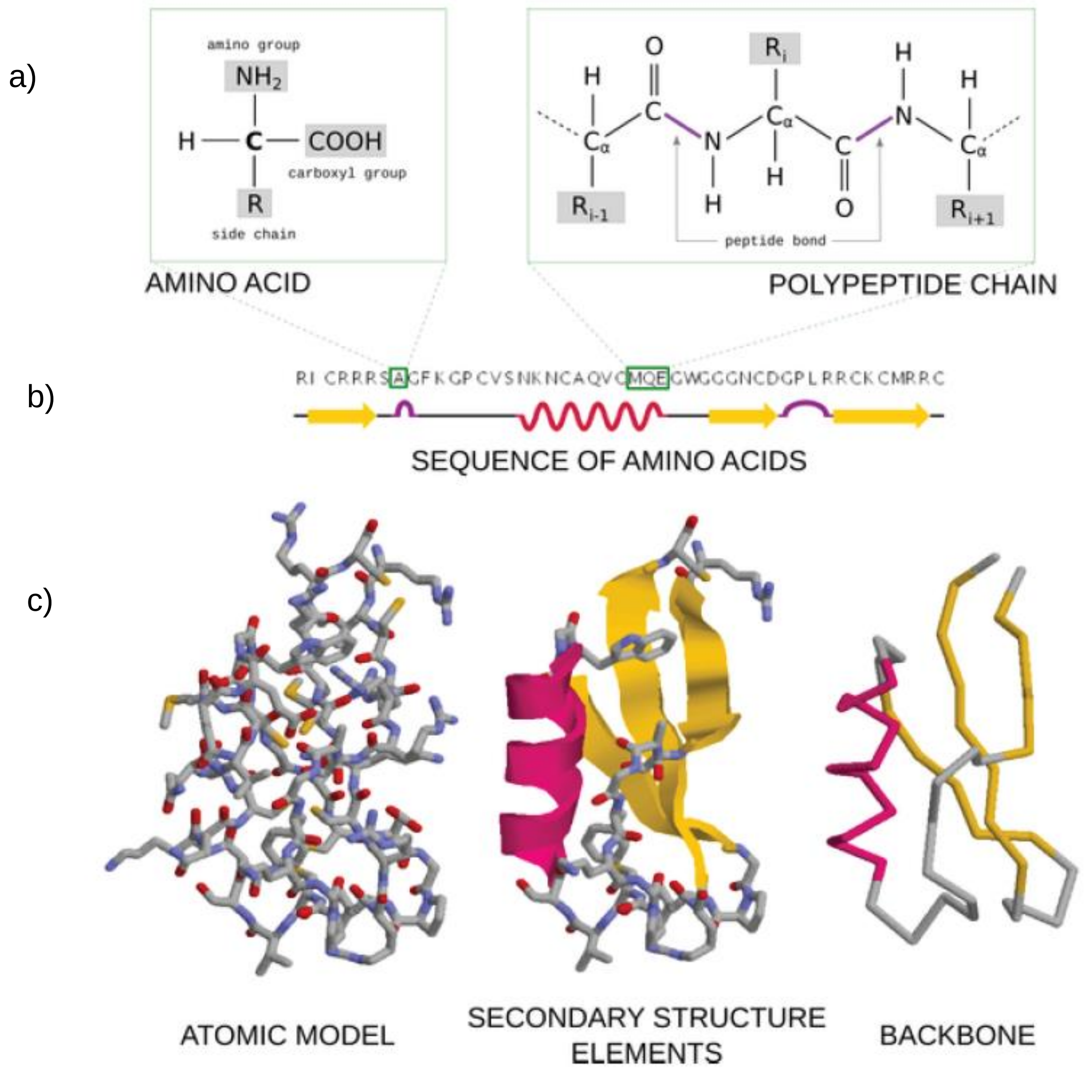


Figure 2.1: **Protein Structure**. a) The atomic structure of a single amino acid and the chain of amino acids are; b) a protein sequence.; c) an all-atom representation of the protein colored by atom type, secondary structure elements, and a backbone representation with colored secondary structure elements.

2.2 Protein Folding

Anfinsen showed that protein ribonuclease can be reversibly denatured / renatured in a test tube (2). It indicates that if a protein always folds into the same native structure, it is possible to develop a protein folding algorithm that uses only the information contained in a sequence to fold a protein towards its native or near native state.

2.2.1 Folding Funnel

The energy landscape of protein folding, known as a folding funnel, can be viewed as a rugged landscape with kinetic traps, energy barriers and some narrow throughway path to native (28). It is considered that the native state of a protein corresponds to the global energy minimum, the lowest point of the landscape, and the folding process is a roll down a free energy hill to the bottom (28). **Figure 2.2** demonstrates a folding funnel, where the native state (N) corresponds to the energy minimum surrounded by steep slopes (29).

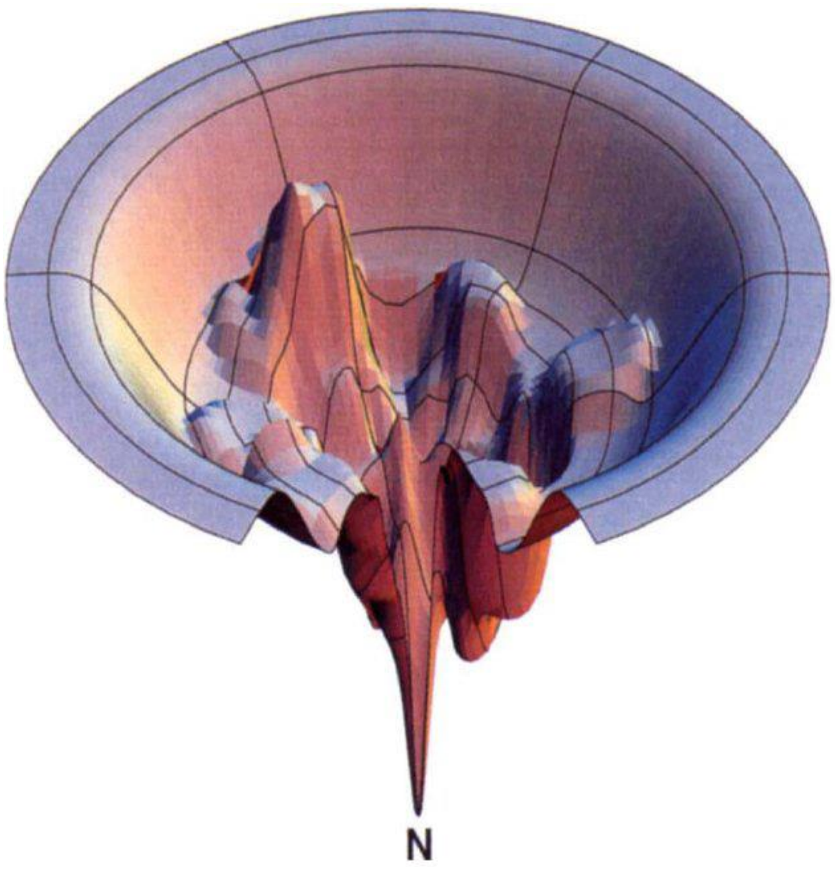


Figure 2.2. Folding funnel (28)

2.2.3 Levinthal's Paradox

If I take into account the enormous number of protein’s possible conformations, the process of a protein self-assembly into the native state is remarkably efficient. The gap between expected and current folding speed was demonstrated by Levinthal (52) and known as Levinthal's paradox. For example, if we have a polypeptide of 101 amino acids and each bond connecting two neighboring amino acids can have 3 possible states then the number of possible configurations is equal to $3^{100} = 5{\cdot}10^{47}$. Assuming a protein sampling rate of 10^{13} configurations per second (0.1 ps per configuration) it will take 10^{27} years to try all of them. Nevertheless, proteins fold in seconds, not performing an exhaustive search of entire conformational space. There are several possible explanations of the paradox: 1) proteins through the evolution process specialized to fold rapidly, meaning that other amino acid sequences simply did not survive; 2) the lack of stability of some proteins (37). It was also noticed that the native configuration may not have the lowest free energy. Thus, absolute stability is not required since a protein has to survive as long as it performs perform its function.

2.2.4 Molecular Dynamics

A potential energy of a whole system represented by a sum of forces for all atoms is called a force field. Two groups of forces affect folding of protein molecules: bonded and non-bonded forces. Bonded forces result from a covalent bond between two atoms or ionic bond between oppositely charged ions, the atoms that lost or gained one or more electrons. Non-bonded forces result from distance interactions between atoms. Bonded forces attribute to bending, stretching and rotating. Bending (**Equation**

2.1) and stretching potentials (**Equation 2.2**) adapt an elastic spring model. They are parametrized with the equilibrium values (l_0 and Θ_0) and bond/angle stiffness (k_l and k_Θ where $k_\Theta \ll k_l$).

$$V(l) = \frac{1}{2}k_l(l - l_0)^2 \tag{2.1}$$

$$V(\theta) = \frac{1}{2}k_\theta(\theta - \theta_0)^2 \tag{2.2}$$

Bond rotation potential (**Equation 2.3**) is represented by a cosinusoid function of parametrized amplitude (V_ω), periodicity (n) and phase shift (ϕ):

$$V(\omega) = \frac{1}{2}V_\omega[1 + cos(n\omega - \phi)] \tag{2.3}$$

Non-bonded forces are represent by steric effects, van der Waals' interactions and electrostatics. Steric interactions are caused by Pauli short range repulsion due to an energetic cost of electron clouds overlap). The polarization of molecules causes long range attractive forces, called Van der Waals' forces They are modelled together with Lennard-Jones potential:

$$V(r) = \epsilon\left[\left(\frac{r_{min}}{r}\right)^{12} - 2\left(\frac{r_{min}}{r}\right)^{6}\right] \tag{2.4}$$

where r_{min} is a distance at the minimum of potential. Electrostatic charge is expressed by Coulomb's law:

$$V(r) = \frac{1}{4\pi\varepsilon_0}\frac{q_1q_2}{r} \tag{2.5}$$

The full force field that can reproduce the basic features of protein energy landscapes at an atomic level of detail is then represented by the following potential energy function [PW]:

$$\sum_{i}^{bonds} V(l_i) + \sum_{i}^{angles} V(\theta_i) + \sum_{i}^{rotations} V(\omega_i) + \\ + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left\{ \epsilon_{ij} \left[\left(\frac{r_{min_{ij}}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{min_{ij}}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \right\} \tag{2.6}$$

Over the last thirty years several families of force fields has been developed. The most used for protein folding are AMBER (Assisted Model Building and Energy Refinement), CHARMM (Chemistry at HARvard Macromolecular Mechanics), OPLS (Optimized Potential for Liquid Simulations) families (76). Their force fields include those terms in **Equation 2.6**. Parameters representing a single force field designed for certain type of molecules may vary for each family. The computational cost of these all-atom energy functions is very high and limits their application to the molecular dynamic simulations of short proteins.

2.3 Protein Structure Prediction

The complete search of all possible protein conformations is not possible and all-atom model simulations demand enormous computational resources. Thus, simplified models were proposed.

2.3.1 Minimalist models for protein folding

Native state topology plays important role in determining the speed and mechanism of folding for small proteins as was shown by simulations using minimalist protein models. Minimalist protein models are able to rapidly collect meaningful statistics about folding pathways and kinetics focusing on the fundamental physics of the problem and linking the results to experimental observations for a target protein. Over the last thirty years many minimalist models have been developed ranging from all-atom Gō potentials to coarse-grained bead models with Gō interactions substituted or enhanced by physics based potentials. The reduction of computational burden provided by coarse-grained models may support folding studies on a genomic scale and protein design.

2.3.1.1 Coarse-grained models

HP model (27) refers to hydrophobic collapse hypothesis. It was observed that ground states have hydrophobic core and polar exterior (14). HP model uses a 2 letter alphabet (H and P) instead of the 20 element amino acid alphabet. Only one property of an amino acid, hydrophobicity or polarity, is taking into consideration. The force of attraction of hydrophobic residues is used as a scoring function. The number of gained H-H contacts measures quality of a folded structure. However, HP model was shown to be NP-complete even for two dimensional lattice (73).

Several residue-level models have been proposed where an atomic representation was reduced to the center of mass of an atomic group. These models are represented by UNRES (54), CAS (101), SICHO (47). The loss of detail is compensated by knowledge-based potentials derived from a statistical analysis of solved protein structures. Such energy functions do not capture the physical free energy explicitly but represent the probability that a given structure is native-like.

2.3.1.2 Gō-like models

Gō potentials favor native state contacts of the target fold and have been used to illustrate the energy landscape. Simulations with Gō potentials usually employ the protein chain as a string of beads (19, 38). Minimalist Gō-models minimize energetic traps (roughness) on the free energy surface. The main limitation of Gō-like models is that they cannot be used to characterize protein landscape regions where energetic frustration is not negligible such as certain compact non-native states and misfolding processes. The studies of misfolding process require take into account non-native interactions, energetic heterogeneity, and frustration within definitions of an appropriate model. Nevertheless, Gō potentials provide a sufficient model to explain rapid and reliable folding of native sequences relative to poorly designed or arbitrary heteropolymer sequences (28, 46, 67, 79). Those conclusions support the hypothesis that evolution has evolved sequences that favor fast folding (62). The argument towards support that minimalist models can give reasonable approximation of the folding of the real proteins is the observation that the magnitude of the folding rate for two-state folders is correlated with average distance between contacting residues in the native state (75).

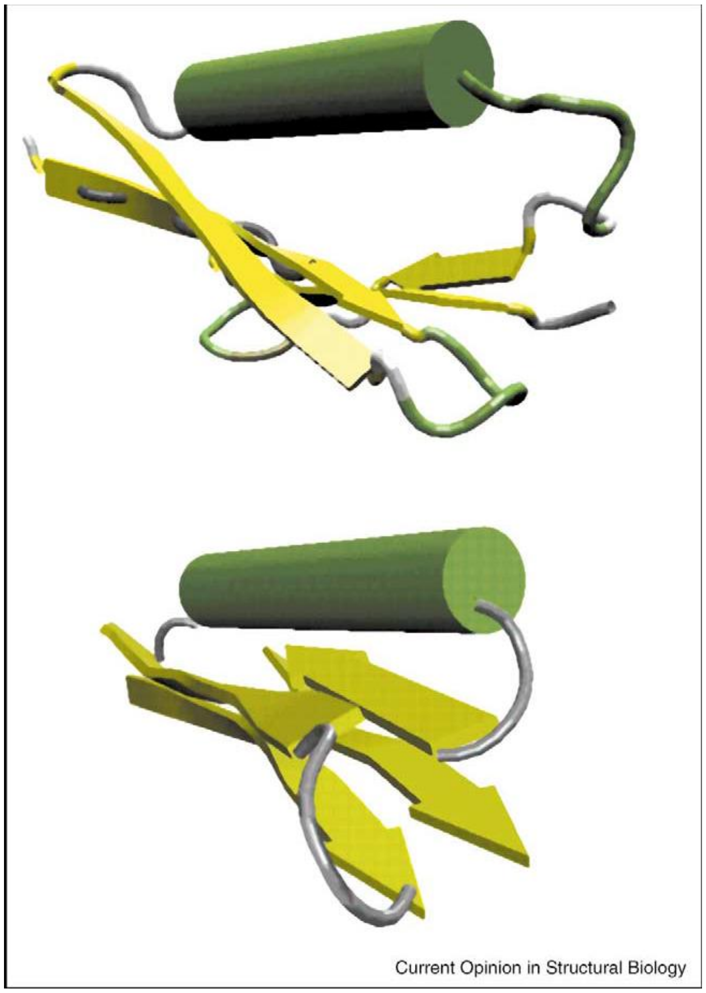


Figure 2.3: (38): **Minimalist model of L and G proteins**. Minimalist model of the native state topology of protein L (bottom) and the NMR solution of G structure (top), showing the similar arrangement of secondary and tertiary structure.

Sequence-independent Gō models are not suitable for exploring folding process of proteins with low sequence identity but high structural homology since they have minimal energetic frustration. An example of delicate balance between energetic and topological frustration was demonstrated for proteins G and L (**Figure 2.3**). It was demonstrated that G and L proteins fold by different pathways (35, 36, 66, 71, 74, 80). In protein L the first β hairpin forms with the second β hairpin unstructured. Protein G folds through a transition state with purported rate-limiting formation of the second b hairpin. It was also demonstrated that protein G folds through multiple pathways that involve intermidiates (**Figure 2.4**).

2.3.2 CASP

CASP (Critical Assessment of Structure Prediction) is an experiment, where structural predictions (the native structures are not known at the moment of submission) of set of the target protein sequences are submitted by the participants. The accuracy of a model is then assessed by the comparison to a real native structure. Targets are classified into three categories: comparative modeling, fold recognition/threading and new fold/ab initio prediction. Comparative modeling methods make use of structure resemblance between proteins that share sequence similarity. Threading approaches compare the target sequence against a set of known protein structures by using statistical knowledge of the relationship. If similar protein structures are not available, *ab initio* methods build the 3D models “from scratch”.

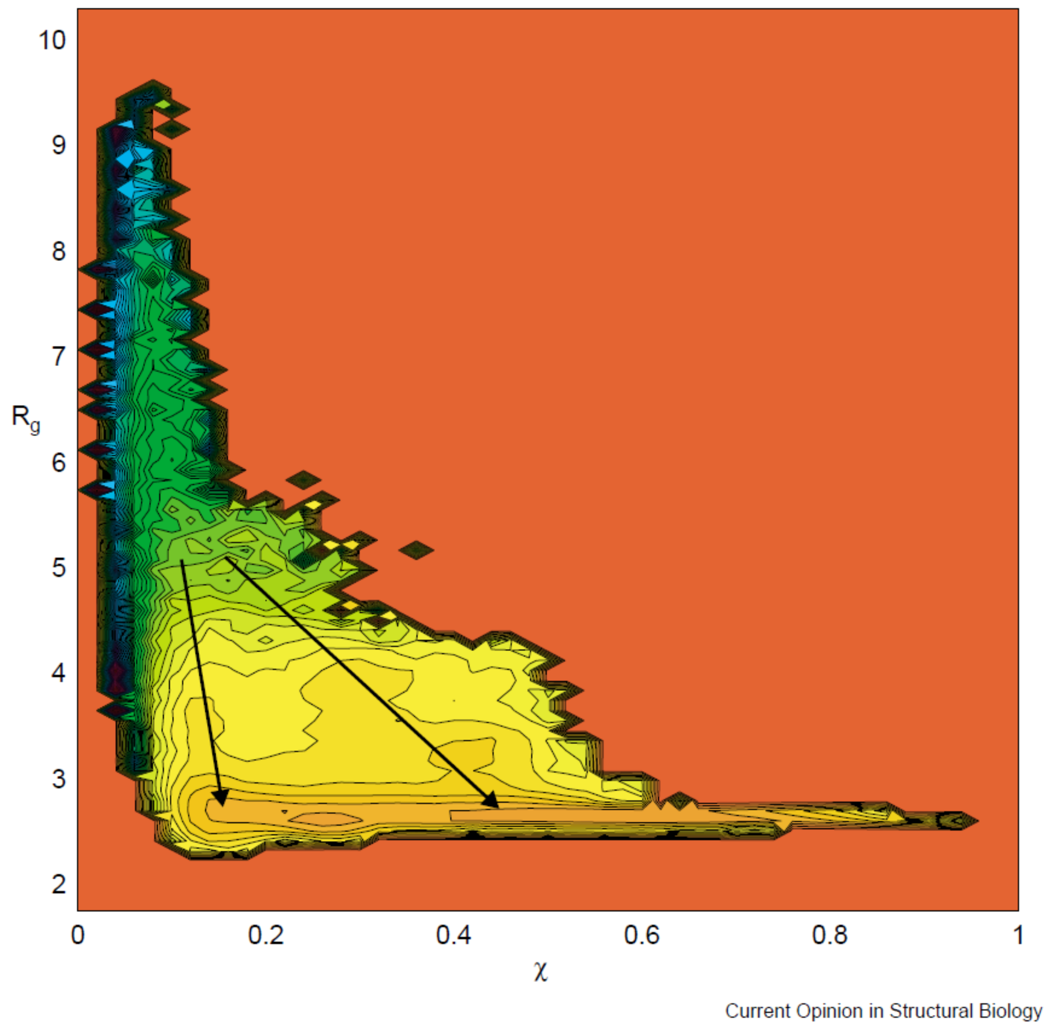


Figure 2.4: (38) Two folding pathways for protein G. Free energy at the folding temperature as a function of radius of gyration (Rg) and native state similarity (w) for protein G. Two folding pathways are present. The fast pathway corresponds to a collapse-concomitant folding pathway (arrow on right), whereas the slow pathway (arrow on left) corresponds to rapid non-native collapse with a structured second b hairpin and a longer process of finding the native structure. The contour lines are spaced at intervals of k_bT, with blue to red representing high to low free energy values

To identify top predictions several distance based methods like Global Distance Test (GDT), Z-Score or TM-Score as well as using a human expert visual evaluation (97, 102) are used. For example, for *ab initio* methods in CASP11 two types of score were produced 1) to compare prediction models to random models (random ratio) and 2) to compare prediction models to top templates (template ratio) (44). The random ratio used to detect promising model predictions corresponds to the ratio of the best server or manual group GDT_TS score to the random model score. To evaluate overall prediction quality of each predictor group the combinations of six scores (GDT_TS (96), TenS (43), QCS (22), ContS (84), lDDT (59), and MolProb (17)) was used. Scores were provided by The Prediction Center for every prediction. Significance scores for CASP11 *ab initio* ranks included bootstraps and T-tests (44).

The considerable progress in prediction quality has been observed for comparative modeling and fold recognition. The ab initio remains the most challenging of the CASP experiment categories. Even though *ab initio* predictions result in the low quality of models they have been verified to be successful on targets where other methods fail.

To improve de novo prediction secondary structure prediction, fragments and motifs identification, contact prediction and assembly of folds from fragments have been applied. These methods usually are combined with the search methods based on the molecular dynamics, Monte Carlo optimization or genetic algorithms. A few methods use pure *ab initio* simulations together with empirical potentials (64). In CASP11 the most successful participants in the template free category were David Baker with the

Robetta server (18) and Yang Zhang with I-TASSER and QUARK servers (99, 100). Importantly, the use of alignment-based contact prediction methods defined the CASP11 progress. These innovations permitted *de-novo* modeling of larger domain structures.

2.3.3 State-of-the-art De Novo Prediction

The most successful prediction methods in the free modelling category of the CASP11 experiment (**see Table 2.1**) (44) are Robetta (69) and I-TASSER (93). Both methods use short fragments of known structures with similar sequence to build initial models. Random perturbations are applied to these models and the Monte Carlo method (30) is used to find a structure with the minimal energy. In both methods the energy is a weighted sum of knowledge-based potentials. Energy terms weights are optimized on a set of decoys by maximizing the correlation between the value of energy function and the similarity of decoys to the native structure. The decoys are generated by introducing small random changes to the known native structure. Thus, the decoys similar to the native structure have the lowest energy values. The root mean square deviation (RMSD) of euclidean distance between C_α atoms is used to measure structure similarity. However, the described optimization method described has drawbacks: 1) decoys of the native structure are biased towards that structure and potentially over fit the energy function; 2) the linear combination of energy terms assumes that all energy terms equally well discriminate between good and bad candidate structures (in practice, these terms works well for some proteins and bad for the others).

Group Code	Group Name	Dom	Best FM-Style Scoring				First FM-Style Scoring				Best Win/Loss	
			SumZ	SumR	AvgZ	AvgR	SumZ	SumR	AvgZ	AvgR	WinF	P-value
184	**BAKER-ROSETTASERVER**	45	**74.87**	**1**	**1.66**	**1**	27.01	4	0.60	4	0.960	0.0E+00
277	**Zhang-Server**	45	59.11	2	1.31	2	**43.80**	**1**	**0.97**	**1**	0.921	0.0E+00
499	**QUARK**	45	57.86	3	1.29	3	41.44	2	0.92	3	0.907	5.36e-311
41	MULTICOM-NOVEL	45	37.43	4	0.83	4	7.74	13	0.17	13	0.826	5.5E-188
38	nns	45	31.41	5	0.70	6	22.62	5	0.50	5	0.809	4.0E-167
216	myprotein-me	45	29.97	6	0.67	7	17.28	8	0.38	8	0.774	1.9E-129
479	**RBO_Aleph**	40	18.88	7	0.72	5	28.41	3	0.96	2	0.748	3.2E-94
8	MULTICOM-CONSTRUCT	45	15.23	8	0.34	8	15.39	9	0.34	9	0.690	6.7E-62
345	FUSION	45	13.68	9	0.30	9	12.16	12	0.27	12	0.695	4.4E-65
420	MULTICOM-CLUSTER	45	11.00	10	0.24	10	18.51	7	0.41	7	0.638	4.8E-33
268	MULTICOM-REFINE	45	7.67	11	0.17	11	14.66	10	0.33	10	0.643	2.6E-35
410	Pcons-net	45	7.26	12	0.16	12	0.31	17	0.01	19	0.634	4.3E-31
50	RaptorX	45	2.42	13	0.05	13	12.44	11	0.28	11	0.590	5.8E-15
228	BhageerathH	45	1.81	14	0.04	14	6.86	14	0.15	14	0.577	1.5E-11
11	Seok-server	45	1.05	15	0.02	15	20.73	6	0.46	6	0.567	5.7E-09
263	STRINGS	45	1.01	16	0.02	16	-4.51	23	-0.10	26	0.571	6.0E-10
160	ZHOU-SPARKS-X	45	0.94	17	0.02	17	-3.26	21	-0.07	24	0.569	1.9E-09
349	Distill	45	-0.90	18	-0.02	19	-4.23	22	-0.09	25	0.556	9.1E-07
436	Alpha-Gelly-Server	45	-1.57	19	-0.03	20	-15.45	27	-0.34	29	0.524	1.9E-02
251	TASSER-VMT	45	-5.41	20	-0.12	22	-9.53	25	-0.21	27	0.500	5.0E-01
210	BioSerf	42	-8.20	21	-0.05	21	-5.00	24	0.02	17	0.542	2.4E-04
454	eThread	45	-8.43	22	-0.19	24	-16.26	28	-0.36	30	0.472	9.9E-01
212	FFAS-3D	45	-9.03	23	-0.20	25	-2.75	20	-0.06	23	0.446	1.0E+00
452	FALCON_EnvFold	45	-10.03	24	-0.22	26	-0.68	18	-0.02	21	0.459	1.0E+00
381	FALCON_MANUAL	45	-10.45	25	-0.23	27	1.30	15	0.03	16	0.448	1.0E+00
335	FALCON_TOPO	45	-10.71	26	-0.24	28	0.98	16	0.02	18	0.445	1.0E+00
414	FALCON_MANUAL_X	45	-10.80	27	-0.24	29	-0.83	19	-0.02	22	0.442	1.0E+00
466	RaptorX-FM	38	-13.68	28	0.01	18	-11.29	26	0.07	15	0.583	3.3E-11
300	PhyreX	43	-17.43	29	-0.39	31	-17.47	29	-0.31	28	0.387	1.0E+00
145	BioShell-server	44	-17.63	30	-0.36	30	-20.64	32	-0.42	31	0.399	1.0E+00
117	raghavagps-tsppred	45	-21.94	31	-0.49	33	-36.52	38	-0.81	39	0.331	1.0E+00
492	slbio	45	-25.44	32	-0.57	34	-25.43	33	-0.57	34	0.337	1.0E+00
110	MATRIX	42	-25.72	33	-0.47	32	-64.28	43	-0.83	40	0.348	1.0E+00
448	FLOUDAS_SERVER	45	-30.57	34	-0.68	35	-19.08	30	-0.42	32	0.249	1.0E+00
73	SAM-T08-server	31	-30.83	35	-0.15	23	-28.17	36	-0.01	20	0.516	1.2E-01
133	IntFOLD3	45	-34.07	36	-0.76	37	-20.29	31	-0.45	33	0.232	1.0E+00
156	Atome2_CBS	41	-36.95	37	-0.71	36	-35.28	37	-0.67	37	0.261	1.0E+00
237	chuo-fams-server	42	-43.10	38	-0.88	38	-43.27	40	-0.89	42	0.187	1.0E+00
171	MUFOLD-Server	44	-43.23	39	-0.94	39	-36.67	39	-0.79	38	0.176	1.0E+00
346	HHPredA	45	-44.17	40	-0.98	40	-27.30	35	-0.61	36	0.151	1.0E+00
279	HHPredX	45	-45.13	41	-1.00	42	-27.23	34	-0.61	35	0.160	1.0E+00
22	3D-Jigsaw-V5_1	36	-53.76	42	-0.99	41	-49.74	41	-0.88	41	0.157	1.0E+00
206	PSF	24	-71.02	43	-1.21	43	-63.49	42	-0.90	43	0.095	1.0E+00
193	FFAS03	27	-74.65	44	-1.45	44	-69.75	44	-1.25	44	0.044	1.0E+00

Table 2.1 (44): FM (free modelling, ab initio) Server Group Performance

2.3.4 Robetta algorithm

Before CASP11 experiment, Robetta algorithm sampled a diverse set of fragment assembly followed by all-atom refinement, and selection of final models based on clustering and Rosetta all-atom energy. This approach for high accuracy models has been limited to small proteins (<100 residues) due to enormous size of the conformational search space. In CASP11, Rosetta employed residue-residue co-evolution derived restraints (69) during sampling and refinement to direct the search towards the native conformation. CASP11 demonstrated that co-evolution derived contacts increase structure prediction accuracy.

2.3.4.1 Residue-residue co-evolution

Two or more proteins can bind together and form a complex to perform various functions. Solving the structures of these complexes remains a challenge even if the structures of the protein subunits are known. Ovchinnikov et al (68) designed an algorithm to predict which parts of the proteins make contact with each other in a protein complex. The similar algorithm is incorporated into Robetta to improve multidomain protein structure prediction (69).

Two amino acids at positions X and Y are co-varied if for any given amino acid at position Y there is often a specific amino acid at position X. It is noticed that when a pair of amino acids co-varied, these two amino acids tends to make contact with each other at the protein–protein interface or the protein multidomain interface (**Figure 2.5**).

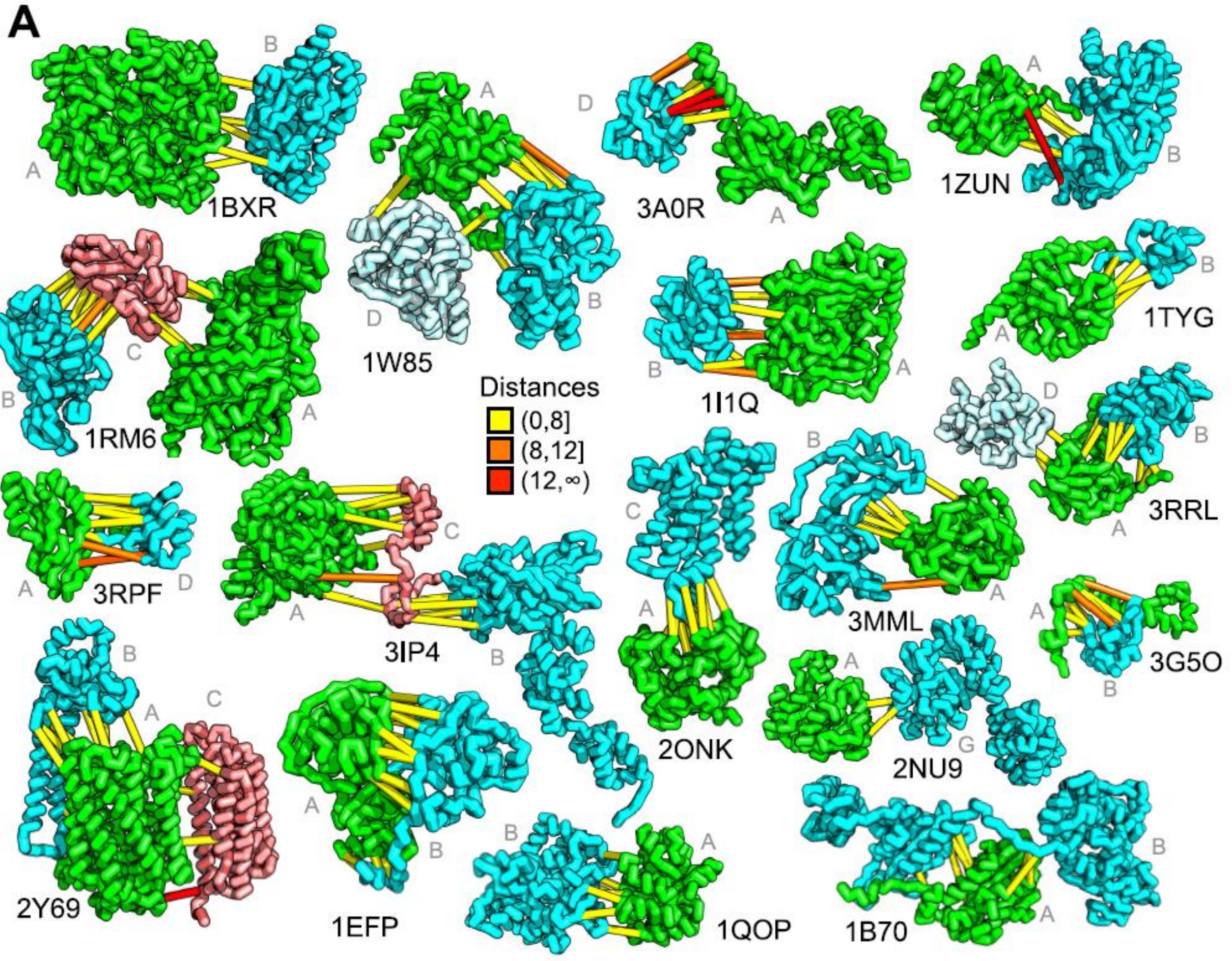


Figure 2.5 (69): Residue covariation in complexes with known structures. Contacts with high GREMLIN scores correlate with residue-pairs across protein interfaces in solved complex structures; the structures are pulled apart for clarity.

2.3.4.2 Robetta’s pipeline

The Robetta algorithm starts with an iterative process of domain boundaries prediction from PDB structures with sequence similarity to the query (**Figure 2.6**) (69). At each iteration step, HHSearch (88), Sparks-X (95), and RaptorX (41) identify templates from pdb database and generate alignments. The sequence is threaded onto the template structures to generate partial models that are clustered to identify distinct topologies. Through this iterative process domain boundaries are assigned. For each domain Robetta finds homologous sequences for the multiple sequence alignment (MSA) to predict residue-residue coevolution contacts. GREMLIN (42) obtains a global statistical model of the prepared MSAs using a pseudo likelihood approach, and contacts are predicted using the residue-residue coupling values derived from the model fitting procedure. The contacts with sequence separation of at least 3 were converted to distance restraints supplement the Rosetta energy function.

2.3.4.3 Rosetta ab initio modeling

The next module of Robetta’s algorithm is attributed to Rosetta ab initio modelling (**Figure 2.7**). Fragments are generated with the use of PSI-BLAST alignments and up to three secondary structure predictions from PSIPRED, SAM-T02, JUFO or PhD. Similarity between the target sequence and a fragment is measured by a sum of similarity scores for the sequence and secondary structure. This fragment library is used in the next step in the fragment insertion. Conformational sampling starts from completely extended chain. It is carried out by a Monte Carlo fragment replacement strategy guided by a low resolution score function that favors protein-like features. Both bond angles and bond lengths are fixed to ideal values, and the side chains are approximated with the center of their mass as a single “centroid” interaction center. Each conformational change starts with random selection of a position in the chain. One of the fragments starting in that position is selected from the library and backbone torsion angles (φ, ψ, ω) from the fragment are applied in that position. Fragment insertion is performed in two stages. First 9-residue fragments are used to construct a rough model, which is refined with 3-residue fragments. The predicted by GREMLIN contacts are used as restraints for sampling and refinement. The energy function used for structure evaluation is composed of Rosetta knowledge based potentials (see Table 2.2): sequence profile, secondary structure (SS), Ramachandran basin, depth dependent structure profile (103), phi and psi torsion (31) and solvent accessibility (31) score terms. Score term weights were optimized from an unpublished benchmark. The number of used potential terms increases gradually with the progress of simulated annealing, starting from steric overlap and finishing with complete potential for the last quart of iterations of 9-residue stage and for the whole 3-residue stage. The best decoy structures are chosen from a top 5% lowest energy subset or cluster analysis is performed to select the representative structures. The chosen decoys structure becomes a starting point for the full-atom refinement. It includes moves like torsion angles perturbation, the fragment insertion which takes only those similar to existing fragment in the model from the fragment database. The refinement also incorporates local fragment gradient descent optimizations and sampling side chains using a backbone-dependent rotamer library (8, 15).

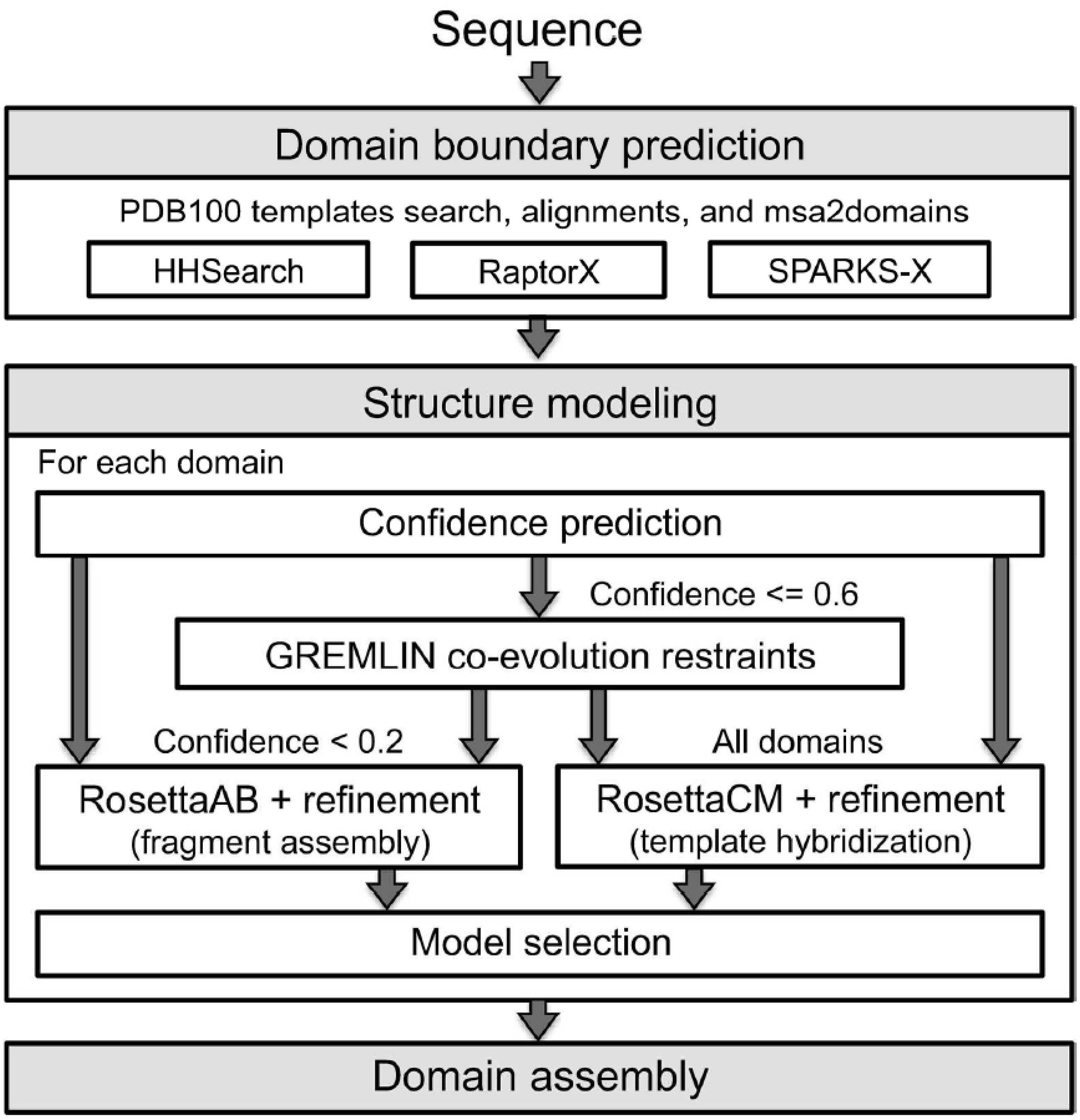


Figure 2.6 (69): Fully automated Rosetta structure prediction protocol.

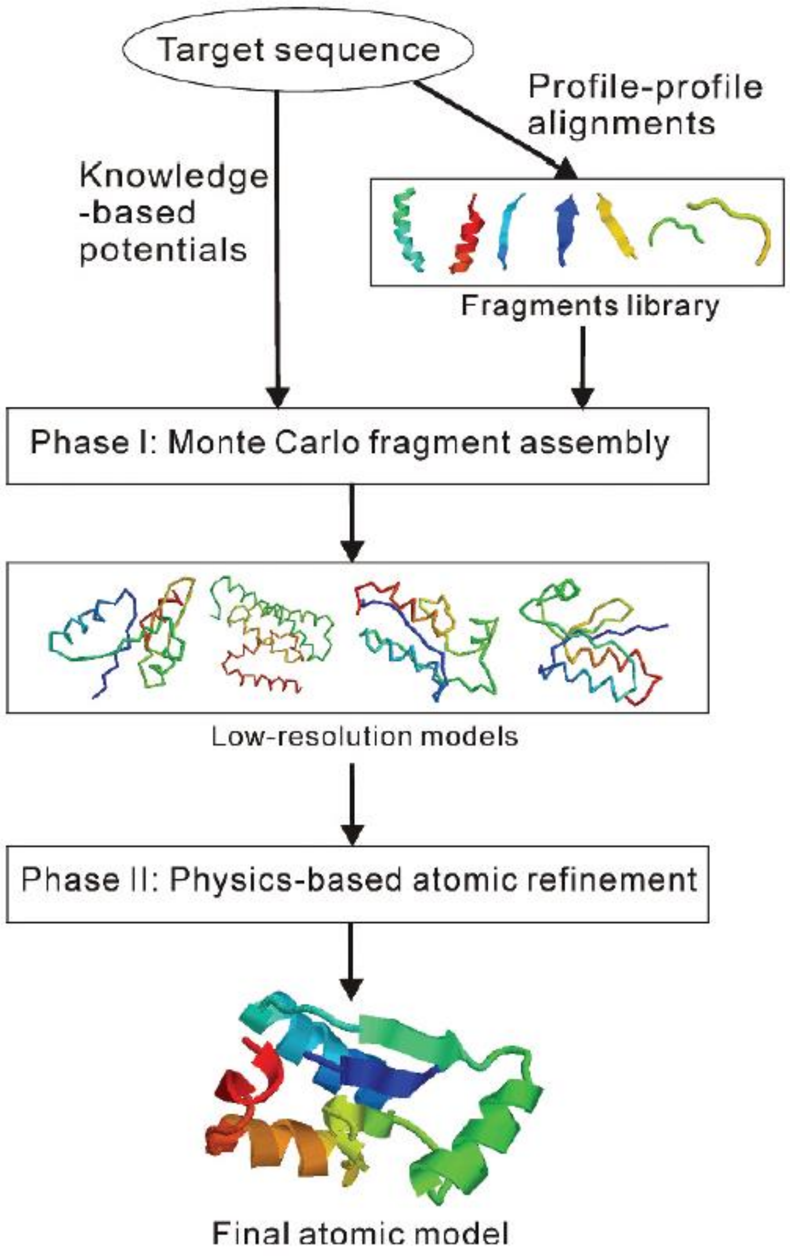

Target sequence
Profile-profile alignments
Knowledge -based potentials
Fragments library
Phase I: Monte Carlo fragment assembly
Low-resolution models
Phase II: Physics-based atomic refinement
Final atomic model

Figure 2.7 (78): Rosetta ab initio modelling

2.3.4.4 Rosetta Energy Function

The components of Rosetta energy function are knowledge/statistical based terms and refer to the probability of “nativeness" of given structure, based on analysis of features of known native structures. A Bayesian model of the likelihood of the structure being a native one, given the sequence of amino acids (Table 2.2, Table 2.3) is used to design Rosetta energy. The statistical analysis of the native structures features is employed to designed the energy terms and describe either the energy of structure independent of sequence (P(structure)) or energy of sequence given particular structure (P(sequence | structure)) (87). The all-atom Rosetta energy function is composed as a linear combination of the Ramachadran torsion preferences, the Lennard-Jones potential, implicit solvation and electrostatic effects, hydrogen bonding and backbone dependent rotamer self-energy potentials. The Lennard- Jones potential utilizes energy parameters used in CHARMM 19 and uses a linear function for repulsion to compensate for fixed rotamer set. The implicit solvation energy is calculated by using Lazaridis and Karplus (50) complete model. Electrostatic interactions are approximated based on PDB statistics. Hydrogen bonding potential depends on secondary structure and orientation of a hydrogen bond. The Van der Waals potential in a low resolution energy function rewards globally compact structures. It is represented by steric repulsion of backbone atoms and side-chain centroids. Solvation potential and hydrogen bonding potentials are based on statistical data.

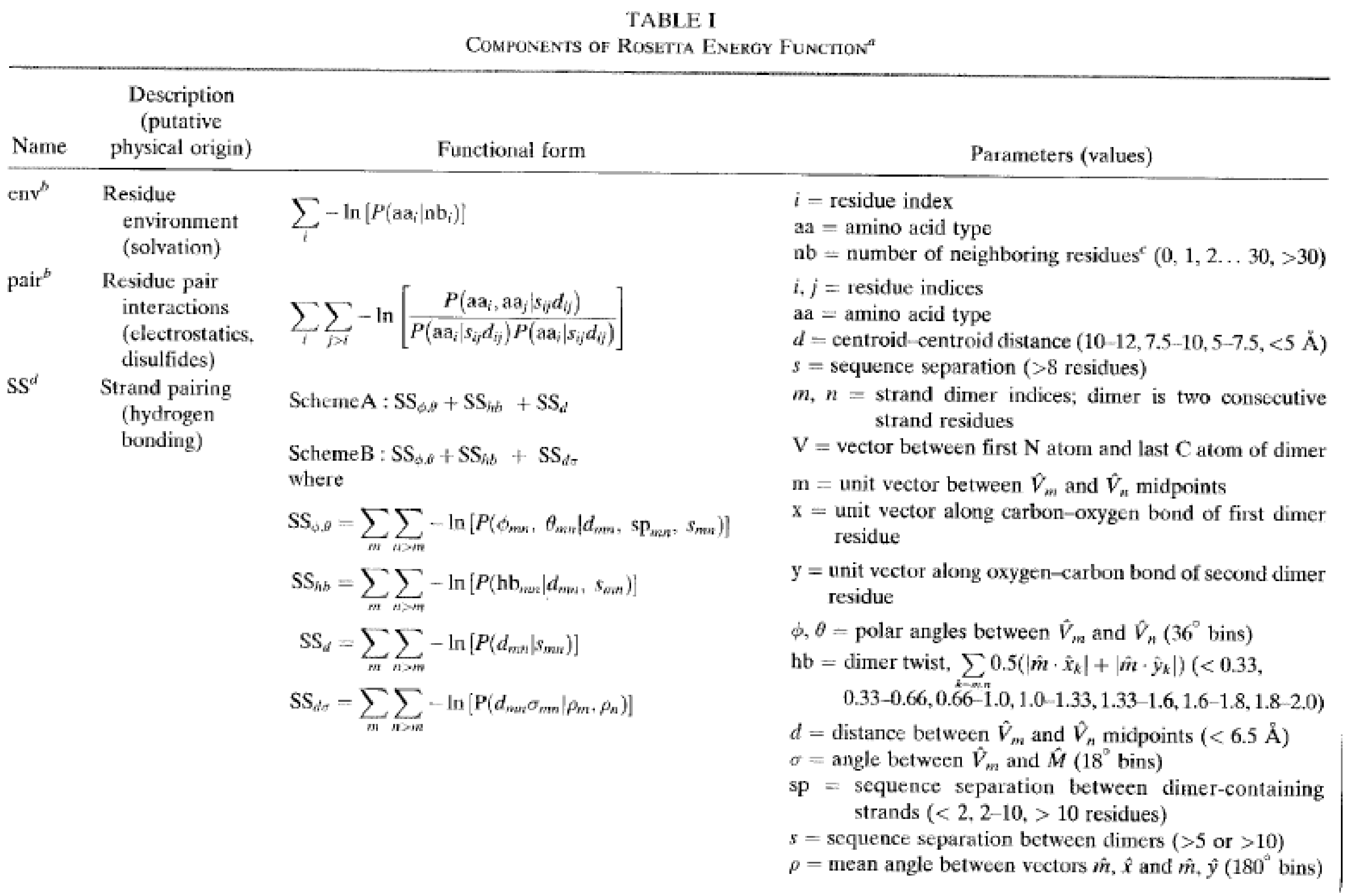


TABLE I
COMPONENTS OF ROSETTA ENERGY FUNCTION[a]

Name	Description (putative physical origin)	Functional form	Parameters (values)
env[b]	Residue environment (solvation)	$\sum_i -\ln[P(\mathrm{aa}_i \mid \mathrm{nb}_i)]$	i = residue index aa = amino acid type nb = number of neighboring residues[c] (0, 1, 2… 30, >30)
pair[b]	Residue pair interactions (electrostatics, disulfides)	$\sum_i \sum_{j>i} -\ln\left[\frac{P(\mathrm{aa}_i, \mathrm{aa}_j \mid s_{ij} d_{ij})}{P(\mathrm{aa}_i \mid s_{ij} d_{ij}) P(\mathrm{aa}_i \mid s_{ij} d_{ij})}\right]$	i, j = residue indices aa = amino acid type d = centroid–centroid distance (10–12, 7.5–10, 5–7.5, <5 Å) s = sequence separation (>8 residues)
SS[d]	Strand pairing (hydrogen bonding)	SchemeA : $\mathrm{SS}_{\phi,\theta} + \mathrm{SS}_{hb} + \mathrm{SS}_d$ SchemeB : $\mathrm{SS}_{\phi,\theta} + \mathrm{SS}_{hb} + \mathrm{SS}_{d\sigma}$ where $\mathrm{SS}_{\phi,\theta} = \sum_m \sum_{n>m} -\ln[P(\phi_{mn}, \theta_{mn} \mid d_{mn}, \mathrm{sp}_{mn}, s_{mn})]$ $\mathrm{SS}_{hb} = \sum_m \sum_{n>m} -\ln[P(\mathrm{hb}_{mn} \mid d_{mn}, s_{mn})]$ $\mathrm{SS}_d = \sum_m \sum_{n>m} -\ln[P(d_{mn} \mid s_{mn})]$ $\mathrm{SS}_{d\sigma} = \sum_m \sum_{n>m} -\ln[\mathrm{P}(d_{mn}\sigma_{mn} \mid \rho_m, \rho_n)]$	m, n = strand dimer indices; dimer is two consecutive strand residues V = vector between first N atom and last C atom of dimer m = unit vector between $\hat{V}_m$ and $\hat{V}_n$ midpoints x = unit vector along carbon–oxygen bond of first dimer residue y = unit vector along oxygen–carbon bond of second dimer residue ϕ, θ = polar angles between $\hat{V}_m$ and $\hat{V}_n$ (36° bins) hb = dimer twist, $\sum_{k=m,n} 0.5(\lvert\hat{m} \cdot \hat{x}_k\rvert + \lvert\hat{m} \cdot \hat{y}_k\rvert)$ (< 0.33, 0.33–0.66, 0.66–1.0, 1.0–1.33, 1.33–1.6, 1.6–1.8, 1.8–2.0) d = distance between $\hat{V}_m$ and $\hat{V}_n$ midpoints (< 6.5 Å) σ = angle between $\hat{V}_m$ and $\hat{M}$ (18° bins) sp = sequence separation between dimer-containing strands (< 2, 2–10, > 10 residues) s = sequence separation between dimers (>5 or >10) ρ = mean angle between vectors $\hat{m}, \hat{x}$ and $\hat{m}, \hat{y}$ (180° bins)

Table 2.2 (78) : Rosetta energy terms for coarse-grain *ab initio* simulations

sheet[e]	Strand arrangement into sheets	$-\ln[P(n_{\text{sheets}}, n_{\text{lonestrands}} \mid n_{\text{strands}})]$	n_{sheets} = number of sheets $n_{\text{lone strands}}$ = number of unpaired strands n_{strands} = total number of strands
HS	Helix–strand packing	$\sum_m \sum_n -\ln[P(\phi_{mn}, \psi_{mn} \mid sp_{mn} d_{mn})]$	m = strand dimer index; dimer is two consecutive strand residues n = helix dimer index; dimer is central two residues of four consecutive helical residues $\hat{V}$ = vector between first N atom and last C atom of dimer ϕ, θ = polar angles between $\hat{V}_m$ and $\hat{V}_n$ (36° bins) sp = sequence separation between dimer-containing helix and strand (binned < 2, 2–10, >10 residues) d = distance between $\hat{V}_m$ and $\hat{V}_n$ midpoints (< 12 Å)
rg	Radius of gyration (vdw attraction; solvation)	$\sqrt{\langle d_{ij}^2 \rangle}$	i, j = residue indices d = distance between residue centroids
cbeta	Cβ density (solvation; correction for excluded volume effect introduced by simulation)	$\sum_i \sum_{sh} -\ln\left[\frac{P_{\text{compact}}(\text{nb}_{i,sh})}{P_{\text{random}}(\text{nb}_{i,sh})}\right]$	i = residue index sh = shell radius (6, 12 Å) nb = number of neighboring residues within shell[f] P_{compact} = probability in compact structures assembled from fragments P_{random} = probability in structures assembled randomly from fragments
vdw[g]	Steric repulsion	$\sum_i \sum_{j>i} \frac{\left(r_{ij}^2 - d_{ij}^2\right)^2}{r_{ij}};\ d_{ij} < r_{ij}$	i, j = residue (or centroid) indices d = interatomic distance r = summed van der Waals radii[h]

[a] All terms originally described in Refs. 16 and 17.
[b] Binned function values are linearly interpolated, yielding analytic derivatives.

Table 2.2 (continued) (78): Rosetta energy terms for coarse-grain ab initio simulations

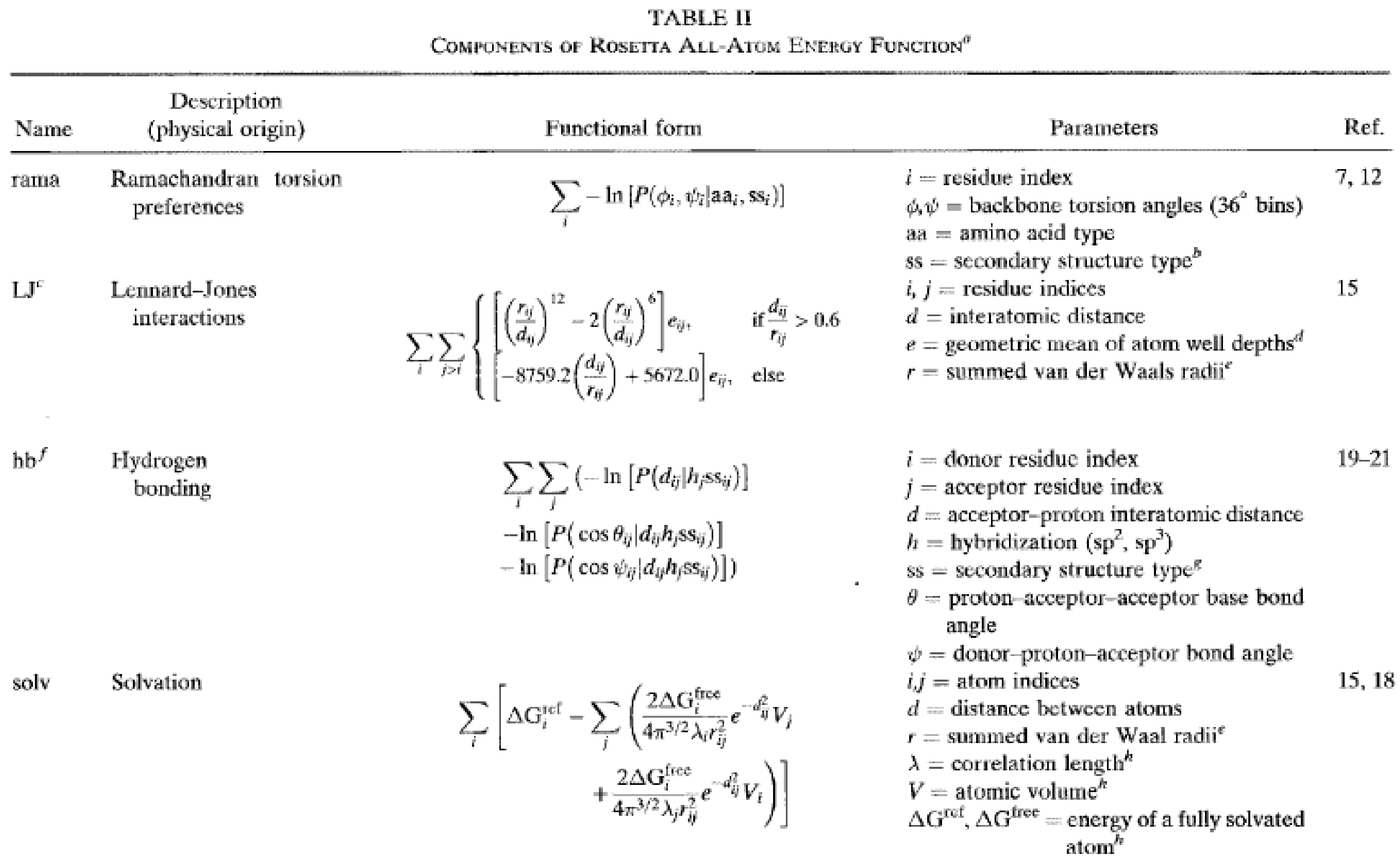


TABLE II

COMPONENTS OF ROSETTA ALL-ATOM ENERGY FUNCTION[a]

Name	Description (physical origin)	Functional form	Parameters	Ref.

rama	Ramachandran torsion preferences	$\sum_i -\ln[P(\phi_i, \psi_i \mid aa_i, ss_i)]$	i = residue index
			ϕ, ψ = backbone torsion angles (36° bins)
			aa = amino acid type
			ss = secondary structure type[b]
				7, 12

LJ[c]	Lennard–Jones interactions	$\sum_i \sum_{j>i} \begin{cases} \left[\left(\frac{r_{ij}}{d_{ij}}\right)^{12} - 2\left(\frac{r_{ij}}{d_{ij}}\right)^{6}\right] e_{ij}, & \text{if } \frac{d_{ij}}{r_{ij}} > 0.6 \\ \left[-8759.2\left(\frac{d_{ij}}{r_{ij}}\right) + 5672.0\right] e_{ij}, & \text{else} \end{cases}$	i, j = residue indices d = interatomic distance e = geometric mean of atom well depths[d] r = summed van der Waals radii[e]	15
hb[f]	Hydrogen bonding	$\sum_i \sum_j (-\ln[P(d_{ij} \mid h_j ss_{ij})]$ $-\ln[P(\cos\theta_{ij} \mid d_{ij} h_j ss_{ij})]$ $-\ln[P(\cos\psi_{ij} \mid d_{ij} h_j ss_{ij})])$	i = donor residue index j = acceptor residue index d = acceptor–proton interatomic distance h = hybridization (sp^2, sp^3) ss = secondary structure type[g] θ = proton–acceptor–acceptor base bond angle ψ = donor–proton–acceptor bond angle	19–21
solv	Solvation	$\sum_i \left[\Delta G_i^{ref} - \sum_j \left(\frac{2\Delta G_i^{free}}{4\pi^{3/2}\lambda_i r_{ij}^2} e^{-d_{ij}^2} V_j + \frac{2\Delta G_i^{free}}{4\pi^{3/2}\lambda_j r_{ij}^2} e^{-d_{ij}^2} V_i\right)\right]$	i,j = atom indices d = distance between atoms r = summed van der Waal radii[e] λ = correlation length[h] V = atomic volume[h] ΔG^{ref}, ΔG^{free} = energy of a fully solvated atom[h]	15, 18

Table 2.3 (78): Rosetta energy terms for ab initio refinement stage

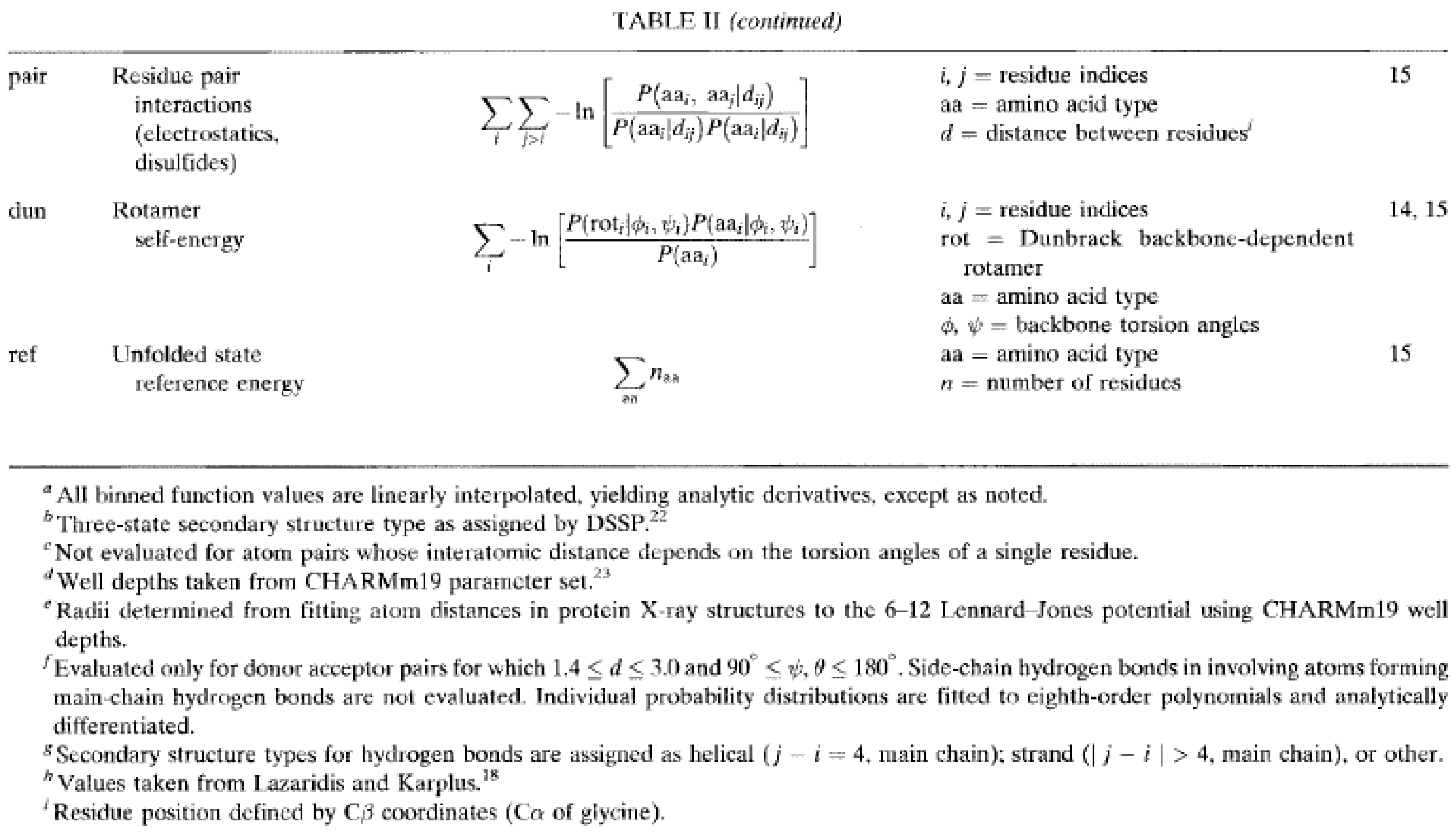


TABLE II *(continued)*

pair	Residue pair interactions (electrostatics, disulfides)	$\sum_i \sum_{j>i} -\ln\left[\frac{P(\text{aa}_i,\ \text{aa}_j \mid d_{ij})}{P(\text{aa}_i \mid d_{ij}) P(\text{aa}_j \mid d_{ij})}\right]$	i, j = residue indices aa = amino acid type d = distance between residues[i]	15

dun	Rotamer self-energy	$\sum_i -\ln\left[\frac{P(\text{rot}_i \mid \phi_i, \psi_i) P(\text{aa}_i \mid \phi_i, \psi_i)}{P(\text{aa}_i)}\right]$	i, j = residue indices
			rot = Dunbrack backbone-dependent rotamer
			aa = amino acid type
			ϕ, ψ = backbone torsion angles
				14, 15

ref	Unfolded state reference energy	$\sum_{aa} n_{aa}$	aa = amino acid type n = number of residues	15

[a] All binned function values are linearly interpolated, yielding analytic derivatives, except as noted.
[b] Three-state secondary structure type as assigned by DSSP.[22]
[c] Not evaluated for atom pairs whose interatomic distance depends on the torsion angles of a single residue.
[d] Well depths taken from CHARMm19 parameter set.[23]
[e] Radii determined from fitting atom distances in protein X-ray structures to the 6–12 Lennard–Jones potential using CHARMm19 well depths.
[f] Evaluated only for donor acceptor pairs for which $1.4 \le d \le 3.0$ and $90^\circ \le \psi, \theta \le 180^\circ$. Side-chain hydrogen bonds in involving atoms forming main-chain hydrogen bonds are not evaluated. Individual probability distributions are fitted to eighth-order polynomials and analytically differentiated.
[g] Secondary structure types for hydrogen bonds are assigned as helical ($j - i = 4$, main chain); strand ($|j - i| > 4$, main chain), or other.
[h] Values taken from Lazaridis and Karplus.[18]
[i] Residue position defined by Cβ coordinates (Cα of glycine).

Table 2.3 (78) (continued): Rosetta energy terms for ab initio refinement stage

Conclusions

Protein structure prediction requires different intricate methods that involve sequence and structure comparison, prediction of secondary structure elements, calculation of solvent accessibility, structure clustering and optimization of complex energy functions. This dissertation touches an aspect of optimization of an energy function. This work is about the design of a new Monte Carlo based approach that optimizes weights for energy terms on a fast folding pathway found with assistance of Gō-like potential. The analysis of the state-of-the-art prediction methods has revealed weak points in the current energy optimization procedures. First, the decoys used in design of the energy functions are biased to the native structure due to the methods of their design. Besides, the process of decoys generation is opposite to the reality of protein structure prediction when decoys have to be built without any knowledge of the native structure. Finally, the linear combination of energy terms often assumes that energy terms are independent, where weights are used to define the relative contribution of terms. However, it is likely that energy terms may correlate with each other. For example, the volume based terms may correlate with van der Waals interactions (VDW). Another example is that hydrogen bonding interactions occur at VDW distances. Thus, it is likely that a linear sum of weighted energy terms cannot capture those covariances, leading to an inaccurate energy (39, 89).

In the following chapters I address the first issue. First, I describe the Monte Carlo based sampling algorithm. It was designed within OOPS software which I optimize for the significant folding speed increase. The algorithm employs effective

Otwinovski’s backbone torsional angles distributions, a soft core Gō potential. Second, I present a new Monte Carlo based energy weights optimization approach, which relies on a fast folding pathway. Several energy terms are tested during the optimization procedure, such as the Lennard-Jones potential (LJ), the Lazaridis-Karplus solvation potential (LK), oriented hydrogen bonding potential (HB), the compactness term and Discrete Optimized Protein Energy (DOPE) term. Three of them, LJ, LK and HB, are shown to be effective. Finally, I apply optimized weights for the energy terms to predict proteins not included into the optimization procedure.

Chapter 3

Sampling algorithm

3.1 All-atom protein representation model

All atoms including hydrogen atoms were explicitly included into simulations from the very beginning. The bond lengths and backbone planar angles, bond lengths were fixed at their mean values. The backbone dihedral angles ω were fixed at their trans conformation, ω = 180°, since ω is observed in the trans conformation more frequently than in cis conformation, ω = 0°. However those ω that are in cis conformation in native structures were held fixed at in cis conformation. The backbone dihedral angles ϕ, ψ and side chain dihedral angles chi varied during simulations.

3.2 Training set

Proteins in different classes may potentially require different weights for each of the energy terms. In order to use an optimized set of weights for any protein regardless of its structure classification, I explored and optimized energy weights by running simultaneously the sampling algorithm and energy weights optimization for representatives of three protein classes (all α, all β, α/β). The overview of three proteins is listed in **Table 3.1** and .their structure shown in **Figure 3.1**

Protein	PDB ID code	Chain	Class	# Residues
crambin	1crn	A	α/β	46
albumin-binding domain	1prb	A	all α	53
ww-domain	1ywj	A	all β	41

Table 3.1. Protein structures used in the training set. The 3 proteins belong to different SCOP classes and have comparable sequence lengths

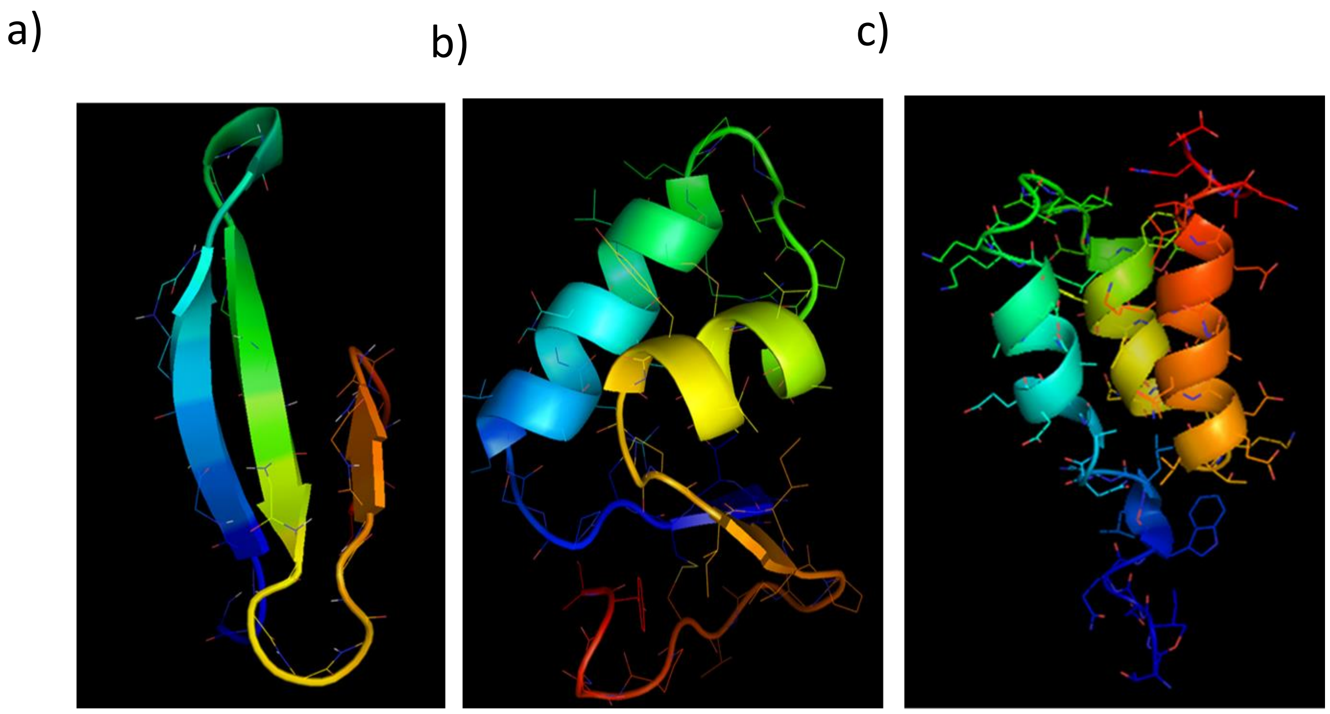


Figure 3.1 Training set for the sampling and the energy weights optimization algorithms..
a) All-beta (WW-domain, 1ywj) – 1μs folding time in solution
b) Alpha/beta (crambin, 1crn) – 1μs folding time in solution
c) All-alpha(albumin-binding domain, 1prb) – 100ns folding time in solution

3.3 Monte Carlo simulations

Conformational space of proteins was explored with Metropolis Monte Carlo method (MMC) (**Figure 3.3**) (61). In the developed algorithm, protein folding simulations always started from a stretched conformation, where all backbone dihedral angles were equal to 180°and side chain dihedral angles were chosen to minimize clashes with a backbone and neighboring side chains. To generate a new conformation I changed dihedral angles of a protein chain as described in the ***Move set*** section below. If the energy of a new conformation was less than the energy of a previous conformation, the former conformation was accepted. Otherwise a new conformation was accepted with probability $\boldsymbol{P_{i,j}}$:

$$\boldsymbol{P_{i,j} = min\left\{1, \exp\left(\frac{-\Delta E_{i,j}}{kT}\right)\right\} \qquad (3.1)}$$

where $\boldsymbol{\Delta E_{i,j}}$is the energy difference between ***i*** and ***j*** .conformations, **k**—the Boltzmann constant, **T**-temperature. The maximum number of MC steps was set to 10^6.

3.4 Scoring function

All-atom Gō potential

Gō potentials favor native state contacts of the target fold and have been used to illustrate the energy landscape. Minimalist Gō-models minimize energetic traps (roughness) on the free energy surface and provide a sufficient model to explain rapid and reliable folding of native sequences.

The Metropolis Algorithm for Monte Carlo (MC) simulations

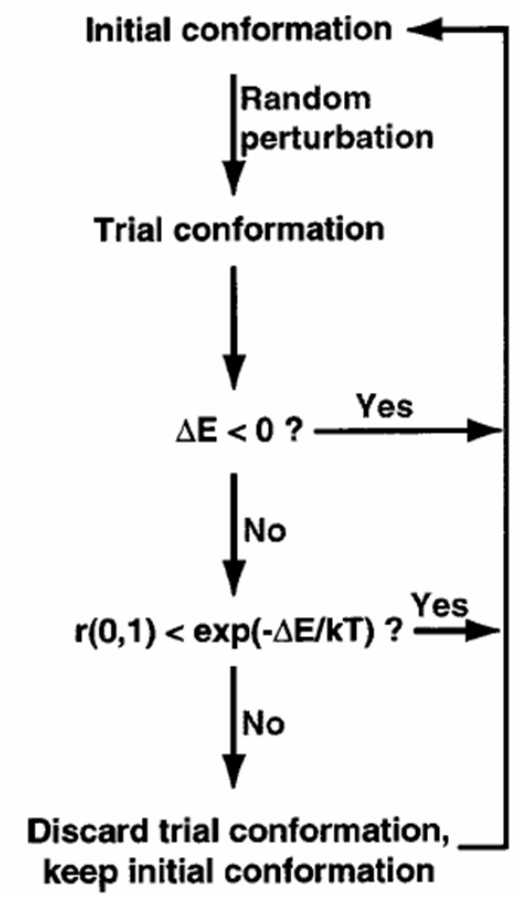

Initial conformation
Random perturbation
Trial conformation
$\Delta E < 0$?
Yes
No
$r(0,1) < \exp(-\Delta E/kT)$?
Yes
No
Discard trial conformation, keep initial conformation

Figure 3.2 Monte Carlo procedure. The initial conformation at the beginning of simulations represents a “stretched” conformation with backbone dihedral angles at 180 °. Side chains dihedral angles are chosen to minimize clashes with a backbone and neighboring side chains. Each random perturbation assigns random psi, phi angles to one randomly chosen amino acid and give random values to all chi angles of all amino acids. A new conformation is accepted if its energy is les s than the energy of a previous one, or satisfies the Boltzmann condition.

A smoothed atom version of an atomic square the Gō potential (32, 85) is used (**Figure 3.4**) for all-atom model protein representation. For two atoms *i* and *j* separated by a distance d, the energy *P_{ij}(d)* was calculated according to the following rules:

If two atoms are in contact in the native conformation, the Gō potential is :

$$\boldsymbol{P_{ij}(d)} = \begin{cases} \boldsymbol{4(d-\sigma)^2}, & \boldsymbol{d < \sigma}, \\ \boldsymbol{-1}, & \boldsymbol{\sigma \le d < \lambda\sigma}, \\ \boldsymbol{-e^{\left(-\left(\frac{d-\lambda\sigma}{2s}\right)^2\right)}}, & \boldsymbol{\lambda\sigma \le d < \lambda\sigma + 3s}, \\ \boldsymbol{0}, & \boldsymbol{d \ge \lambda\sigma + 3s} \end{cases} \qquad (\mathbf{3.2})$$

the Gō potential for contacts that are not present in the native conformation:

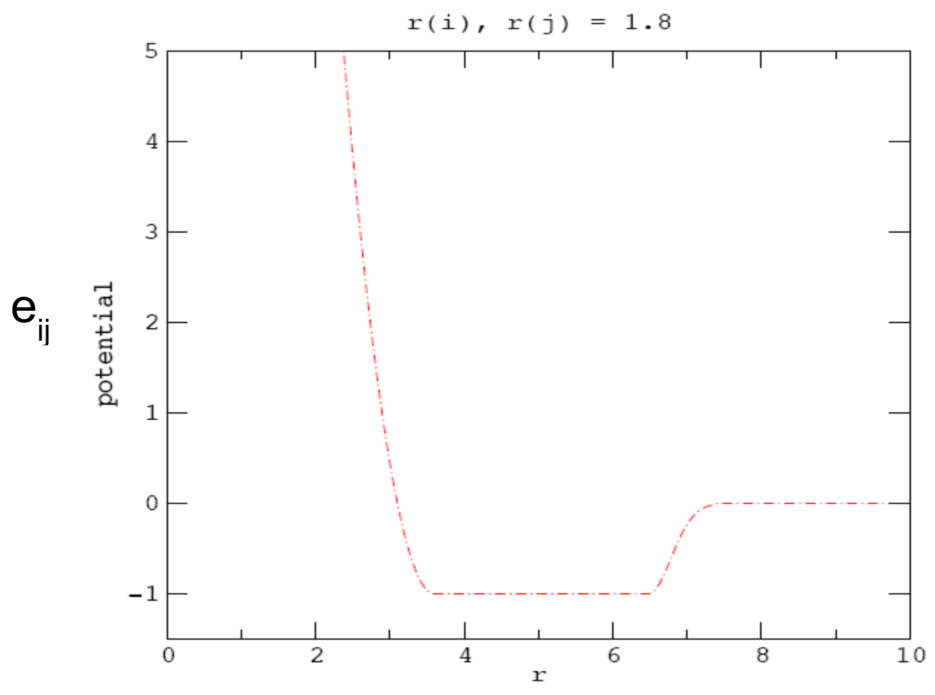
$$\boldsymbol{P_{ij}(d)} = \begin{cases} \boldsymbol{4(d-\sigma)^2}, & \boldsymbol{d < \sigma} \\ \boldsymbol{0}, & \boldsymbol{d \ge \sigma} \end{cases} \qquad (3.3)$$

$$\boldsymbol{\sigma = \alpha \cdot r_{ij}} \qquad (3.4)$$

$$\boldsymbol{s = {}^{0.1 \cdot r_{ij}}\!\big/\!_{\sqrt{2 \cdot ln2}}} \qquad (3.5)$$

where $\boldsymbol{r_{ij}}$ is a sum of Van Der Waals radii. I chose α = 0.66 and *λ=2* since they give the fastest folding time and sufficient number of trajectories with rmsd in range [1.3Å, 3Å] for all three proteins. The total energy of a conformation was computed as the sum over all pairs. The folding results for the three proteins from the training set presented in **Figure 3.4** as scatter plots with RMSD of folded trajectories vs their folding time. I ran 100 trajectories for statistical validation. About 40% of 1crn trajectories, more than 20% of 1prb trajectories and near 8% of 1ywj trajectories were folded with RMSD ≤ 3Å.

a)



b)

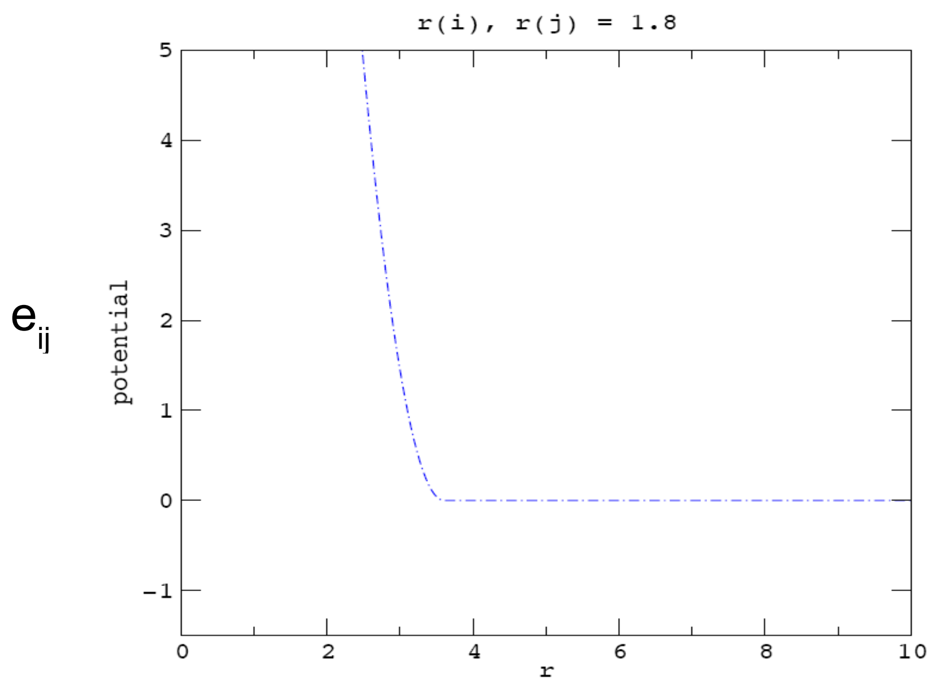
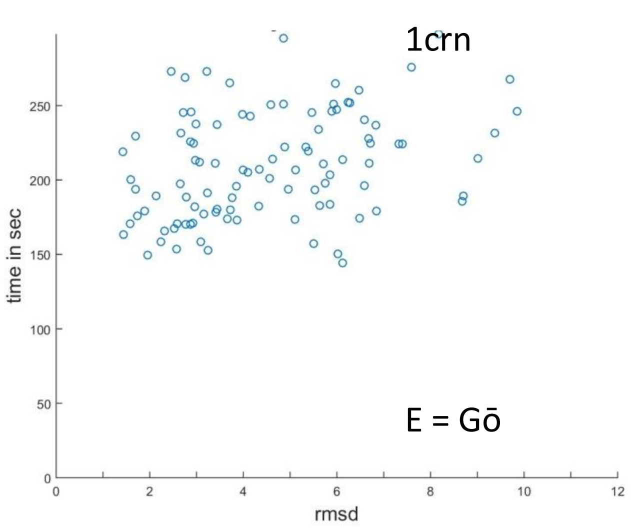
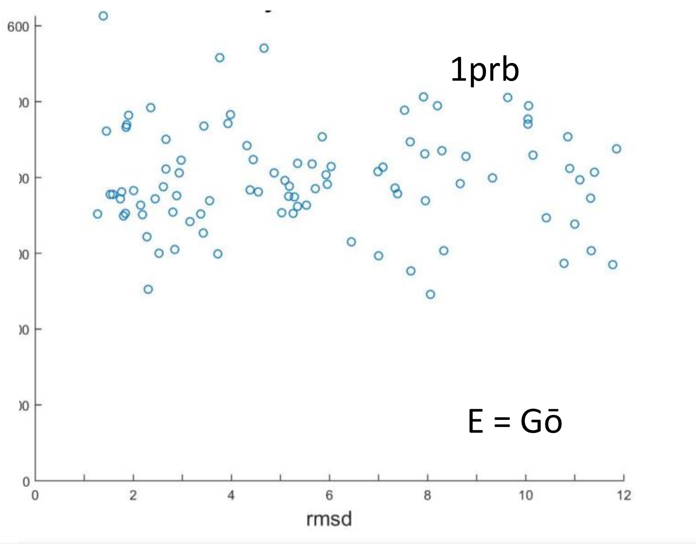


Figure 3.3 Soft Core Gō Potential Function.
e_{ij}- two atoms potential, r-distance between two atoms (Å); a) the Gō potential for native contacts; b) the Gō potential for non-native contacts





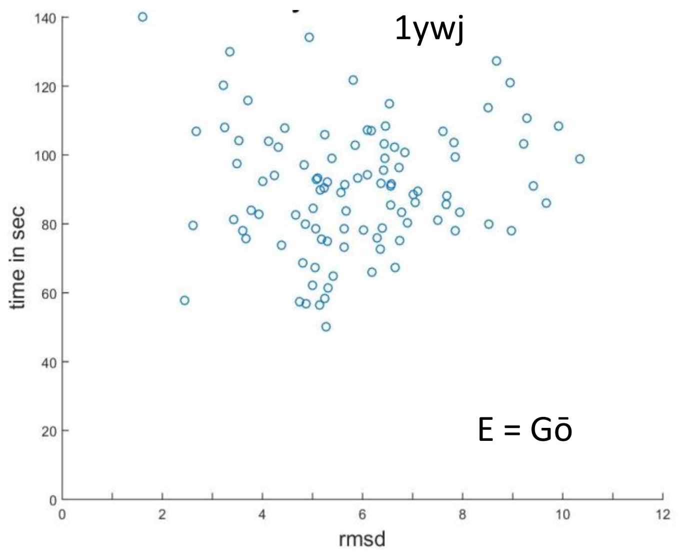


Figure 3.4. Scatter plots of representative runs of RMSD vs folding time for proteins trajectories folded with the Gō potential. Folding results for 1crn, 1prb, 1ywj

The square version of the Gō potential described in (85) was initially tested. However, the energy function was not able to handle steric clashes even being accompanied with different types of backbone and side chains torsional angles distributions (see **The move set** section).

3.4.1 Exploring the Gō Potential well depth

In order to explore the influence of change of the potential well depth during folding simulations, the average number of Monte Carlo steps required to fold a protein (simulations time) was divided into 2 equal periods for the first experiment and into 3 periods for the second experiment. For both studies s I started simulations with the well depth 1; for the next period the depth of the well was increased by 1 (**Figure 3.5**). The idea behind those experiments was to allow accepting more trial conformations in the beginning of simulations and less to the end, accepting conformations that were closer to the native state. Therefore, I forced the algorithm to look for the fastest folding pathway. The results (**Figure 3.6-Figure 3.11**) revealed that for all proteins in the training set the decrease in folding time was accompanied with significant drop in the numbers of folded trajectories which meant that more trajectories were trapped in local minima. Thus, I had to reject this strategy for the sampling algorithm.

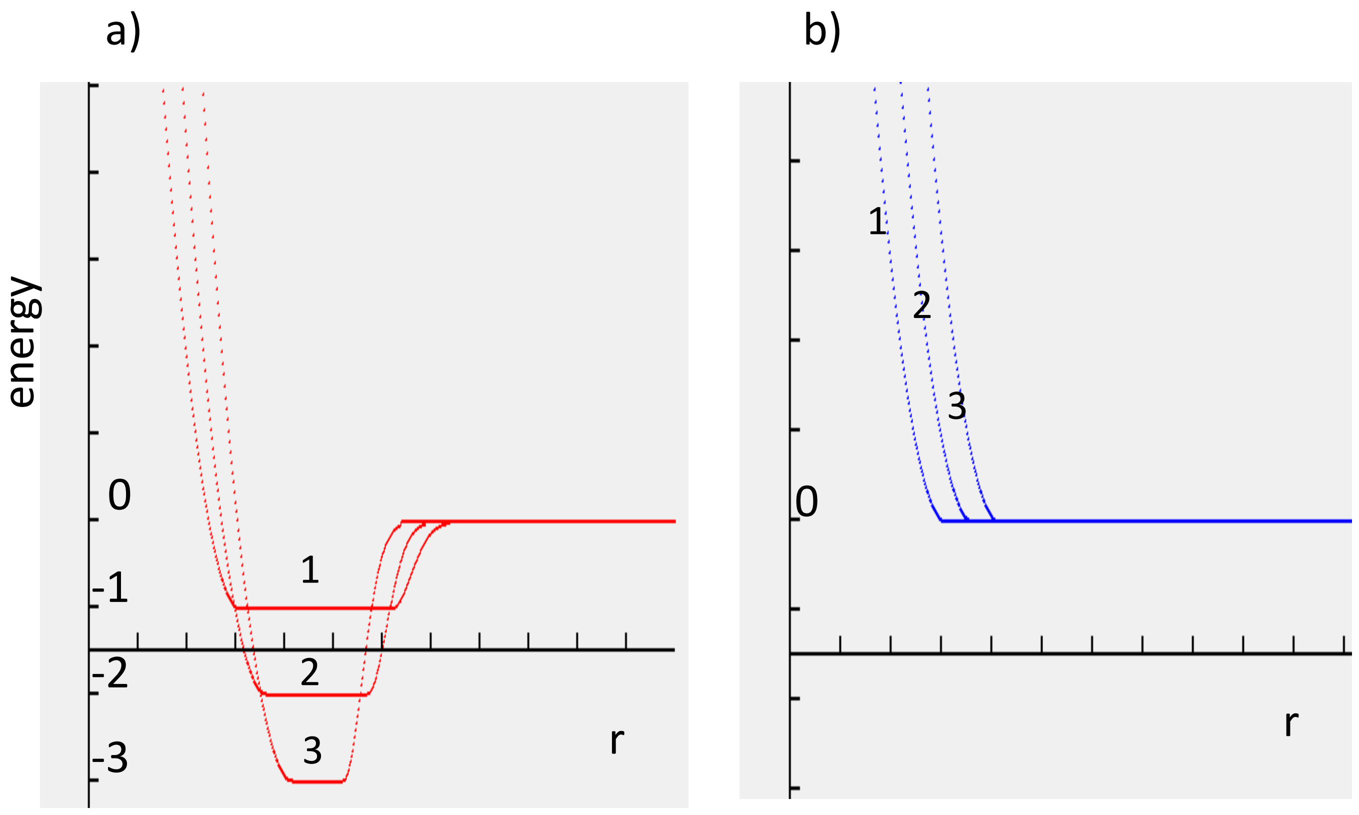


Figure 3.5: Changing soft core Gō potential well depth. During the length of protein folding simulations with the Gō potential as a scoring function, the depth of the potential well was changed from 1 to 3 to accelerate protein folding, which is equivalent to the finding of a fast folding pathway.
a) an atomic pairwise Gō potential for native contacts **b)** an atomic pairwise potential for non-native contacts

a) b)

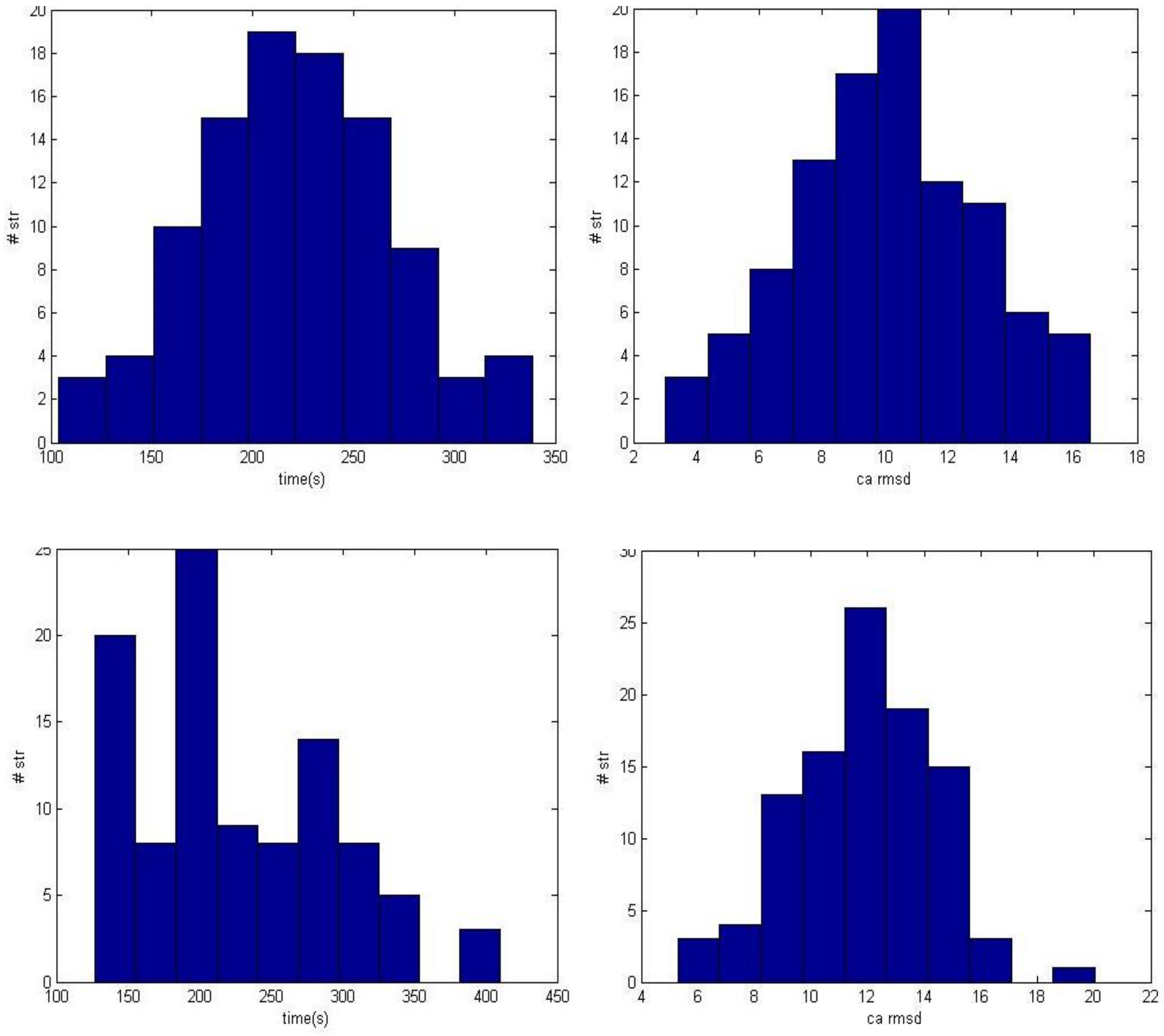
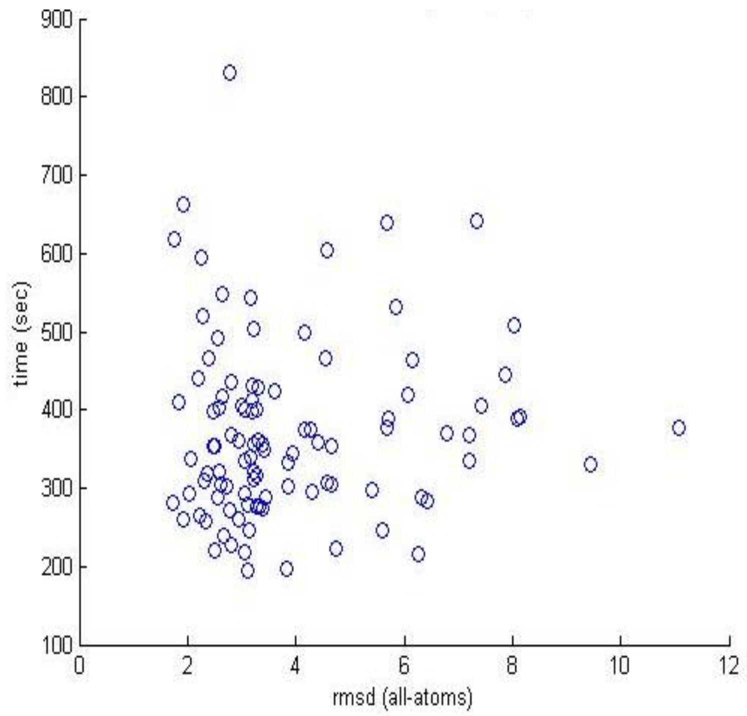


Figure 3.6: Comparison of folding time and RMSD distributions between 2-step and 3-step change of the Gō potential well depth for 1CRN. For both 2-step and 3-step energy well depth change a folding time decrease was accompanied with a folded trajectories (RMSD below 3 Å) dropping off.
a) a top figure demonstrates a folding time distribution for 2-step energy well depth change (the well depth was changed from 1 to 2, see Figure 3.5), a bottom figure shows folding time distribution for 3-step potential well change, see Figure 3.5 **b)** a top figure illustrates RMDS distribution for 2-step energy well depth change, a bottom figure presents results for 3-step potential well depth change, see Figure 3.5

a)



b)

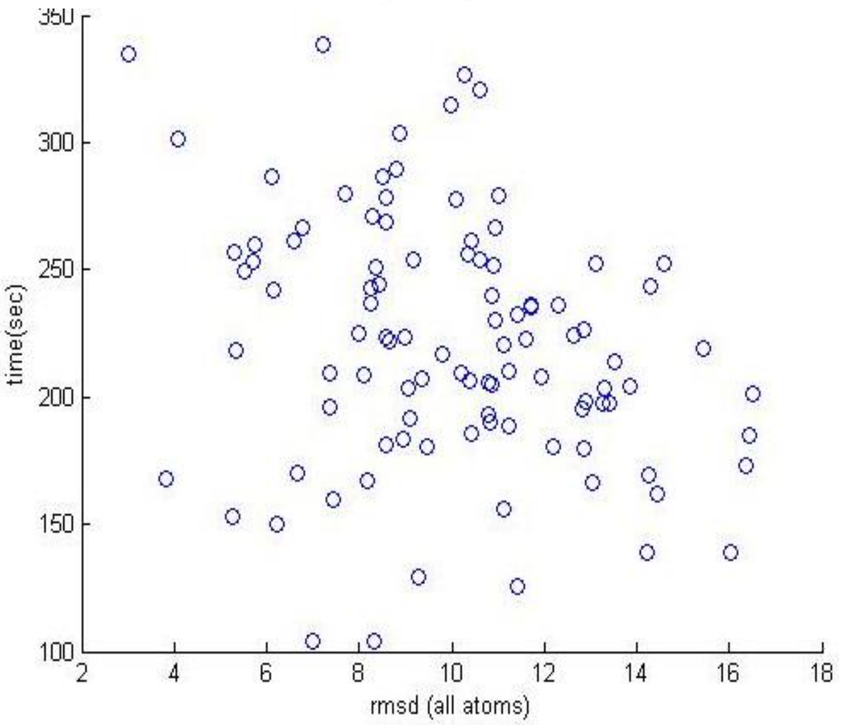
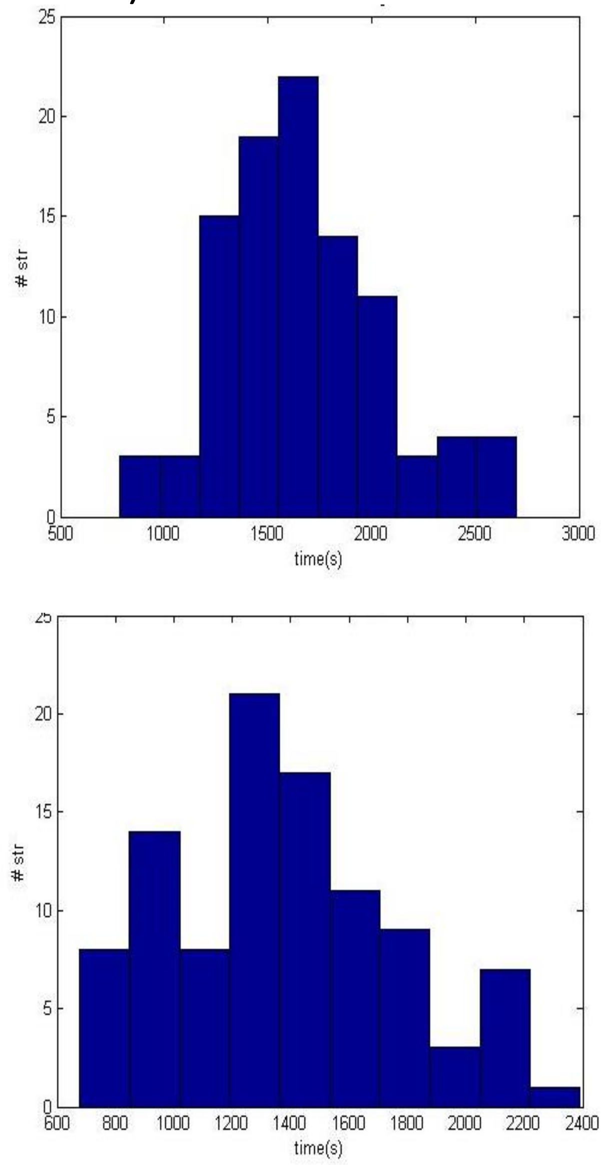


Figure 3.7: Scatter plot RMSD vs folding time comparison between constant Gō potential well depth and 2-step well depth change for 1CRN. Speeding up the sampling algorithm by changing well depth of the Gō potential leads to a fast trapping in local minima. a) a scatter plot of 1crn trajectories with a frozen energy well depth b) a scatter plot for 1crn for 2-step energy well change demonstrates the loss of folded trajectories with RMSD below 3Å.

a)



b)

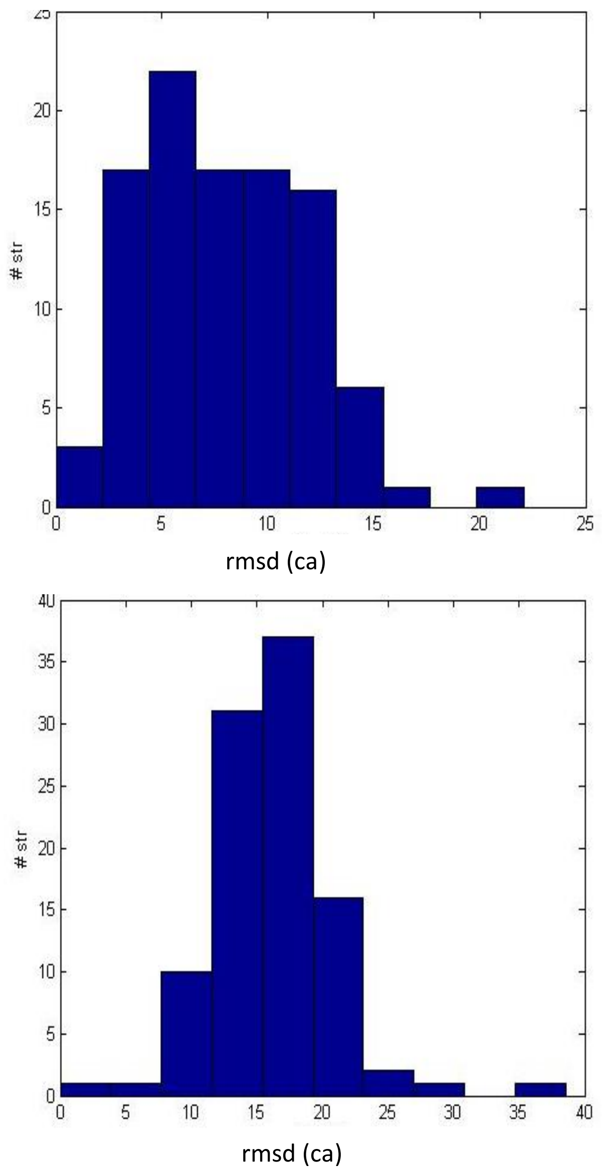
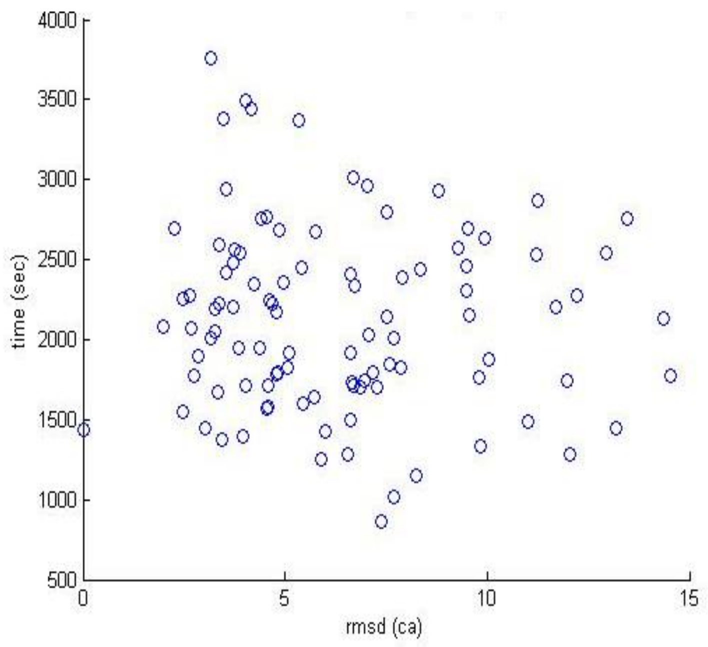


Figure 3.8: Comparison of folding time and RMSD distributions between 2-step and 3-step change of the Gō potential well depth for 1PRB. For both 2-step and 3-step energy well depth change a folding time decrease was accompanied with a folded trajectories (RMSD below 3 Å) dropping off. **a**) a top figure demonstrates a folding time distribution for 2-step energy well depth change (the well depth was changed from 1 to 2, see Figure 3.5), a bottom figure shows folding time distribution for 3-step potential well change, see Figure 3.5 **b)** a top figure illustrates RMDS distribution for 2-step energy well depth change, a bottom figure presents results for 3-step potential well depth change, see Figure 3.5

a)



b)

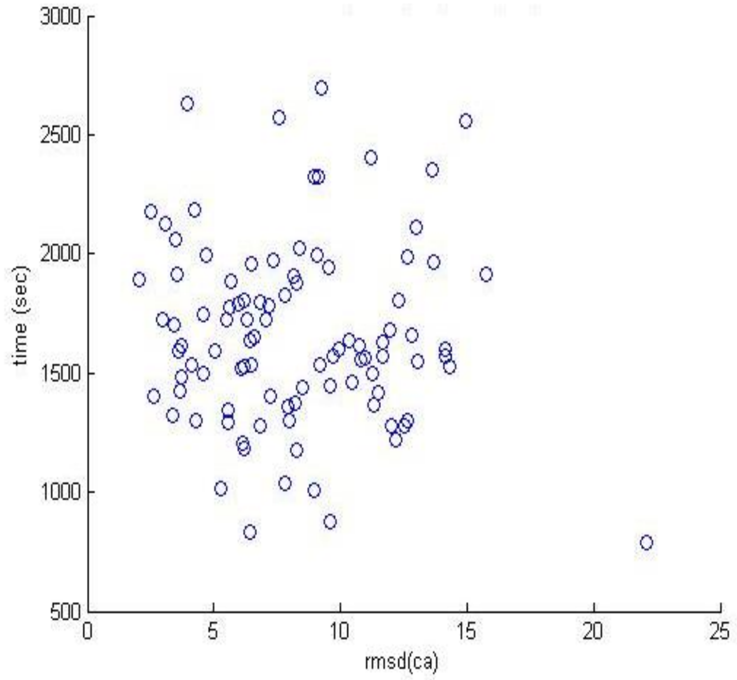


Figure 3.9: Scatter plot RMSD vs folding time comparison between constant Gō potential well depth and 2-step well depth change for 1PRB. Speeding up the sampling algorithm by changing well depth of the Gō potential leads to a fast trapping in local minima. a) a scatter plot of 1prb trajectories with a frozen energy well depth b) a scatter plot for 1prb for 2-step energy well change demonstrates the loss of folded trajectories with RMSD below 3Å.

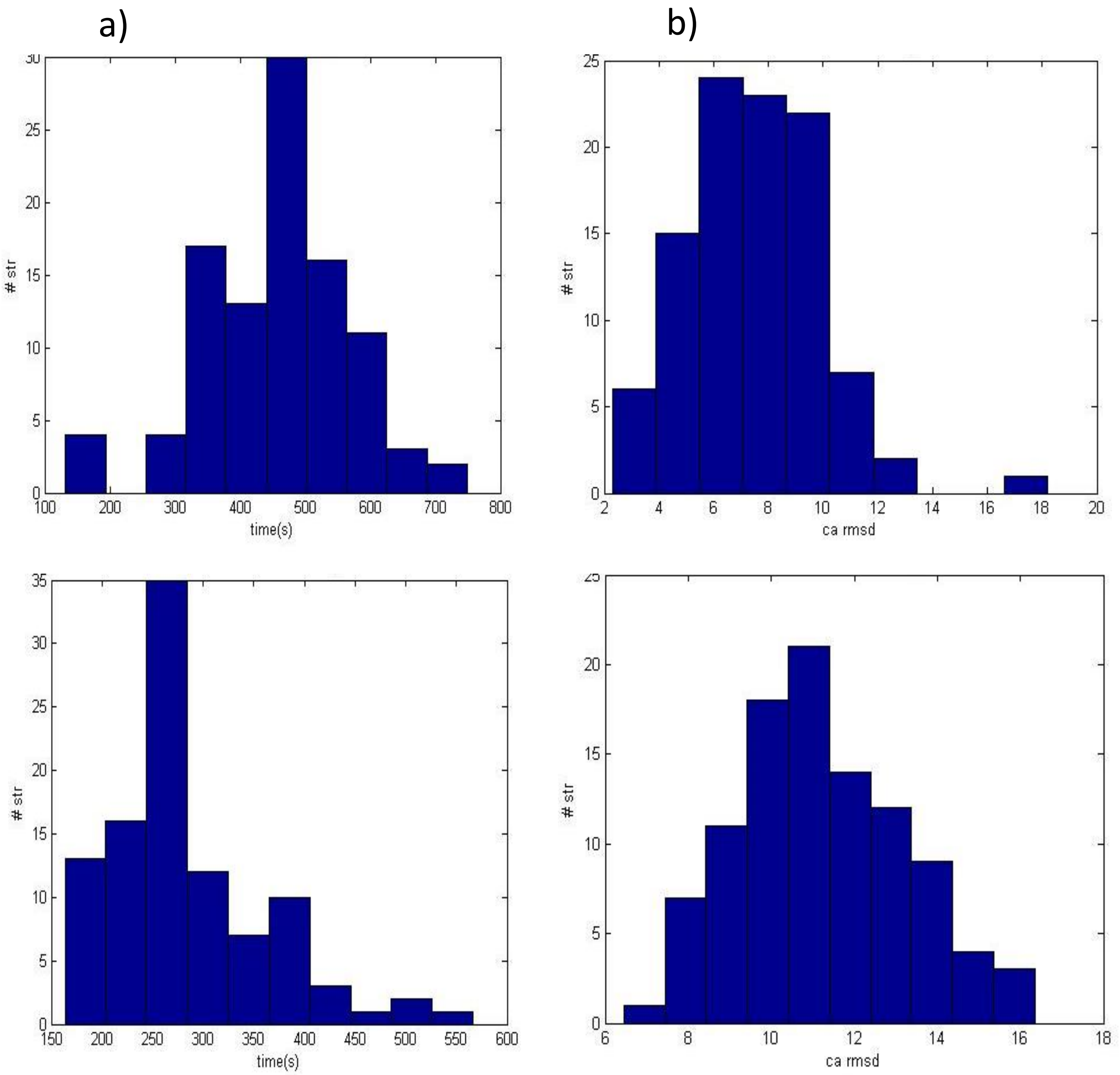
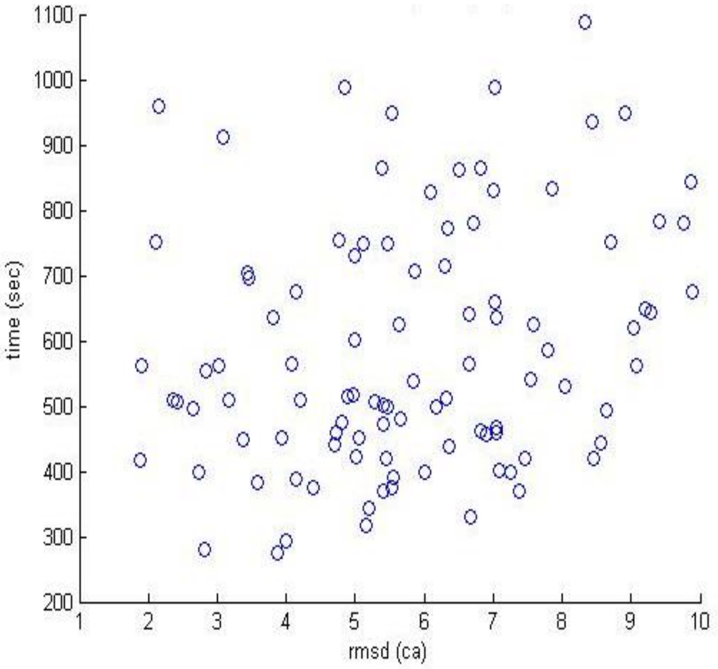


Figure 3.10: Comparison of folding time and RMSD distributions between 2-step and 3-step change of the Gō potential well depth for 1YWJ. For both 2-step and 3-step energy well depth change a folding time decrease was accompanied with a folded trajectories (RMSD below 3 Å) dropping off. **a**) a top figure demonstrates a folding time distribution for 2-step energy well depth change (the well depth was changed from 1 to 2, see Figure 3.5), a bottom figure shows folding time distribution for 3-step potential well change, see Figure 3.5 **b)** a top figure illustrates RMDS distribution for 2-step energy well depth change, a bottom figure presents results for 3-step potential well depth change, see Figure 3.5

a)



b)

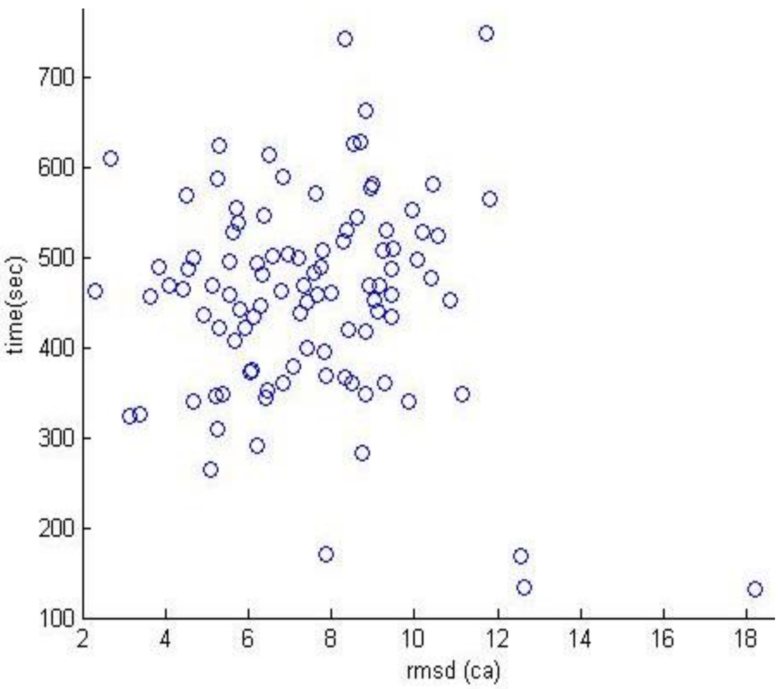


Figure 3.11: Scatter plot RMSD vs folding time comparison between frozen Gō potential well depth and 2-step well depth change for 1YWJ. Speeding up the sampling algorithm by changing well depth of the Gō potential leads to a fast trapping in local minima. a) a scatter plot of 1ywj trajectories with a frozen energy well depth b) a scatter plot for 1ywj for 2-step energy well change demonstrates the loss of folded trajectories with RMSD below 3Å.

3.5 The move set

A fast sampling algorithm needs a simple effective move to sustain its speed. The choice of such move should be defined by the general design of a protein molecule as a computational object, in spite of the temptation to mimic real protein moves in solution. Taken into an account that the fewer calculations are required to generate a new protein conformation for the faster move, I opted for a simple torsional rotation. It might not be the most effective solution but it was the fastest in terms of our software design. A single MC step consisted of a backbone move and followed by all side-chain moves. A backbone move consisted of the rotations of phi and psi angles of one randomly chosen amino acid. A side-chain move consisted of rotating of the all side-chain torsion angles. Values for the backbone dihedral angles ϕ, ψ were sampled from refined Ramachandran basins (72) at each Monte Carlo step. In contrast to traditional-histogram-based method, continuous and differentiable 2D-distribution functions were derived for backbone torsional angles (ϕ, ψ) for 8 groups of amino acids. Twenty amino acids were grouped as following: (1) Val, Ile; (2) Asp, Asn; (3) Ser, Thr (4) Glu, Gln; (5) Leu, Ala; (6) Gly; (7) Pro; (8) Arg, Cys, His, Lys, Met, Phe, Tyr, Trp. For each group two 2D sampling distribution functions were provided: 1) if an amino acid was not followed by Pro and 2) if an amino acid was followed by Pro (**Figure 3.12 – Figure 3.19**). The backbone-dependent rotamer library (83) was used to sample amino acid side chain conformations. The library contains rotamer frequencies, mean dihedral angles, and variances as a function of the backbone dihedral angles. The backbone-dependent rotamer library is created by using adaptive kernel density estimates for the rotamer frequencies and adaptive kernel regression for the mean dihedral angles and variances.

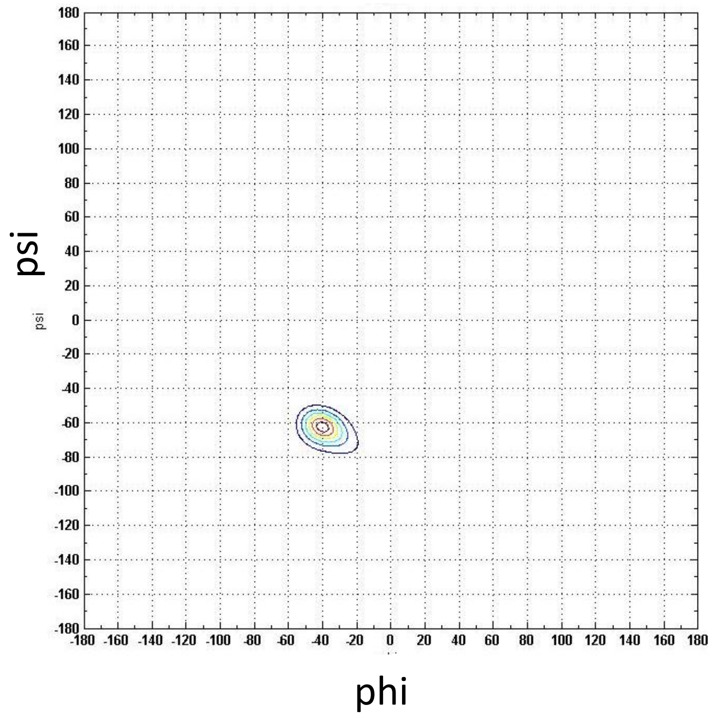
The method gives a smooth and continuous function of phi and psi to evaluate the rotamer probabilities, mean angles, and variances. For the non-rotameric degrees of freedom of amides, carboxylates, and aromatic side chains probability density estimates are presented as a function of the backbone dihedrals and rotamers of the remaining degrees of freedom.

Other distribution and strategies for the move set

Backbone. Uniform and Gaussian sampling distributions were also tested for the development of a fast sampling algorithm. First, values for the backbone dihedral angles were uniformly sampled from [-180°, 180°] interval at each Monte Carlo step during the entire length of simulations. Second, angles were drawn from Gaussian distribution with from 2° to 15° standard deviation at each Monte Carlo step. However, those both sampling significantly slowdown the speed of simulations and do not converge to acceptable final structures (a final structure is expected to gain 70 % of the energy of a native protein structure and has rmsd > 4 A) within 10^{15} MC steps.

Side chains. Uniform and Gaussian sampling distributions were also used to sample torsional angles of amino acids side chains. Similar to backbone torsional angle sampling, values for chi were uniformly sampled from [-180°, 180°] interval at each Monte Carlo step for all amino acids during the entire length of simulations. For sampling from Gaussian distribution angles were drawn with 15° standard deviation at each Monte Carlo step. Those both sampling gave slightly worse performance in comparison to Dunbrack‘s rotamer library.

a)



b)

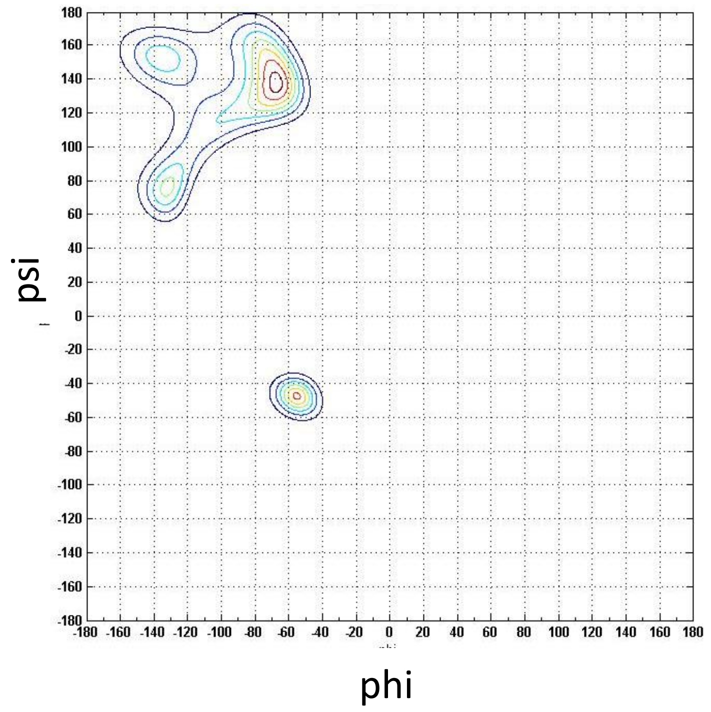
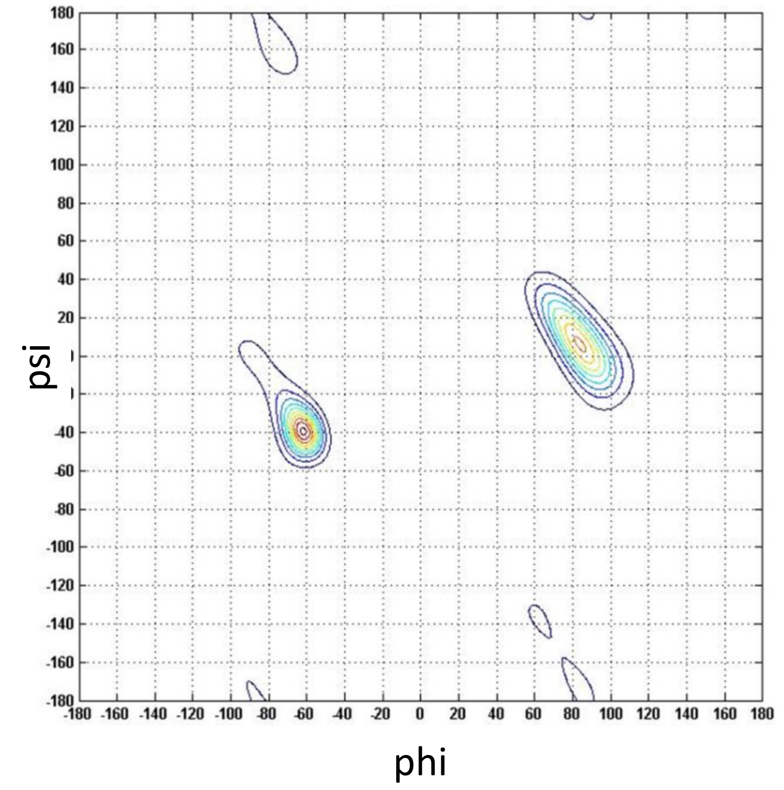


Figure 3.12: Otwinowski's backbone torsional angles distributions for all 20 amino acids. **a)** an amino acid is followed by PRO; **b)** an amino acid is not followed by PRO

a)



b)

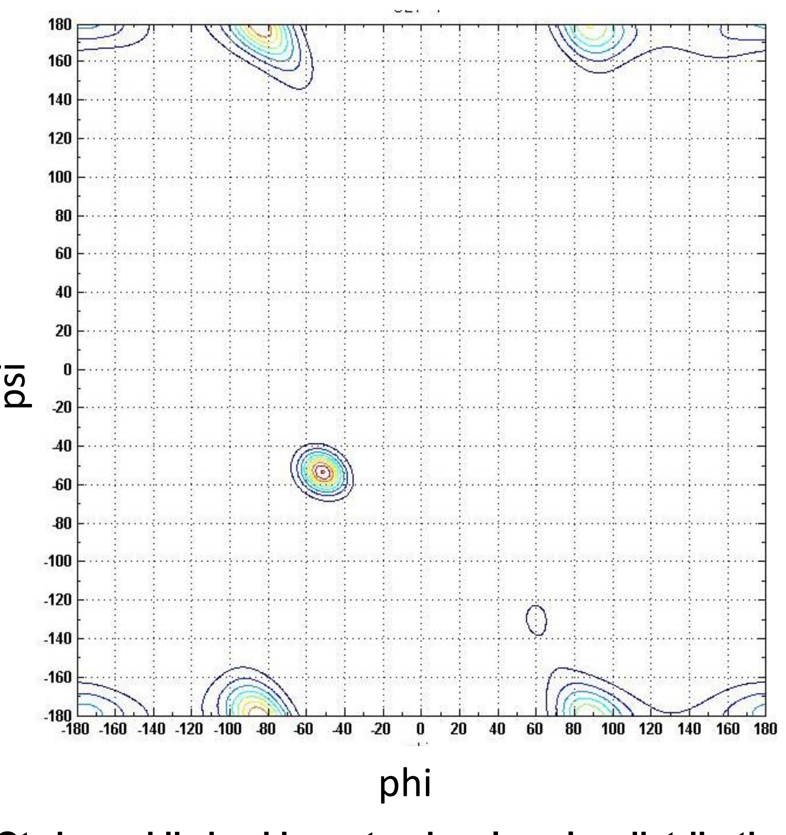
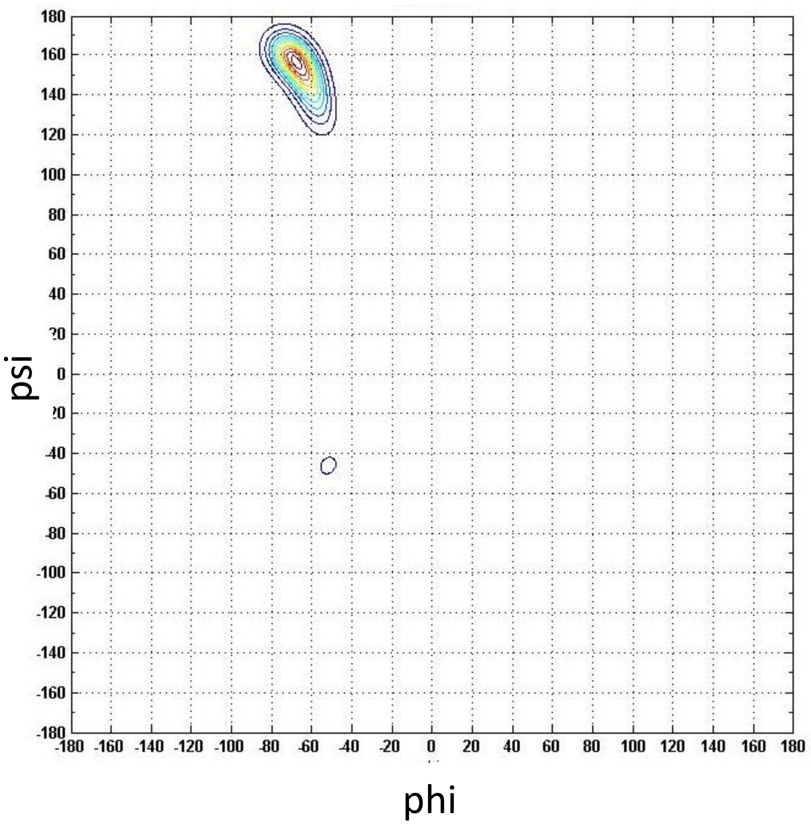


Figure 3.13: Otwinowski's backbone torsional angles distributions for GLY. **a)** GLY is followed by PRO; **b)** GLY is not followed by PRO

a)



b)

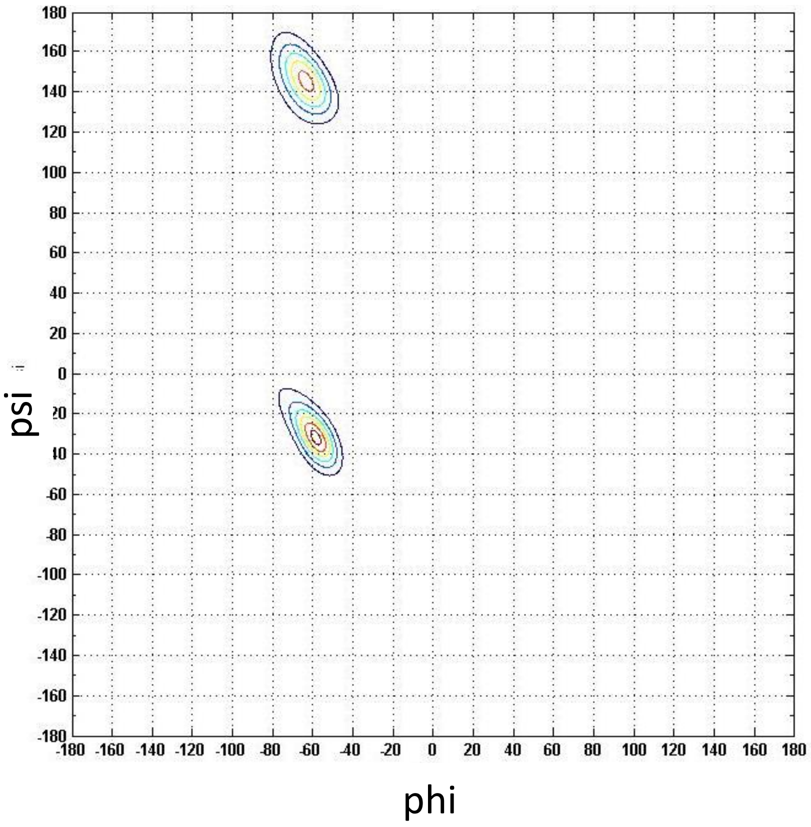
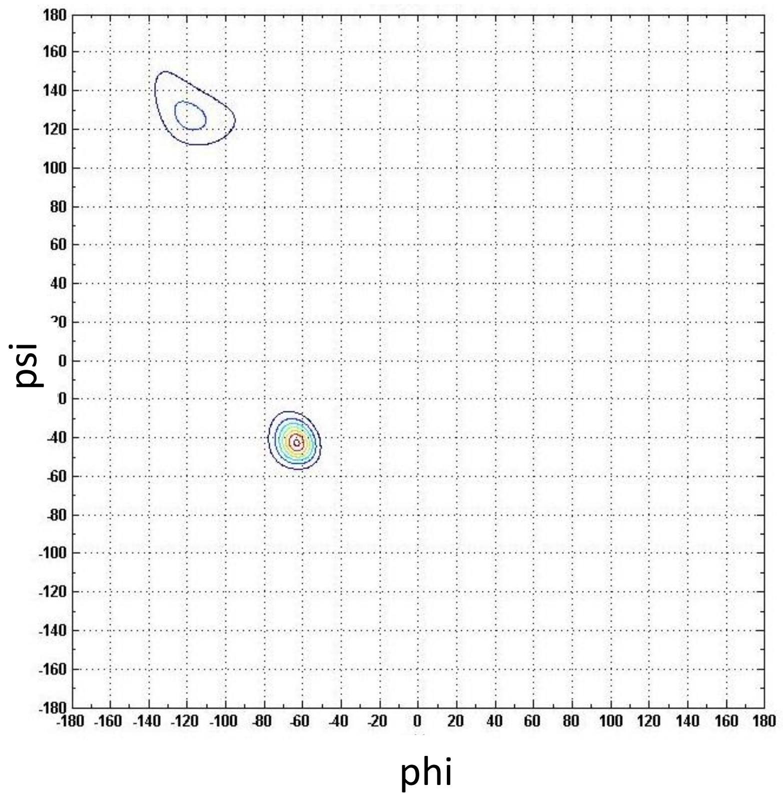


Figure 3.14: Otwinowski's backbone torsional angles distributions for PRO. **a)** PRO is followed by PRO; **b)** PRO is not followed by PRO

a)



b)

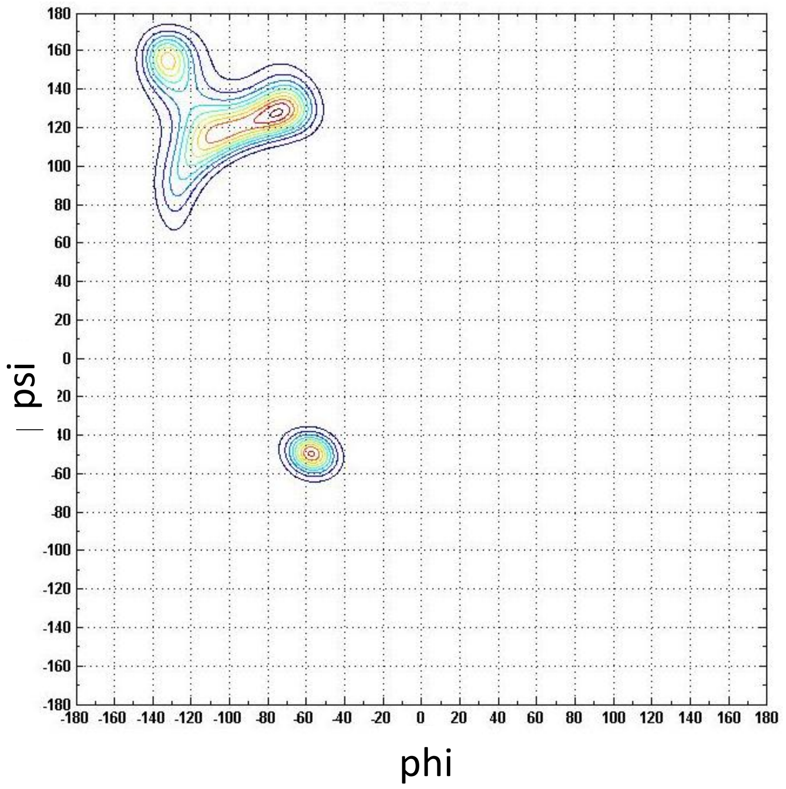
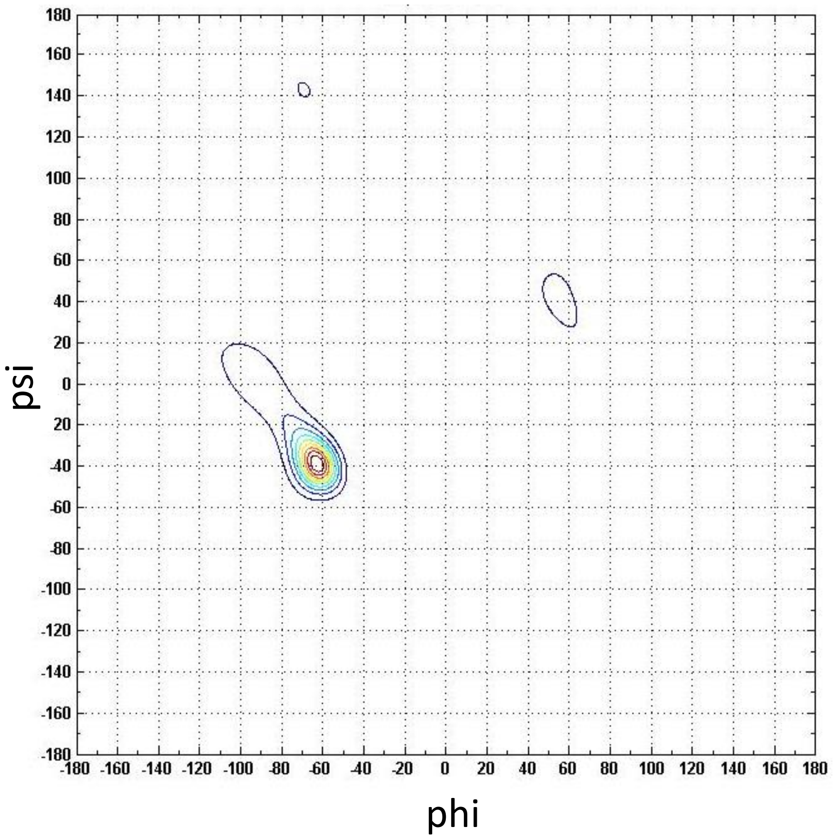


Figure 3.15: Otwinowski's backbone torsional angles distributions for VAL, ILE. a) VAL, ILE are followed by PRO; **b)** VAL, ILE are not followed by PRO

a)



b)

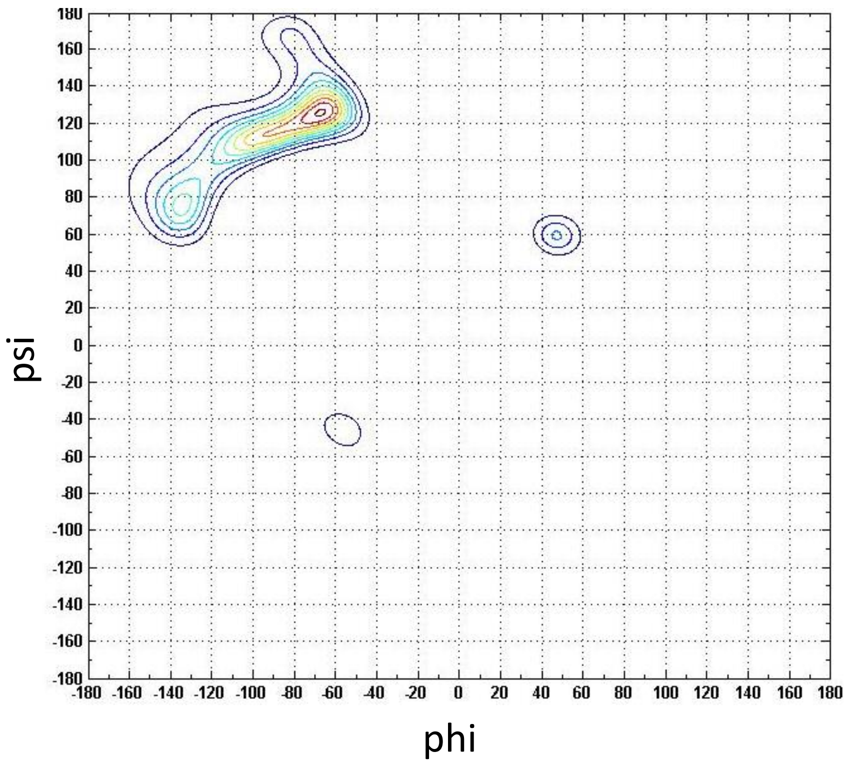
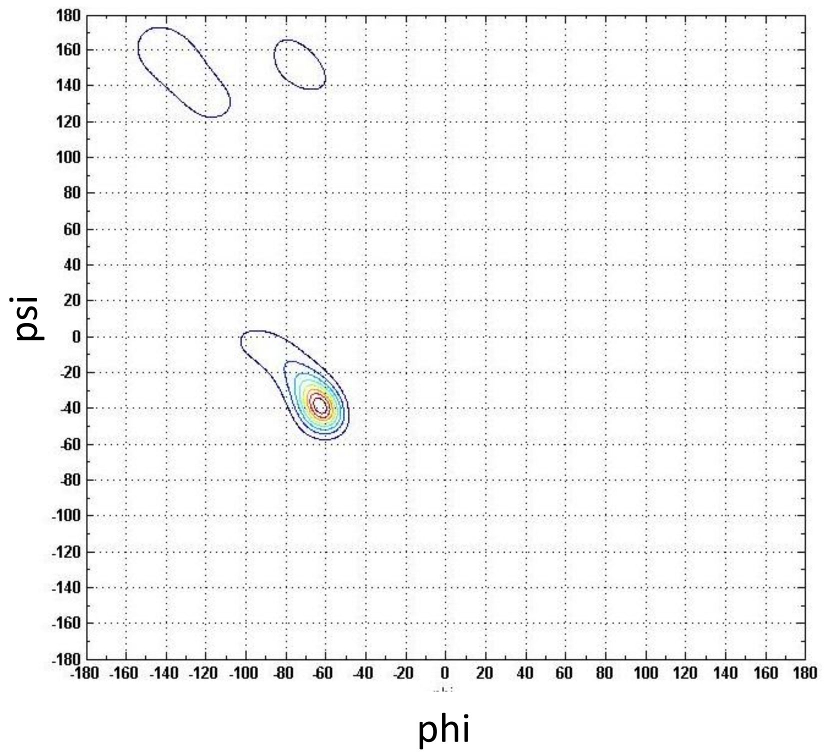


Figure 3.16: Otwinowski's backbone torsional angles distributions for ASP, ASN. **a)** ASP, ASN are followed by PRO; **b)** ASP, ASN are not followed by PRO

a)



b)

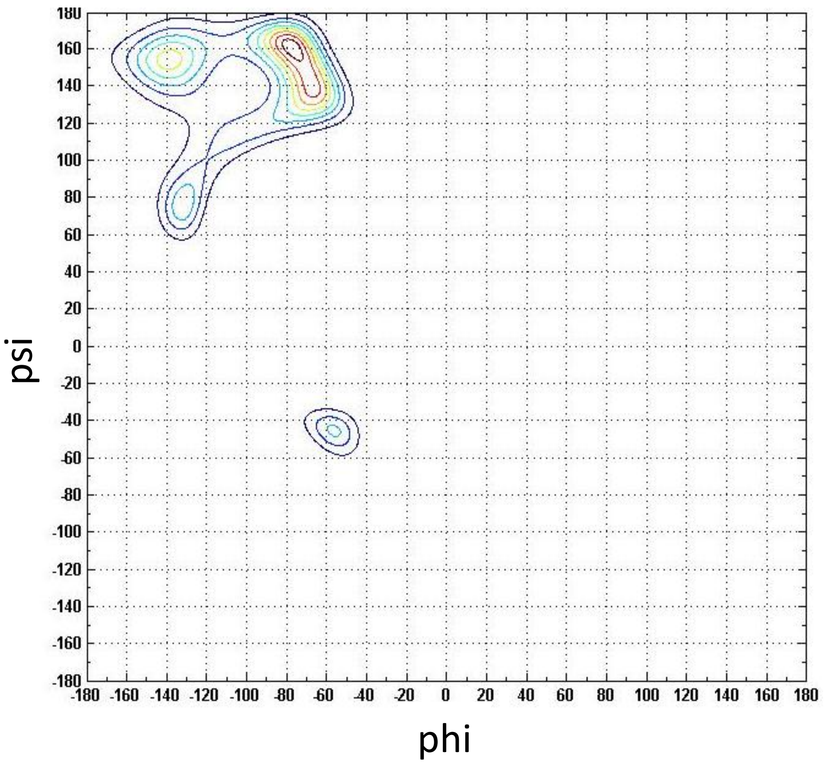
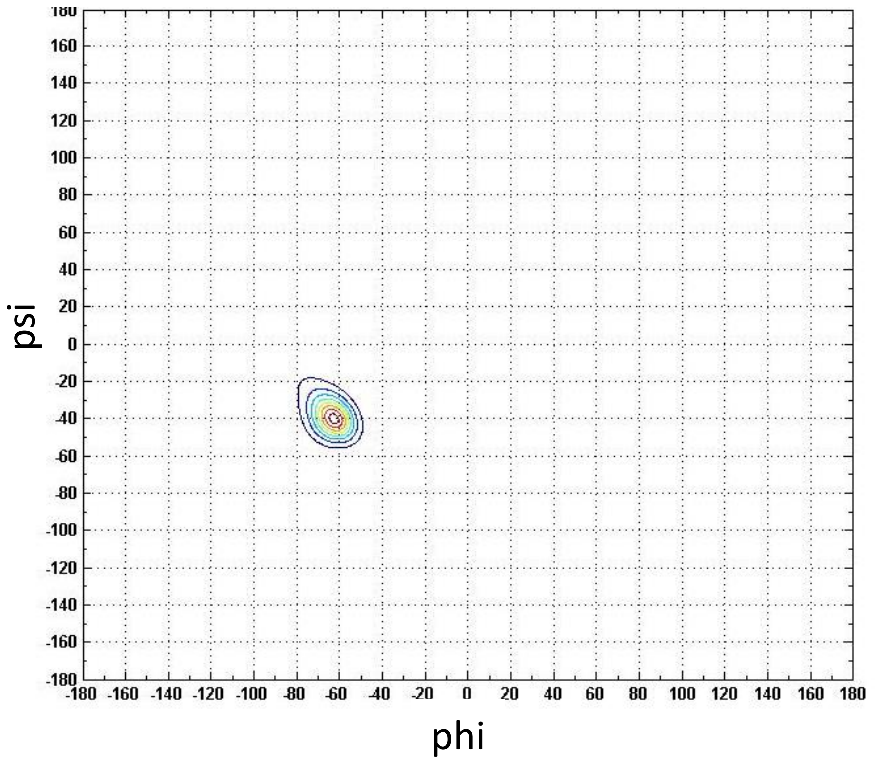


Figure 3.17: Otwinowski's backbone torsional angles distributions for SER, THR. a) SER, THR are followed by PRO; **b)** SER, THR are not followed by PRO

a)



b)

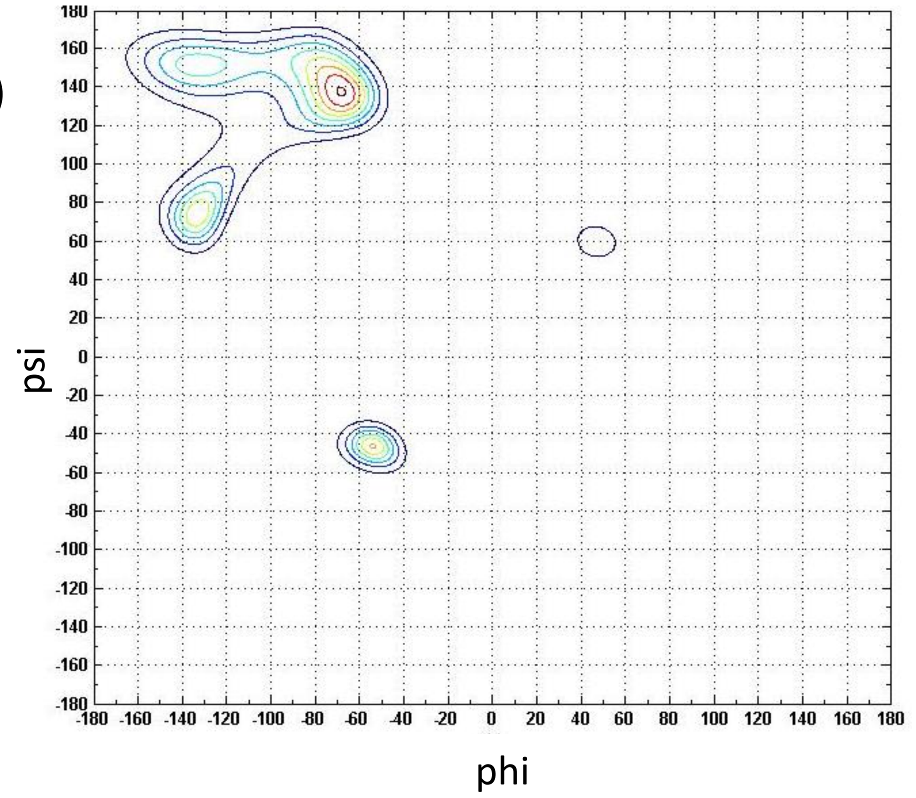
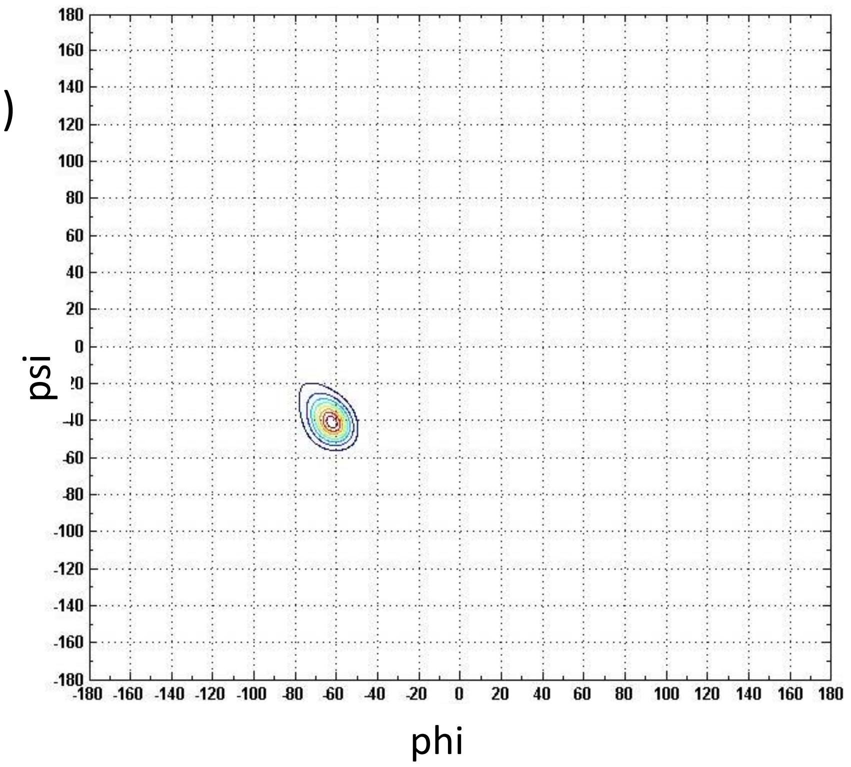


Figure 3.18: Otwinowski's backbone torsional angles distributions for GLU, GLN.
a) GLU, GLN are followed by PRO; **b)** GLU, GLN are not followed by PRO

a)



b)

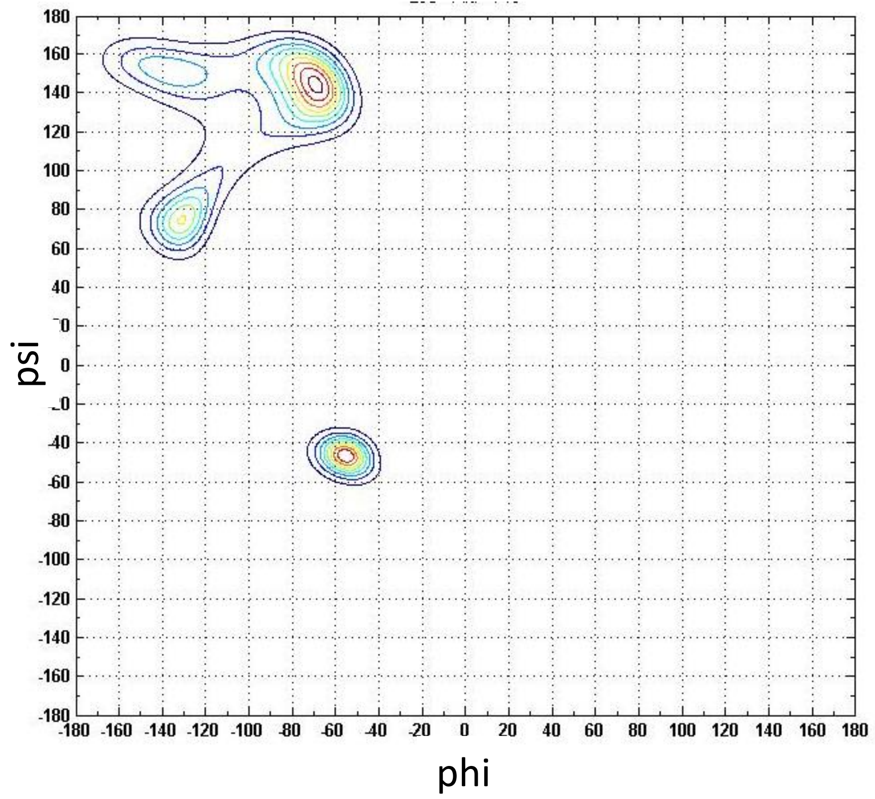


Figure 3.19: Otwinowski's backbone torsional angles distributions for LEU, ALA. a) LEU, ALA are followed by PRO; **b)** GLU, GLN are not followed by PRO

Figures 3.20 – 3.23 illustrate that Dunbrack's rotamer libraries help to increases the number of folded trajectories with RMSD ≤ 3Å and accelerate folding process for all tested proteins. However, this effect is less significant for all-β protein 1ywj which can be explained by overall challenge to predict β-sheets (11).

3.6 Simulated annealing

Monte Carlo procedure **(Figure 3.2**) is accompanied by Simulated Annealing scheme (48) (**Figure 3.24**). Simulated annealing is a local search that uses hill-climbing moves to escape local optima. The concept of the algorithm is inspired by an analogy between the physical annealing process of solids. Annealing is a thermal process for obtaining low energy states of a solid in a heat bath. It consists of two steps (48):

1) the temperature is increased to a maximum value at which the solid melts;

2) the temperature is gradually decreased until the particles arrange themselves in the ground state of the solid.

Particles in the liquid phase are arranged randomly. In the solid (ground) state they are arranged in a structured lattice with the corresponding minimal energy. The ground state is obtained when the initial value of the temperature is sufficiently high and the cooling is slow. Otherwise, the solid will be frozen into a meta-stable state.

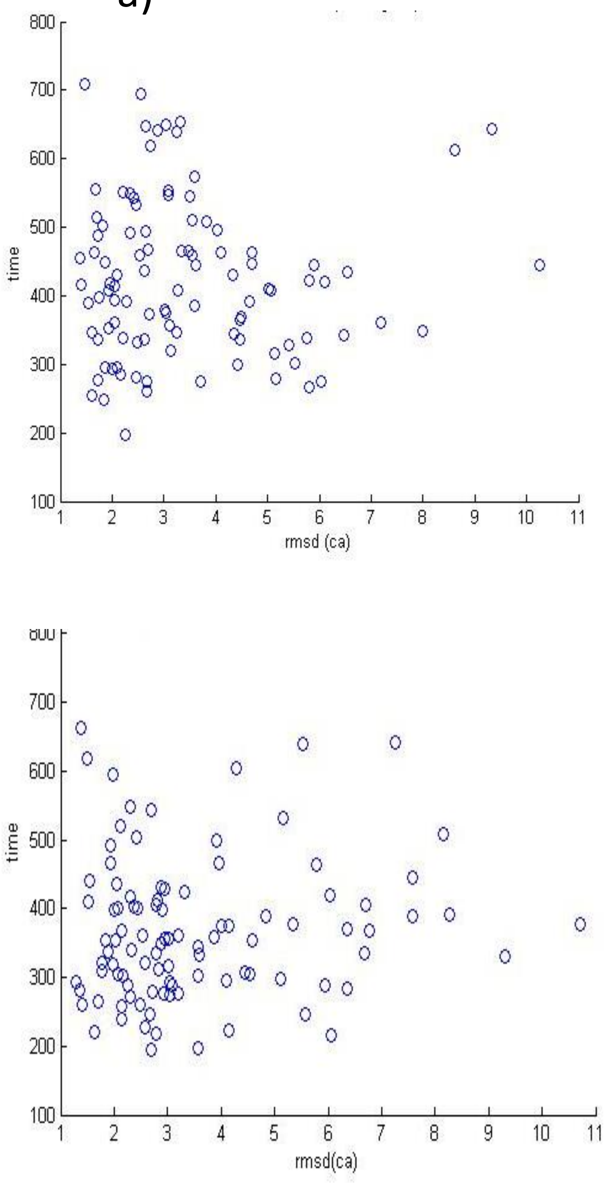
In my annealing procedure the initial temperature T_0 was set to 8 and was updated every 100th MC step. Three cooling schedules were used:

a) an exponential schedule

$$T_k = T_0 a^k, 0 < a < 1 \,, \qquad k - Monte\ Carlo\ step \qquad (3.6)$$

b) a linear schedule

a)



b)

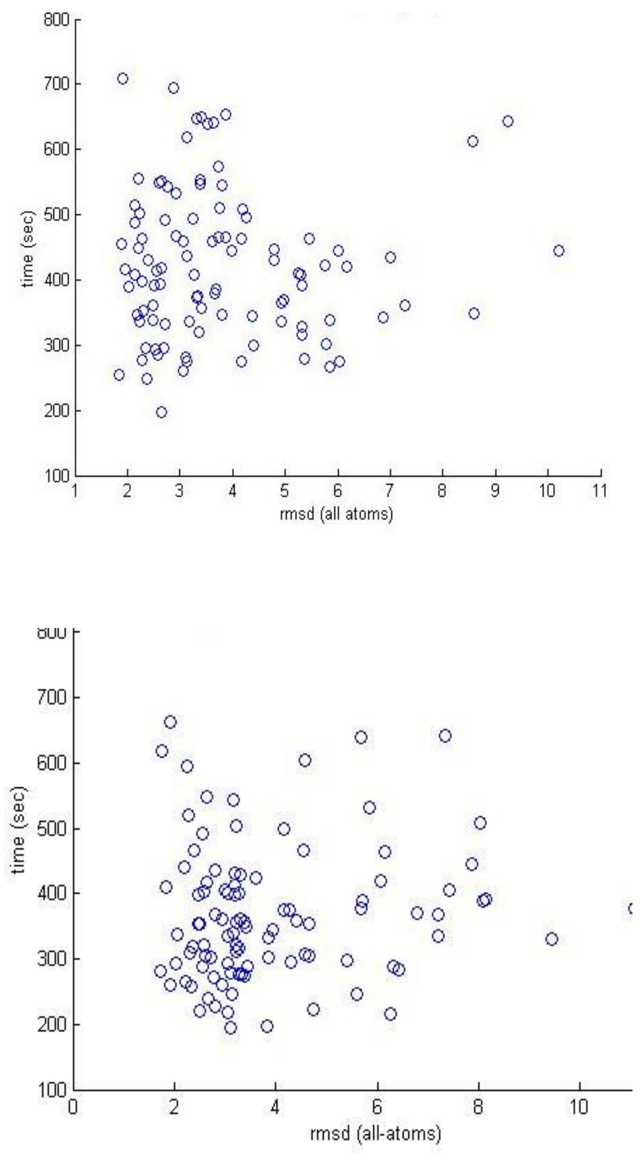
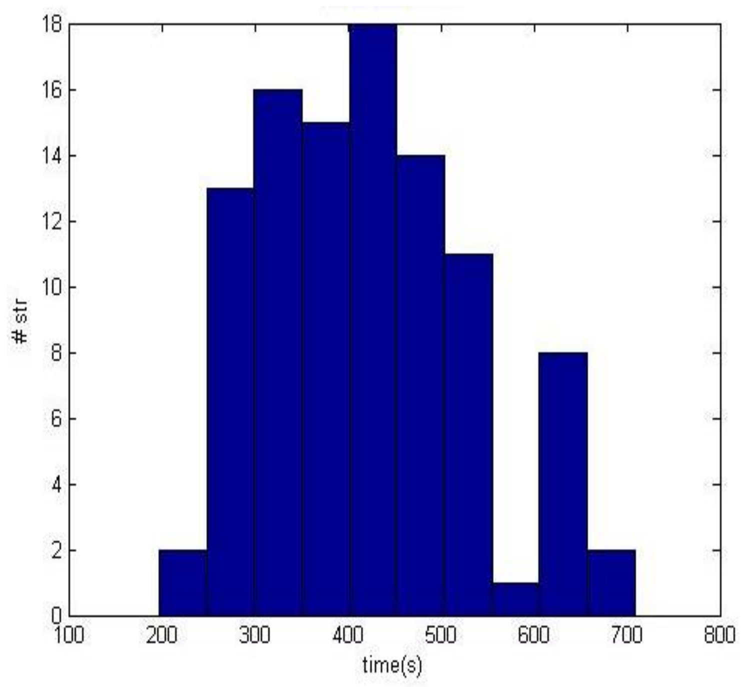


Figure 3.20: Dunbrack's library improve prediction of 1CRN vs Gaussian distributions for side chains torsional angles. a) scatter plots Cα RMSD vs folding time, a top figure – Gaussian distribution for chi angles with 15° standard deviation, a bottom figure – Dunbrack's library for chi angles; **b)** scatter plots all-atom RMSD vs folding time, a top figure – Gaussian distribution for chi angles with 15° standard deviation, a bottom figure – Dunbrack's library for chi angles

a)



b)

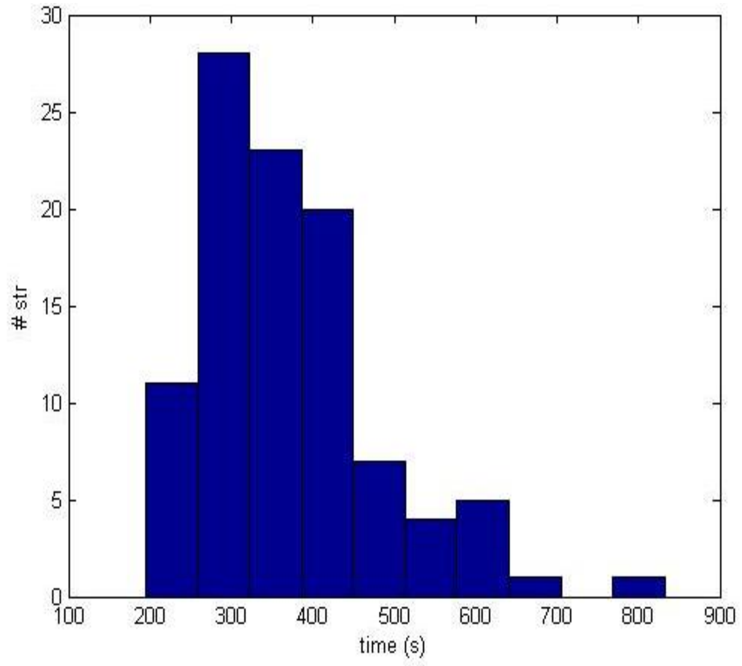


Figure 3.21: Application of Dunbrack's library decrease folding time for 1CRN.
a) Folding time distribution with applied Gaussian distribution for chi angles; **b)** Folding time distribution with applied Dunbrack's library for chi angles.

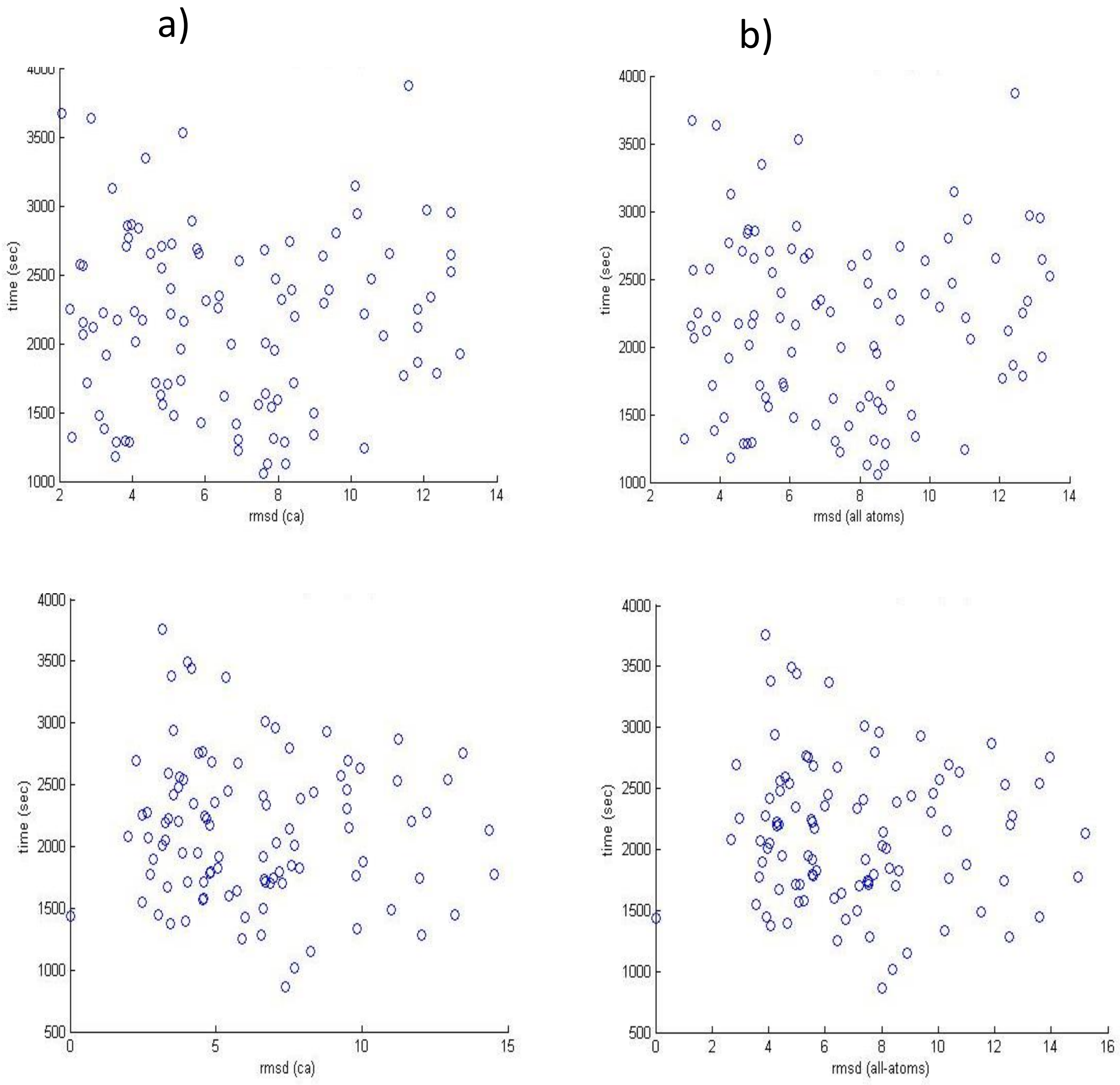


Figure 3.22: Dunbrack's library improve prediction of 1PRB vs Gaussian distributions for side chains torsional angles. a) scatter plots Cα RMSD vs folding time, a top figure – Gaussian distribution for chi angles with 15° standard deviation, a bottom figure – Dunbrack's library for chi angles; **b)** scatter plots all-atom RMSD vs folding time, a top figure – Gaussian distribution for chi angles with 15° standard deviation, a bottom figure – Dunbrack's library for chi angles

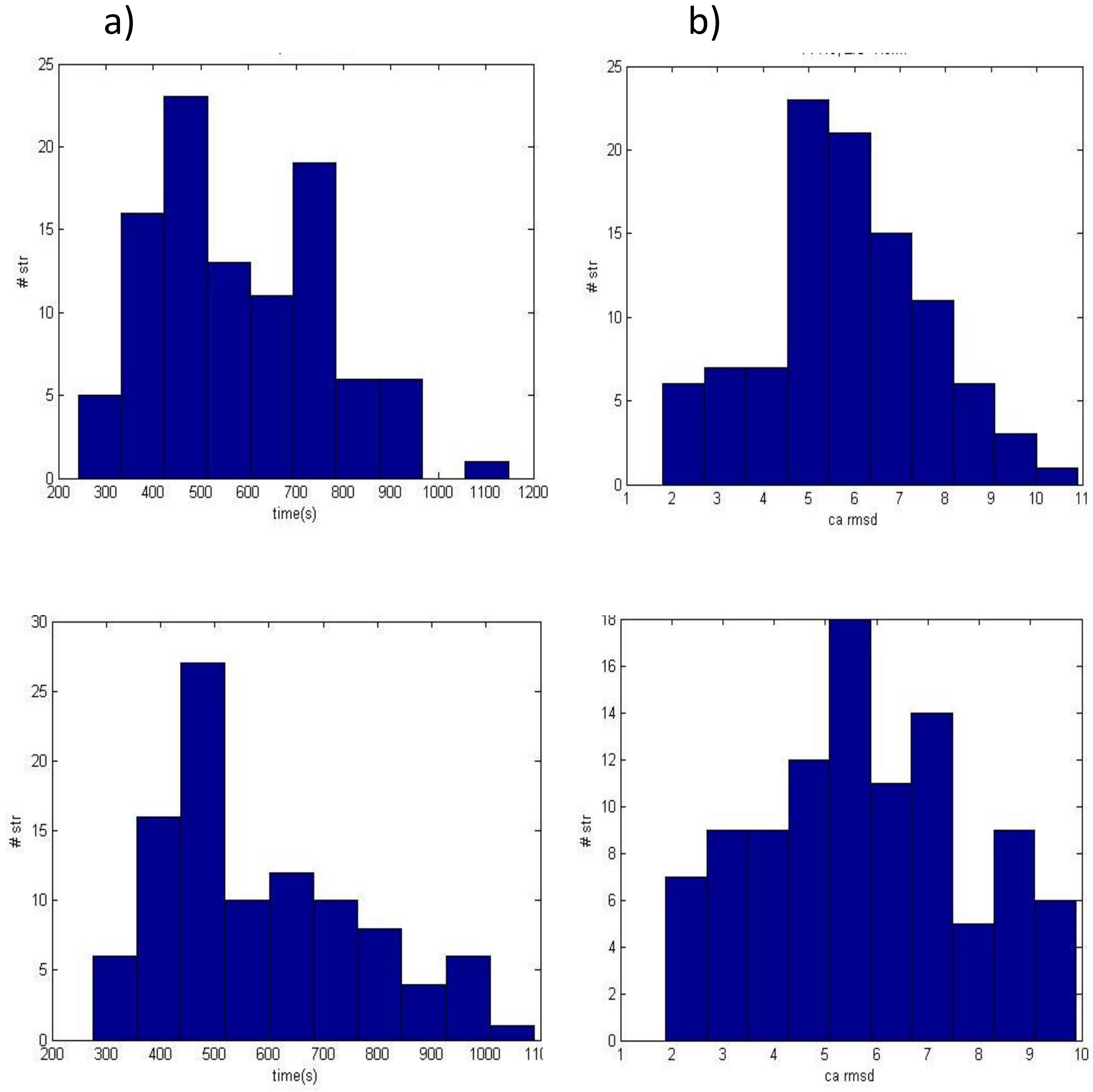


Figure 3.23: Application of Dunbrack's library decrease folding time and increase folded trajectories with RMSD below 4Å for 1YWJ.
a) Folding time distribution with applied Gaussian distribution (top) vs. Dunbrack's library (bottom), for chi angles; **b)** RMSD distribution with applied Gaussian distribution (top) vs. Dunbrack's (bottom) library for chi angles.

$$T_k = T_0 - \eta k, \qquad (3.7)$$

c) a logarithmic schedule

$$T_k = \frac{T_{k-1}}{1 + \frac{T_{k-1}\log(1+\delta)}{3 * smoothed\ protein\ energy\ deviation}}, \delta = 1 \qquad (3.8)$$

I ran 100 trajectories for each protein with the same initial conditions for the statistical validation of the folding results. The only difference between trajectories is a seed value that is used for random number generation (see Random number chapter).

The most successful in terms of balance of folding time and folded trajectories number was a logarithmic schedule (**Equation 3.8**). By changing δ values I speeded up the temperature decrease that should force the sampling algorithm to find a faster folding pathway or to trap trajectories at local minimum (an energetic trap). The results of those experiments, shown in **Figures 3.25 - Figure3.33**, illustrated the latter; the decrease in folding time had a high price of losing folded trajectories with RMSD below 3Å. The optimal for parameter δ was 1.

```
Generated initial conformation X;
Generated initial temperature $T_0$ ;
n = 0;
REPEAT
  n = n + 1;
  m = 0;
  REPEAT
    m = m + 1;
    Y = generate_new_conformation(X, $T_k$);
    IF accept_conformation(X, Y, $T_k$) THEN X=Y
    UNTIL m = M  OR  m = N;
    $T_{k+1}$ = update($T_k$);
    k = k + 1;
UNTIL $T_k$ = 0  OR  Protein  Native Energy is reached
          OR in Protein Energy Local Minimum
                OR n = N
```

Fig 3.24: Pseudo code for simulated annealing algorithm.
N – the total number of Monte Carlo Steps; M-number of Monte Carlo steps after which the temperature is updated; M < N; T_k – current temperature.

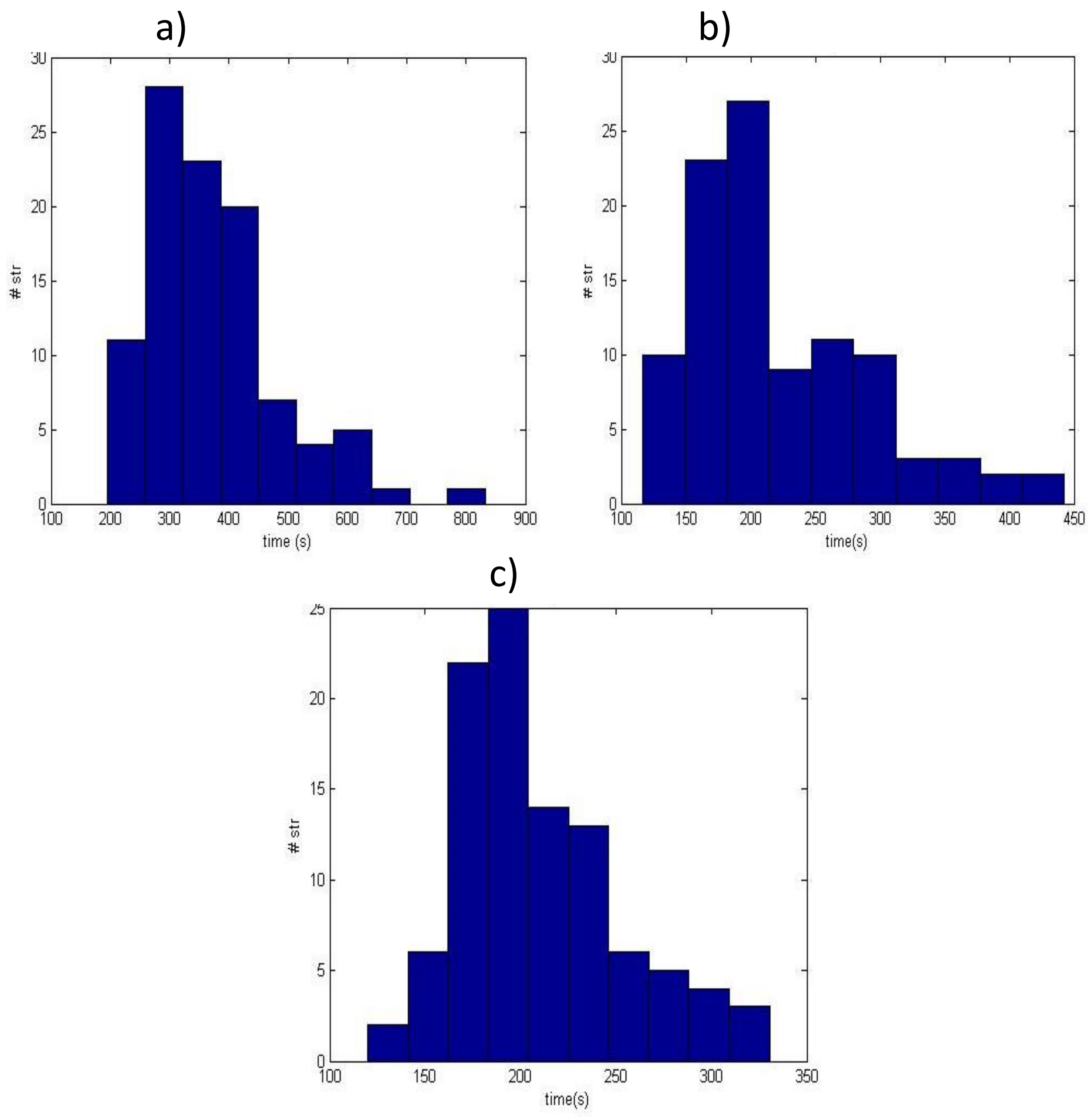


Figure 3.25: Folding speed distribution for 1CRN with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

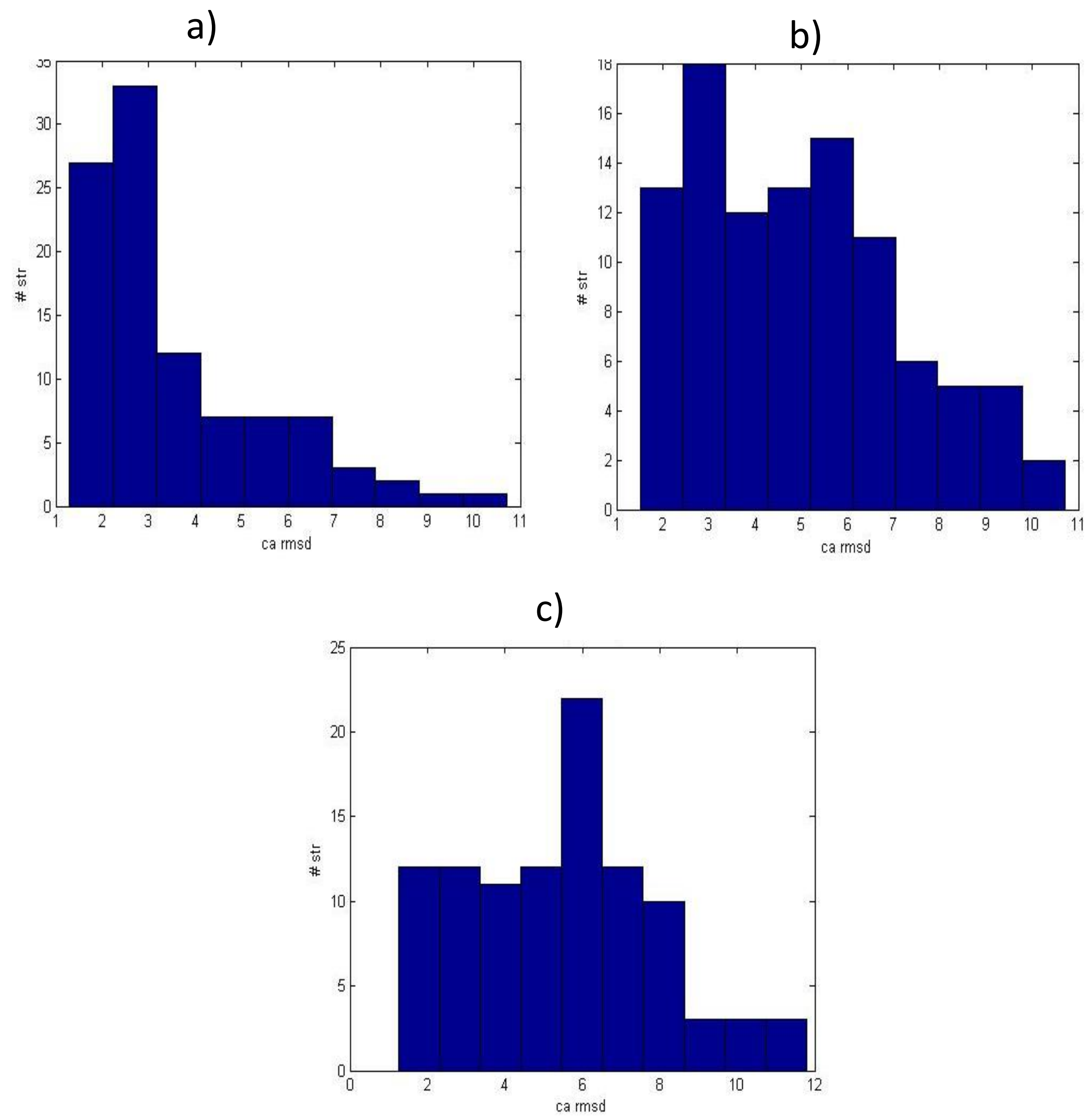
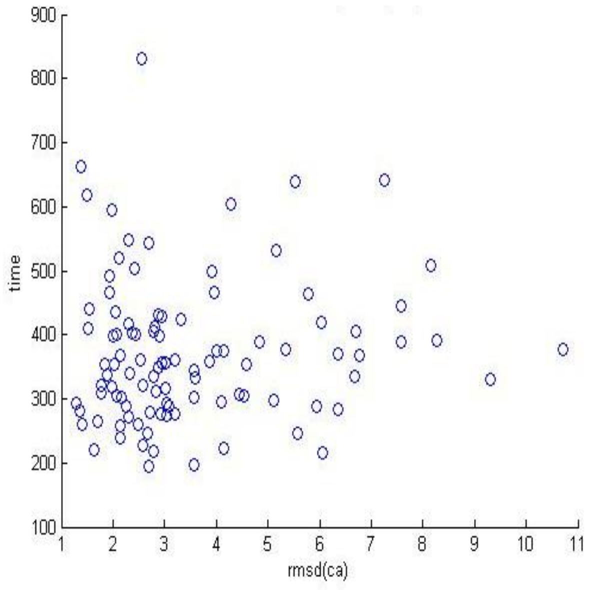
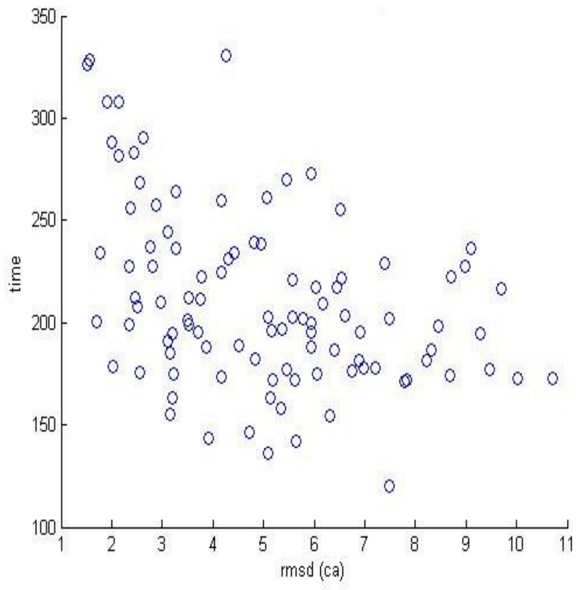


Figure 3.26: Cα RMSD distribution for 1CRN with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

a)



b)



c)

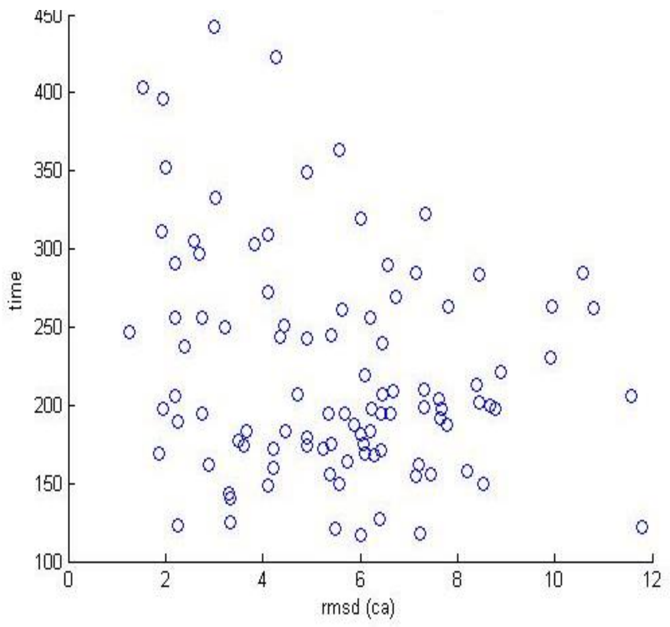


Figure 3.27: Scatter plots Cα RMSD vs, folding speed for 1CRN with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

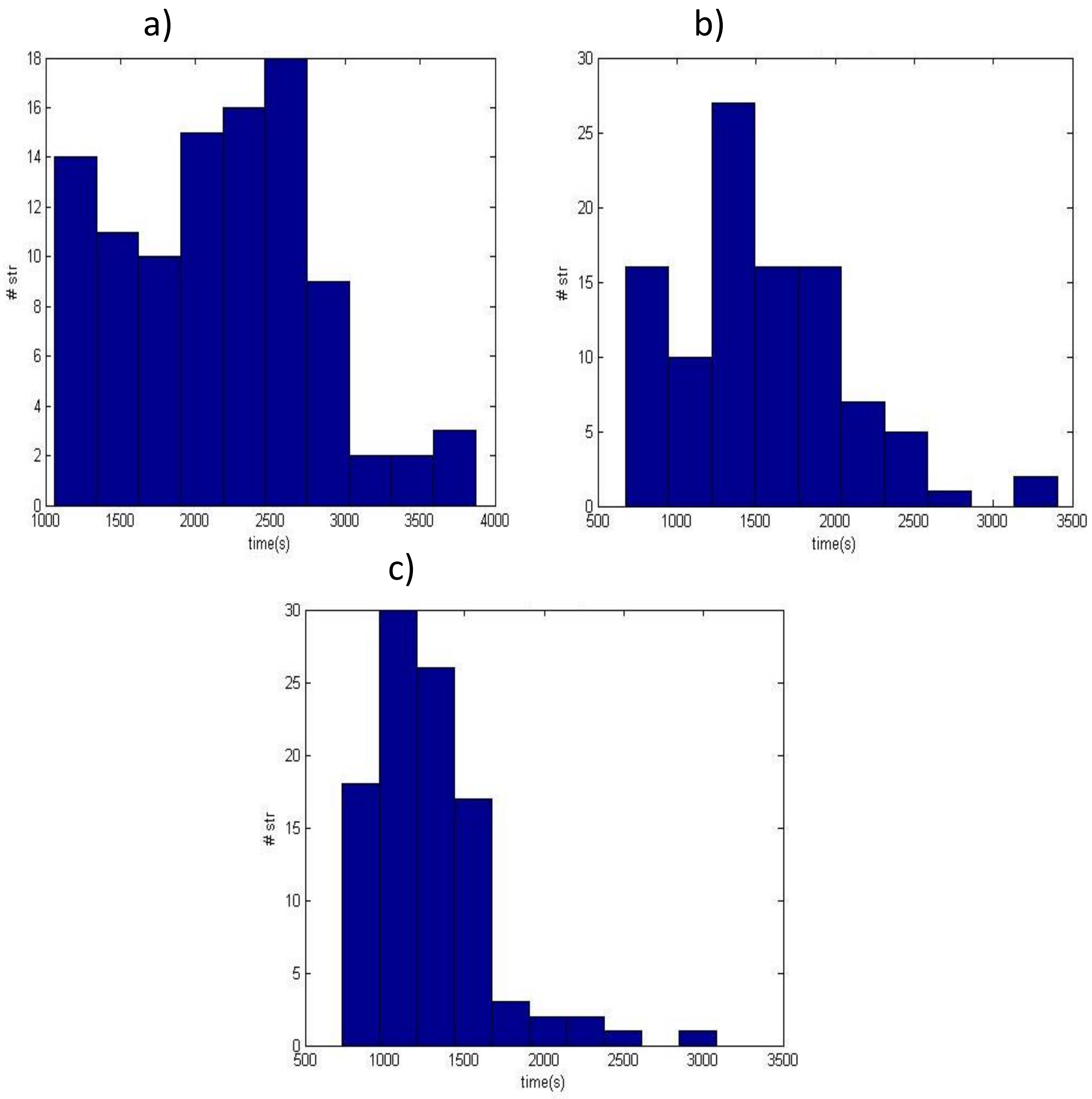


Figure 3.28: Folding speed distribution for 1PRB with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

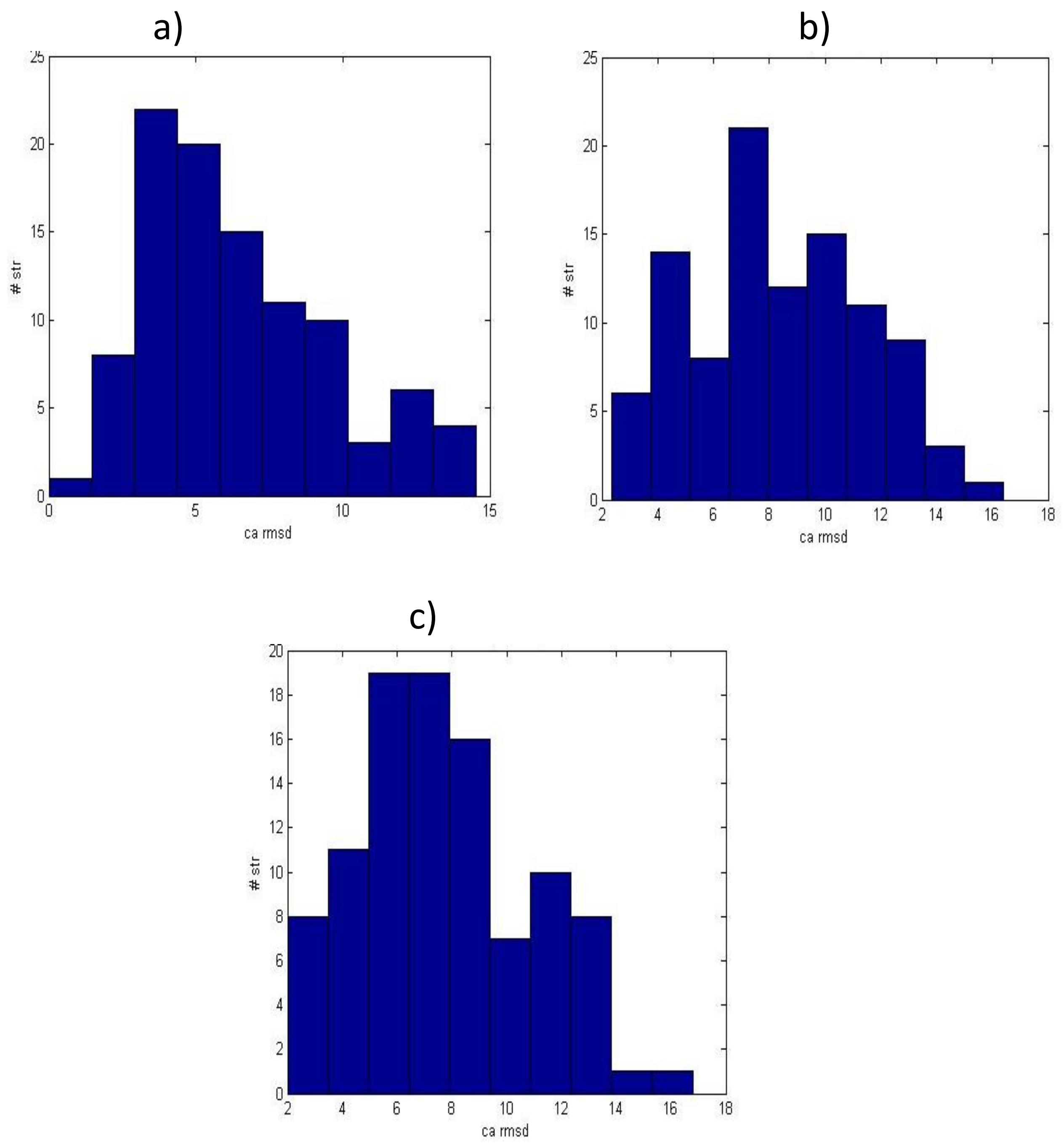


Figure 3.29: Cα RMSD distribution for 1PRB with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

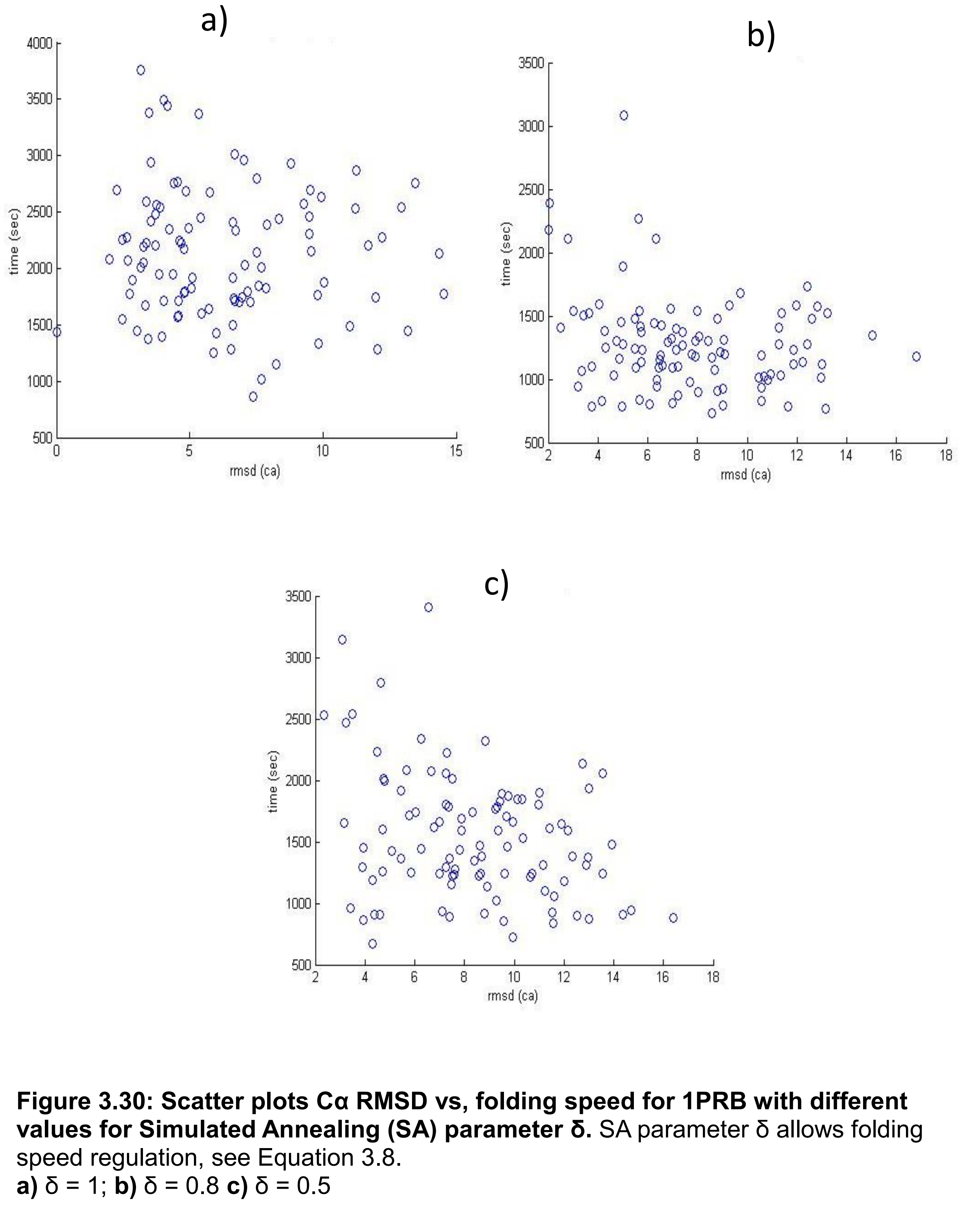


Figure 3.30: Scatter plots Cα RMSD vs, folding speed for 1PRB with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

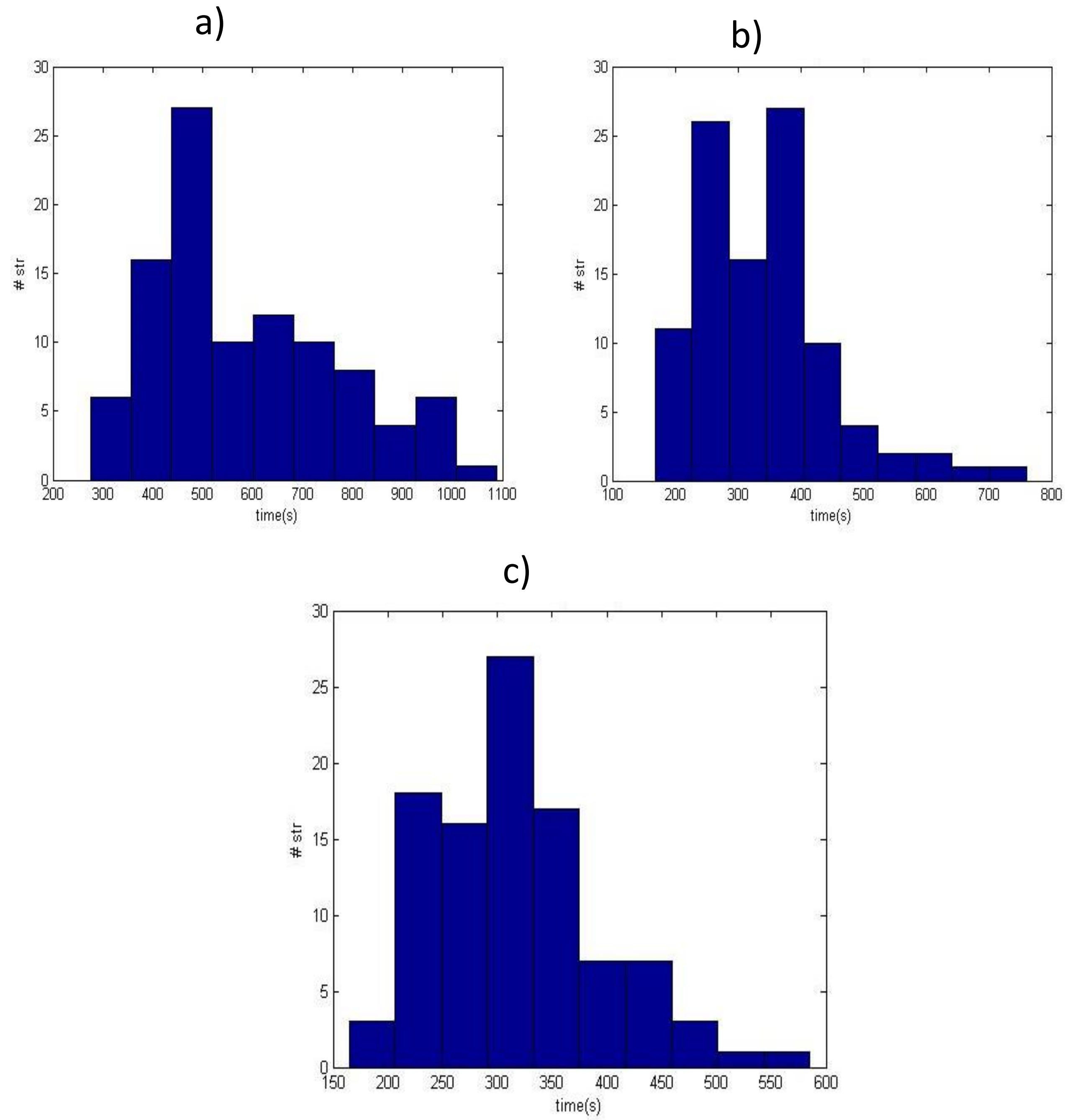


Figure 3.31: Folding speed distribution for 1YWJ with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

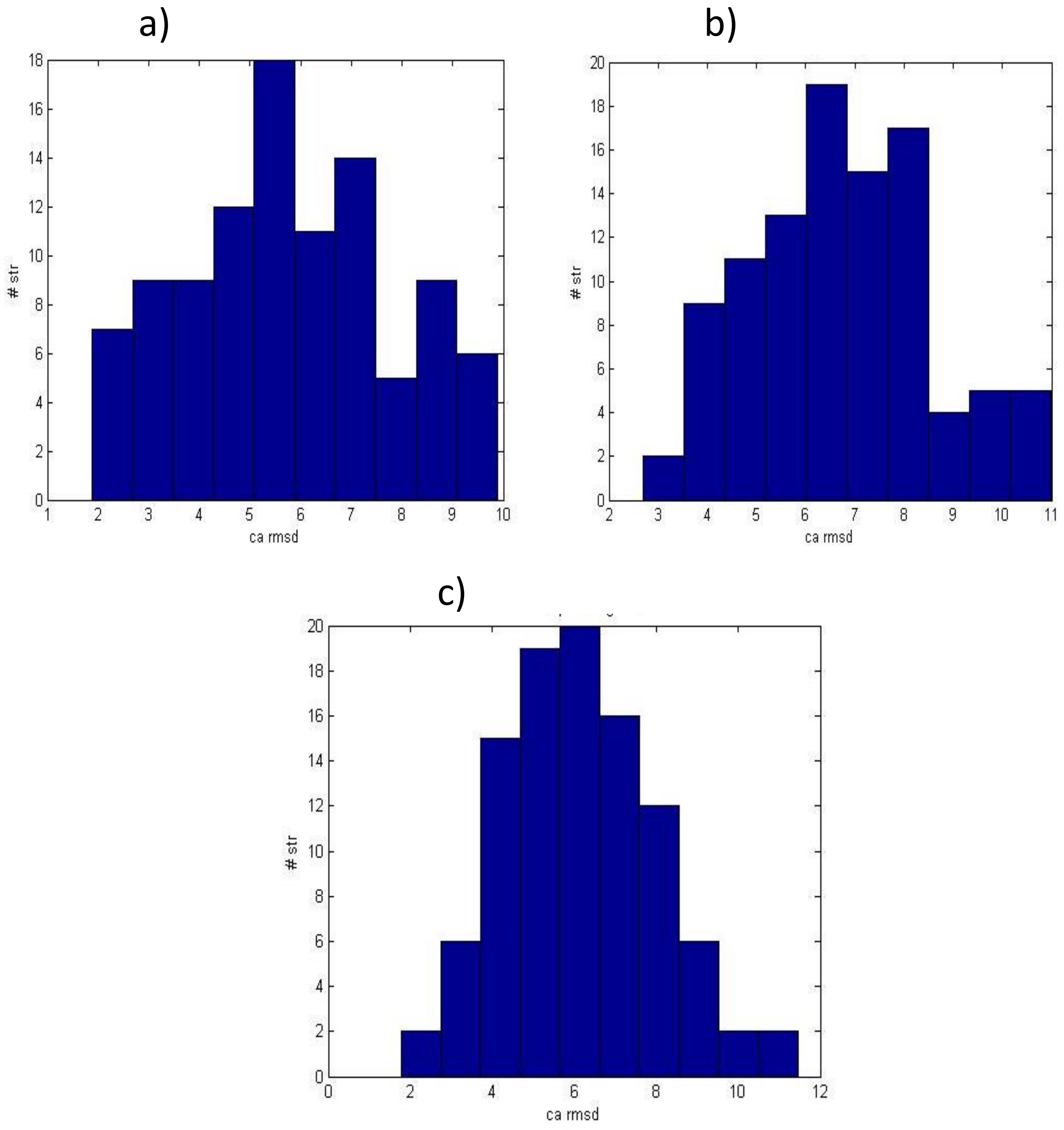


Figure 3.32: Cα RMSD distribution for 1YWJ with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

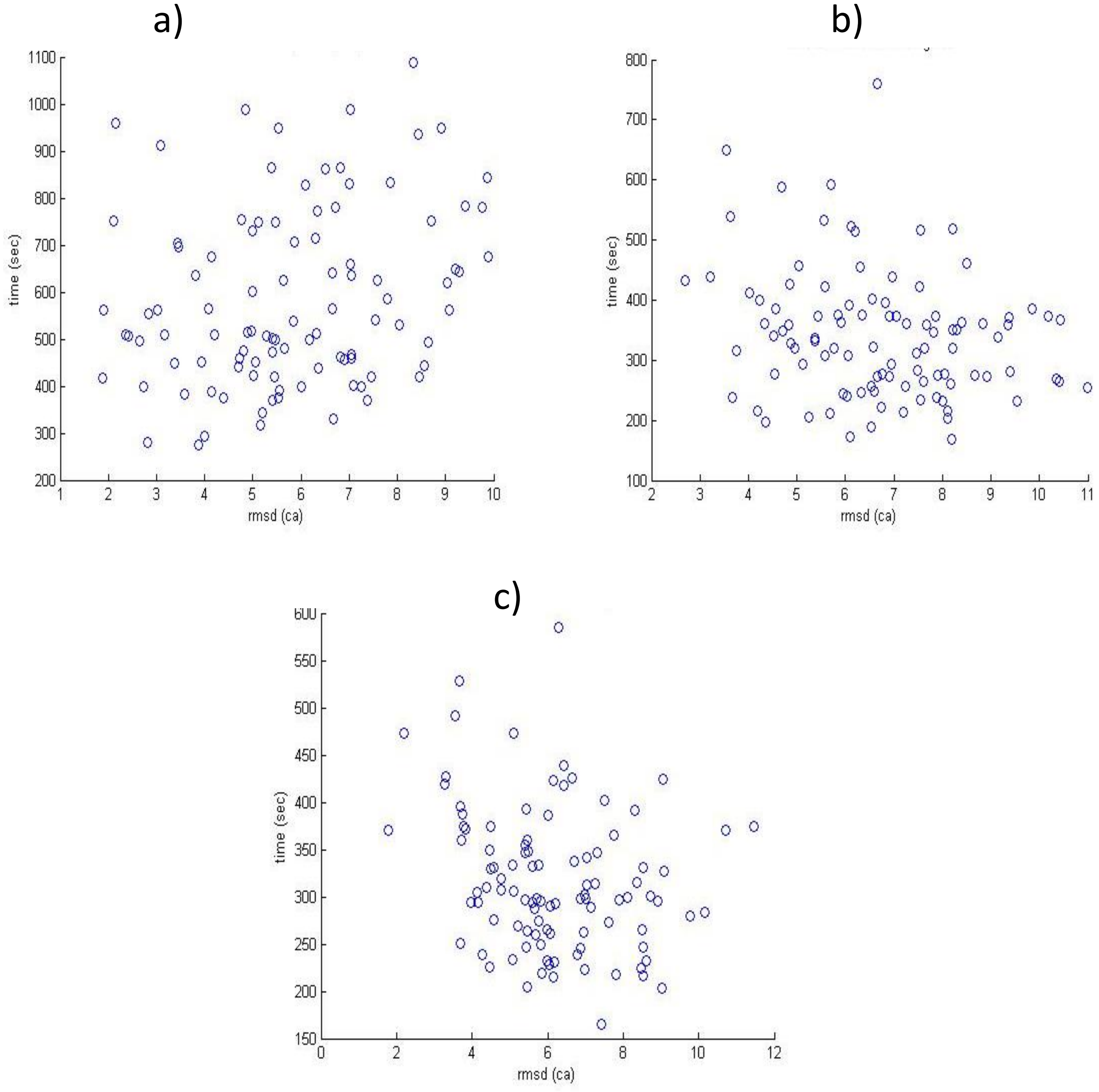


Figure 3.33: Scatter plots Cα RMSD vs, folding speed for 1YWJ with different values for Simulated Annealing (SA) parameter δ. SA parameter δ allows folding speed regulation, see Equation 3.8.
a) δ = 1; **b)** δ = 0.8 **c)** δ = 0.5

3.7 Random numbers generation

A random number generator is an important algorithmic tool for Monte Carlo based simulations. In my algorithm it is used for the generation and the acceptance of protein conformations. At each MC step the sampling algorithm requires at least 6*N random numbers, where N is a protein sequence length. Therefore it is essential to choose a suitable random number generation algorithm for protein structure prediction. A random number generator (RNG) is a computational device that produces a sequence of numbers that cannot be reasonably predicted better than by a random chance. However, any program will produce output that is entirely predictable, hence not truly “random.” Nevertheless, practical computer “random number generators” are in common use. Good random number generators should pass a certain list of statistical tests. The user should also be aware of any limitations of RNGs, so that he or she will be able to judge whether they are relevant to the case at hand.

Mersenne Twister (MT) (C/C++ code is listed in Appendices section) was chosen for the sampling algorithm. MT is a pseudo RNG (PRNG) that satisfies all the requirements to be rated as a good PRNG. It provides fast generation of very high-quality pseudorandom numbers with a long period length of 2^{19937}-1. Since MT uses a machine word length, a programmer should be very careful when employs specific to a programming language/compiler variables that refer to machine architecture. For example, I noticed that the same C++ code can produce different results in two different OS, such as Windows and Linux. **Figure 3.34** illustrates how original 2-dimensional distribution (**Figure 3.34, a)**) was correctly sampled in Windows7 (**Figure 3.34, b)**) and incorrectly sampled in Linux (32-bit) (**Figure 3.34, c)**). The discovery was crucial for the sampling algorithm.

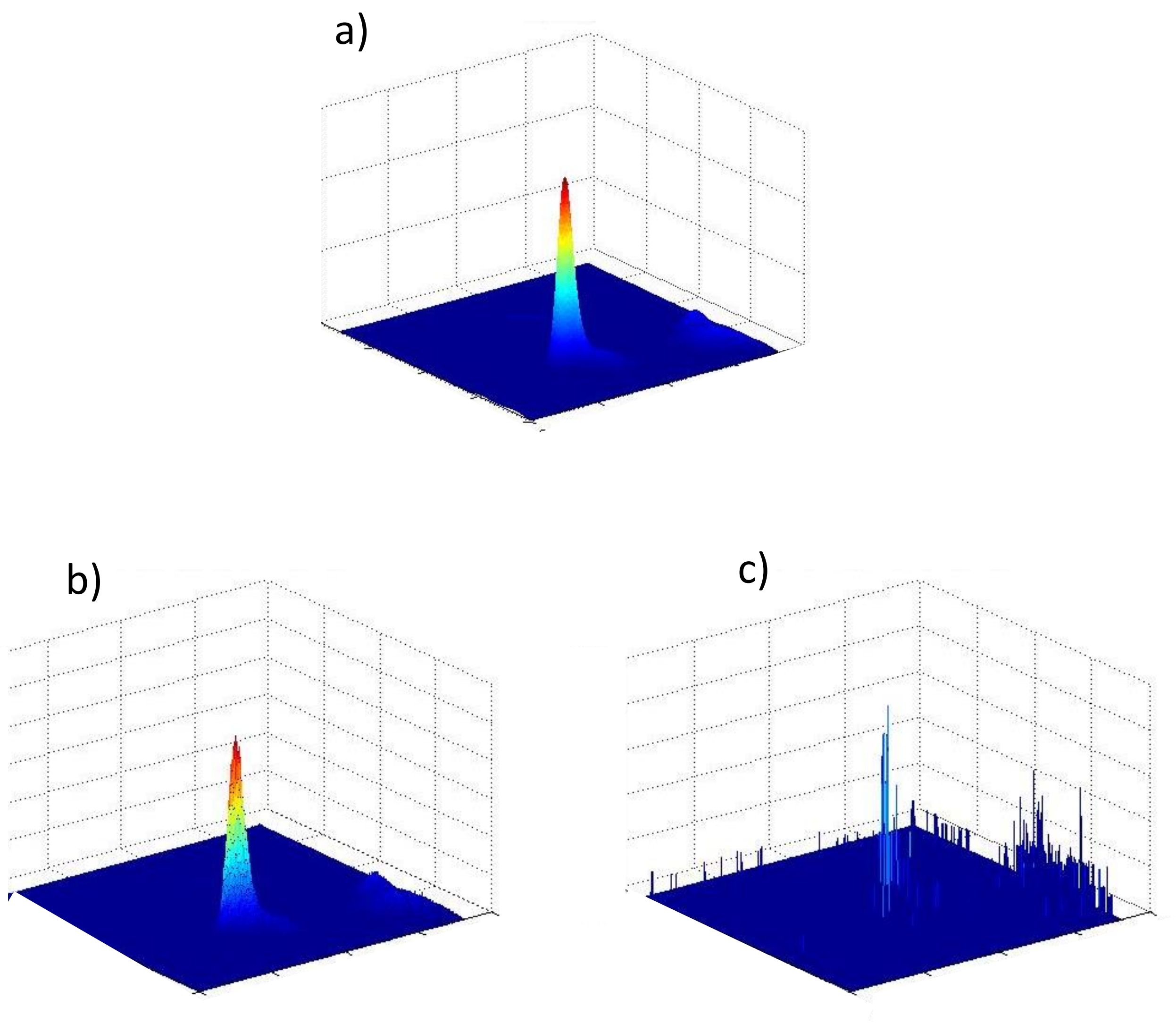


Figure 3.34: The same random number generator algorithm implementation may give dissimilar results in different Operating Systems (OS)
a) original distribution; **b)** sampling of the original distribution in Windows 7; **c)** sampling of the original distribution in Linux

3.8 The protein structure prediction software

The software is built on OOPS, an Open Protein Simulator. OOPS provides a framework where different approaches and algorithms for protein folding and structure prediction can be tested. OOPS is based on the PL and its plugin architecture (21). The overall design of OOPS is illustrated in **Figure 3.35**. The C++ main() functions call plugins to initiates and finalizes the main *ab initio* routine and energy terms functions located in separate plugins. The module that contains the design of a molecule and functions required for molecule manipulations such as a rotation of torsional angles, property of atoms and amino acids and etc. is located in PL library which is an independent entity of the software.

There are main (but not limited to) additions and changes that has been introduced to the original version of OOPS:

1) Optimized rotation procedure for a backbone and side chains.

2) Corrected side chains definitions.

3) A new Monte Carlo sampling algorithm and related to it functions.

4) A new version of Mersenne Twister, a random number generator.

5) Introduction of torsional angles sampling distributions to the algorithm.

6) Energy function design.

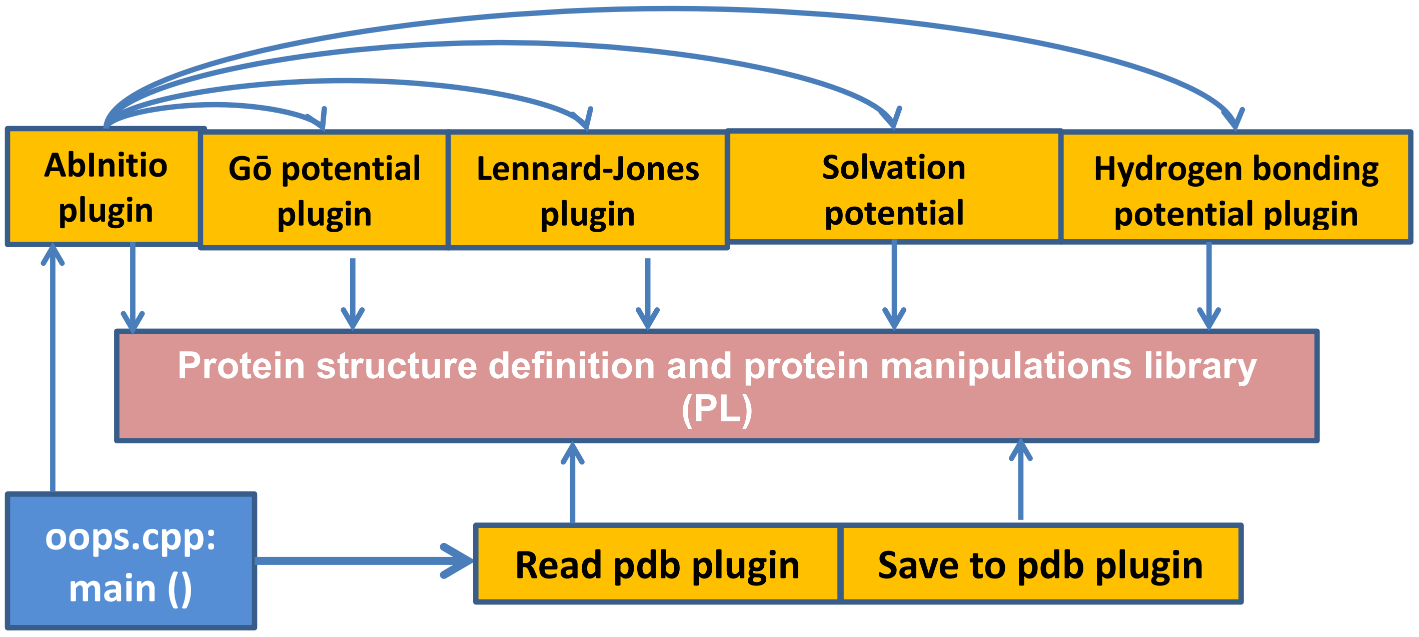


Figure 3.35: OOPS software design

Chapter 4

Energy Optimization

4.1 The optimization of energy weights

In the optimization algorithm an energy function was a weighted sum of energy terms I chose to test, the Lennard-Jones potential, the Lazaridis-Karplus solvation potential and hydrogen bonding potential, and the Gō potential, for which the sampling algorithm was tuned to maximize the number of folded trajectories and to minimize protein folding time:

$$E = w_0 E_{G\bar{o}} + w_1 E_1 + w_2 E_2 + \cdots + w_N E_N, \sum_{i=0}^{N} w_i = 1 \qquad (4.1)$$

The optimization process started with the Gō weight $w_0 = 1$ and zero weights for other energy terms. At the next iteration step I used the Dirichlet distribution [42] with $\alpha = 1$ to generate random $N + 1$ weights that summates to 1 and $w_0 < 1$. I calculated how many protein trajectories out of 100 were folded for each protein. A protein trajectory was considered to be folded if the backbone RMSD of a final structure was equals or less 3 Å. If the number of folded trajectories exceeded 3, I saved the current Gō weight as a reference for the next step and generated a new random combination of weights with $w_0 < w_0^{prev}$. If the number of folded trajectories were less than 3, I generated a new set of weights. The optimization process was stopped after a pre-defined number of steps M = 100, or the Gō weight became zero. I also incorporated the tuning procedure for the reduced Gō potential weight w_0. For the fixed value of w_0 I generated L= 10 sets of $(w_1, \dots, w_N)$, $\sum_{i=0}^{N} w_i = 1$, and chose the combinations of weights that gave the bigger number of folded trajectories for all proteins (**Figure 4.1**).

4.2 Finding a fast folding pathway with the Gō Potential

To outline the energy funnel I used our MMC routine with the Gō potential as a scoring function $E = E_{G\bar{o}}$. For each of the tested proteins I ran 100 trajectories to assess how many of them were folded. I considered a protein to be folded if the backbone RMSD was within 3 Å to the native state. It should be mentioned that the criterion alone might not distinguish correctly formed secondary structures due to the nature of a Gō potential (85). Simulations were started from a fully extended protein conformation with backbone dihedral angles at 180º. Side chain dihedral angles were chosen to minimize the number of clashes with a backbone and neighboring side chains. The average numbers of folded trajectories over 5 runs folding were 35 for 1crn trajectories, 21 for 1prb trajectories and 6 ywj trajectories (**Table 4.1**). Scatter plots of RMSD *vs*. folding time for a representative run are presented in **Figure 3.4**. Successful trajectories for 1crn and 1prb were folded within 6K Monte Carlo steps, for 1ywj successful folding was reached within 4K Monte Carlo steps. Energy landscapes for representative trajectories are shown in **Figure 4.2**.

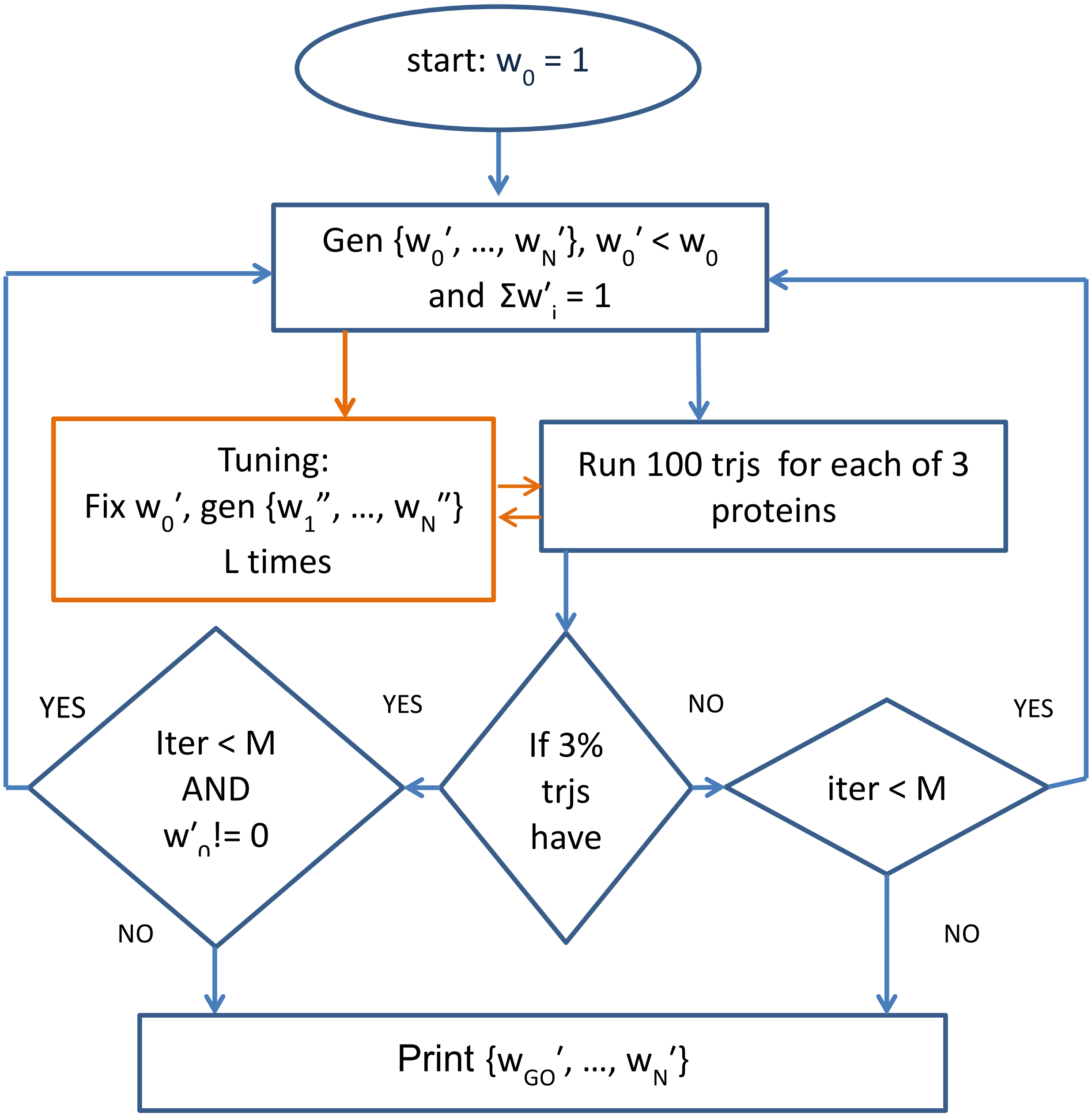

start: $w_0 = 1$
Gen $\{w_0', \ldots, w_N'\}$, $w_0' < w_0$ and $\Sigma w'_i = 1$
Tuning: Fix w_0', gen $\{w_1'', \ldots, w_N''\}$ L times
Run 100 trjs for each of 3 proteins
YES
Iter < M AND $w'_0 != 0$
YES
If 3% trjs have
NO
iter < M
YES
NO
NO
Print $\{w_{GO}', \ldots, w_N'\}$

Figure 4.1. The optimization of energy weights flow chart.
The optimization process starts with Gō weight equals to 1and zero weights for other energy terms. At the next iteration step I generate random weights that summate to 1 and Gō weight less than 1, calculate how many protein trajectories out of 100 are folded for each protein. If the number of folded trajectories exceeds 3, I save the current Gō weight as a reference for the next step and generate a new random combination of weights with Gō weight less than the previous one. If the number of successful trajectories less than 3, I generate a new set of weights. The optimization process stops after defined number of steps or the weight of Gō becomes zero.

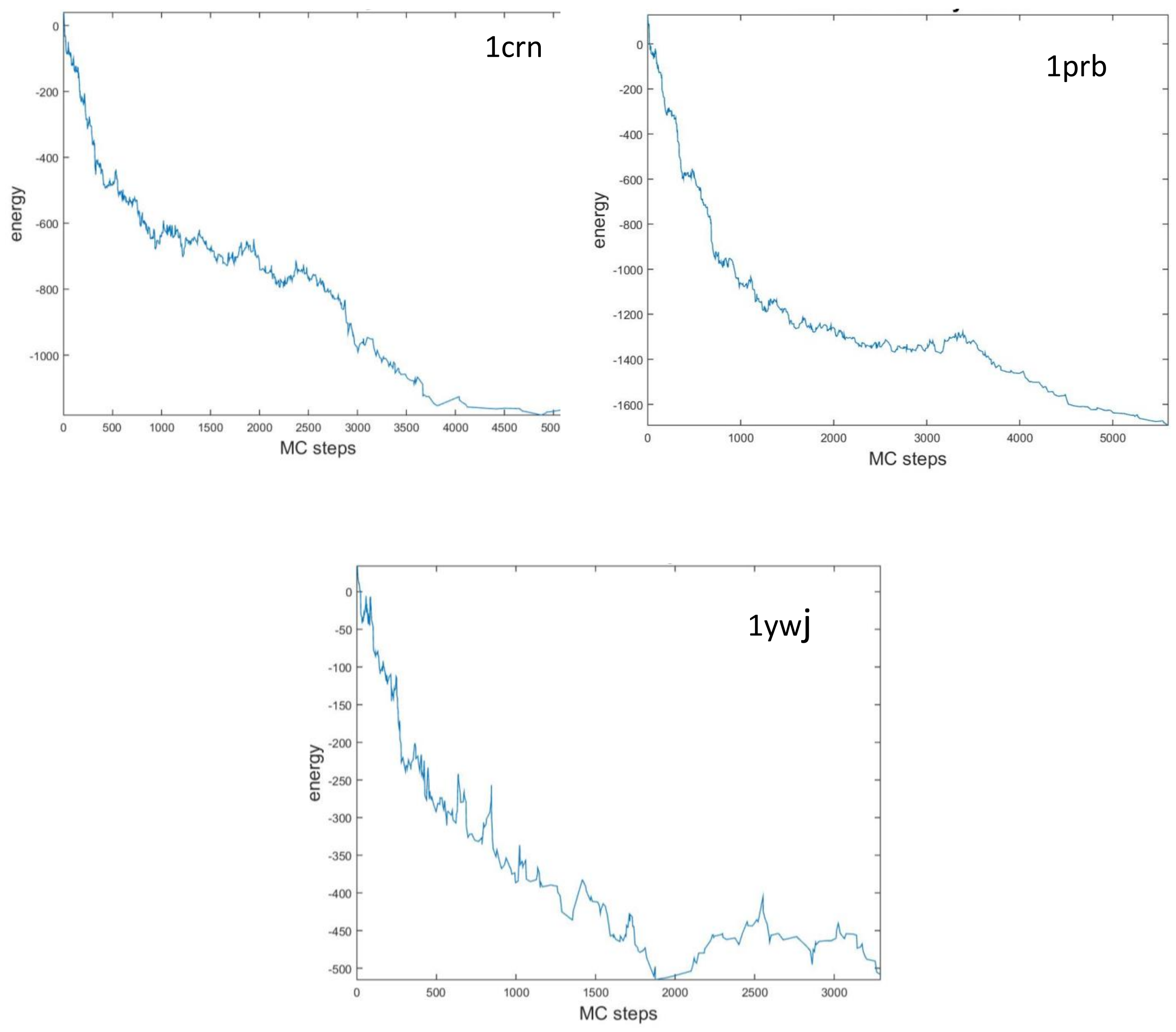


Figure 4.2. Energy landscapes for representative protein folding trajectories folded with the Gō potential only. Results for 1crn, 1prb, 1ywj

4.3 Addition of a single energy term to the Gō potential allows reduction of its weight

To explore effects of additional energy terms on protein folding and the Gō potential weight reduction, the Lennard-Jones (LJ) potential **(Equation 4.3**), the Lazaridis-Karplus (LK) solvation potential (**Equations 4.4-4.5**),, and hydrogen bonding (HB) potentials (Equation 4.6), were used in the following scoring function:

$$E = w_{G\bar{o}}\, E_{G\bar{o}} + w_{add}E_{add}, \qquad w_{G\bar{o}} + w_{add} = 1 \qquad (4.2)$$

where $E_{G\bar{o}}$ is the Gō potential, E_{add} is one of three potentials chosen for the test, $w_{G\bar{o}}$ and w_{add} are corresponding energy weights. Scaling factors were applied to each added energy term. I started simulations with $w_{G\bar{o}} = 0.95$ and continued to reduce the values of $w_{G\bar{o}}$ by 0.01 for each next run of simulations till $w_{G\bar{o}} = 0.8$ or the number of successful trajectories was significantly decreased.

4.3.1 Lennard Jones Potential

Van der Waals interactions were modeled with 6-12 Lennard-Jones potential (23), where a linear function was used in a repulsion mode. This potential was utilized in early versions of Rosetta algorithm for the final refinement stage (12, 78), where physically realistic, atomic- level potentials were required for a better presentation of the primary contributions to stability and structural specificity.

$$\boldsymbol{E_{VDW}} = \sum_{i}\sum_{j>i}\begin{cases}\left(\left(\frac{\boldsymbol{r_{ij}}}{\boldsymbol{d_{ij}}}\right)^{\boldsymbol{12}} - \boldsymbol{2}\left(\frac{\boldsymbol{r_{ij}}}{\boldsymbol{d_{ij}}}\right)^{\boldsymbol{6}}\right)\boldsymbol{e_{ij}}, & \frac{\boldsymbol{d_{ij}}}{\boldsymbol{r_{ij}}} > \boldsymbol{0.6} \\ \left[-\boldsymbol{8759}\left(\frac{\boldsymbol{d_{ij}}}{\boldsymbol{r_{ij}}}\right) + \boldsymbol{5672.0}\right]\boldsymbol{e_{ij}}, & \boldsymbol{else}\end{cases} \qquad (4.3)$$

where r_{ij} is a sum of VDW radii of atoms i and j, d_{ij} is a distance between two atoms i and j, e_{ij} is a depth of VDW well which is calculated as a square root of energy CHARMM parameters [19] for Van der Waals interactions e_i and e_j**.**

4.3.2 Solvation Energy

Solvation effects are presented by the model of Lazaridis and Karplus (50), a Gaussian solvent-exclusion model. It based on theoretical considerations and parametrized with experimental data. The Lazaridis-Karplus solvation energy (EEF1) is used in CHARMM 19 and ROSETTA.

$$\boldsymbol{E_{solv}} = \sum_{i} \boldsymbol{\Delta G_i^{ref}} - \sum_{i>j} \left\{ \frac{\boldsymbol{2\Delta G_i^{free}}}{\boldsymbol{4\pi\sqrt{\pi}\lambda_i r_{ij}^2}} \boldsymbol{exp\left(-x_{ij}^2\right)V_j} + \frac{\boldsymbol{2\Delta G_j^{free}}}{\boldsymbol{4\pi\sqrt{\pi}\lambda_j r_{ij}^2}} \boldsymbol{exp\left(-x_{ji}^2\right)V_i} \right\} \qquad (4.4)$$

where $\boldsymbol{\Delta G_i^{free}}$ and $\boldsymbol{\Delta G_i^{ref}}$ are pre-calculated the solvation free energy and the reference solvation free energy of for an atom type i, r_{ij} is the distance between atoms type i and j atoms, $\boldsymbol{\lambda_i}$ is a correlation length for atom type i, V_i is the volume of atom type i, R_i is the Van der Waals radius of atom i and

$$\boldsymbol{x_{ij}} = \frac{\boldsymbol{r_{ij} - R_i}}{\boldsymbol{\lambda_i}} \qquad (4.5)$$

4.3.3 Hydrogen Bonding

I applied an orientation-dependent hydrogen bonding potential in this study (49). This potential is a secondary structure- and orientation-dependent potential derived from the analysis of hydrogen bond geometries in high-resolution protein structures. In our simulations I used only backbone-backbone hydrogen bonding potential since side chain-side chain and backbone-side chain hydrogen bonding potentials would require different weights (6, 7, 49, 77, 86). The hydrogen bond energy is a linear combination of the four terms:

$$\boldsymbol{E_{HB}} = \boldsymbol{E}(\boldsymbol{\delta_{HA}}) + \boldsymbol{E}(\boldsymbol{\Theta}) + \boldsymbol{E}(\boldsymbol{\Psi}) + \boldsymbol{E}(\boldsymbol{X}) \qquad (4.6)$$

where $\boldsymbol{E}(\boldsymbol{\delta_{HA}})$ depends on the distance between hydrogen and acceptor atoms, $\boldsymbol{E}(\boldsymbol{\Theta})$ depends on the angle at the hydrogen, $\boldsymbol{E}(\boldsymbol{\Psi})$ depends on the angle at the acceptor atom and $\boldsymbol{E}(\boldsymbol{X})$ depends on the dihedral angle in the hydrogen bonds involving in sp^2 hybridized acceptor.

4.3.4 Results

The analysis of simulations for weights pairs $(w_{G\bar{o}}, w_{LJ})$ showed that the increase of the LJ potential weight reduced the average number of folded trajectories for $w_{G\bar{o}} < 0.95$ without any significant change in folding time, despite the fact that more calculations for the scoring function were required (**Figure 4.3**-**Figure 4.5**). The inclusion of the LJ was the most beneficial for 1crn, 1prb. I observed some increase in the average number of successful trajectories for $w_{G\bar{o}} = 0.95, w_{LJ} = 0.05$ for those proteins. The increase of HB potential weight dropped the average number of folded trajectories with no significant change in folding speed for the examined proteins (**Figure 4.6** - **Figure 4.8**). The most effective energy potential was the LK potential in terms of the increased average number of folded trajectories. For the weights $w_{G\bar{o}} = 0.8$, $w_{LK} = 0.2$ the average number of folded trajectories was comparable to that when the Gō potential only was applied for the tested proteins (**Figure 4.9**-**Figure 4.11**). At the same time, folding time for the proteins was increased.

4.4 Adding all three energy terms to the Gō potential

I composed a scoring energy function to test the influence of three additional energy terms on protein folding as following:

$$E = w_{G\bar{o}}E_{G\bar{o}} + w_{LJ}E_{LJ} + w_{LK}E_{LK} + w_{HB}E_{HB}, \qquad w_{G\bar{o}} + w_{LJ} + w_{LK} + w_{HB} = 1 \qquad (12)$$

To minimize the burden of calculations at this stage, I chose weights for energy terms from a 4D-grid $[\min w_{G\bar{o}}, \max w_{G\bar{o}})$ X$[\min w_{LJ}, \max w_{LJ}]X[\min w_{LK}, \max w_{LK}]X[\min w_{HB}, \max w_{HB}]$ with the extreme values $\min w_{G\bar{o}} = 0.8$ and $\max w_{G\bar{o}} = 1$, $\min w_{LJ\backslash LK\backslash HB} = 0$ and $\max w_{LJ\backslash HB\backslash LK} = 0.2$. A grid step for each dimension was set to 0.05. For each quadruplet $(w_{G\bar{o}}, w_{LJ}, w_{LK}, w_{HB})$ I ran five trials to simultaneously fold three proteins to calculate the average percentage of folded trajectories. **Table 4.1** lists the average number of folded trajectories for each protein, where at least two weights are not equal to 0. Four combinations of weights, (0.8, 0, 0.05, 0.15), (0.8, 0.05, 0.05, 0.1), (0.8, 0.05, 0, 0.15), (0.8, 0.1, 0, 0.1) gave the best performance with at least 3% of folded trajectories for each protein.

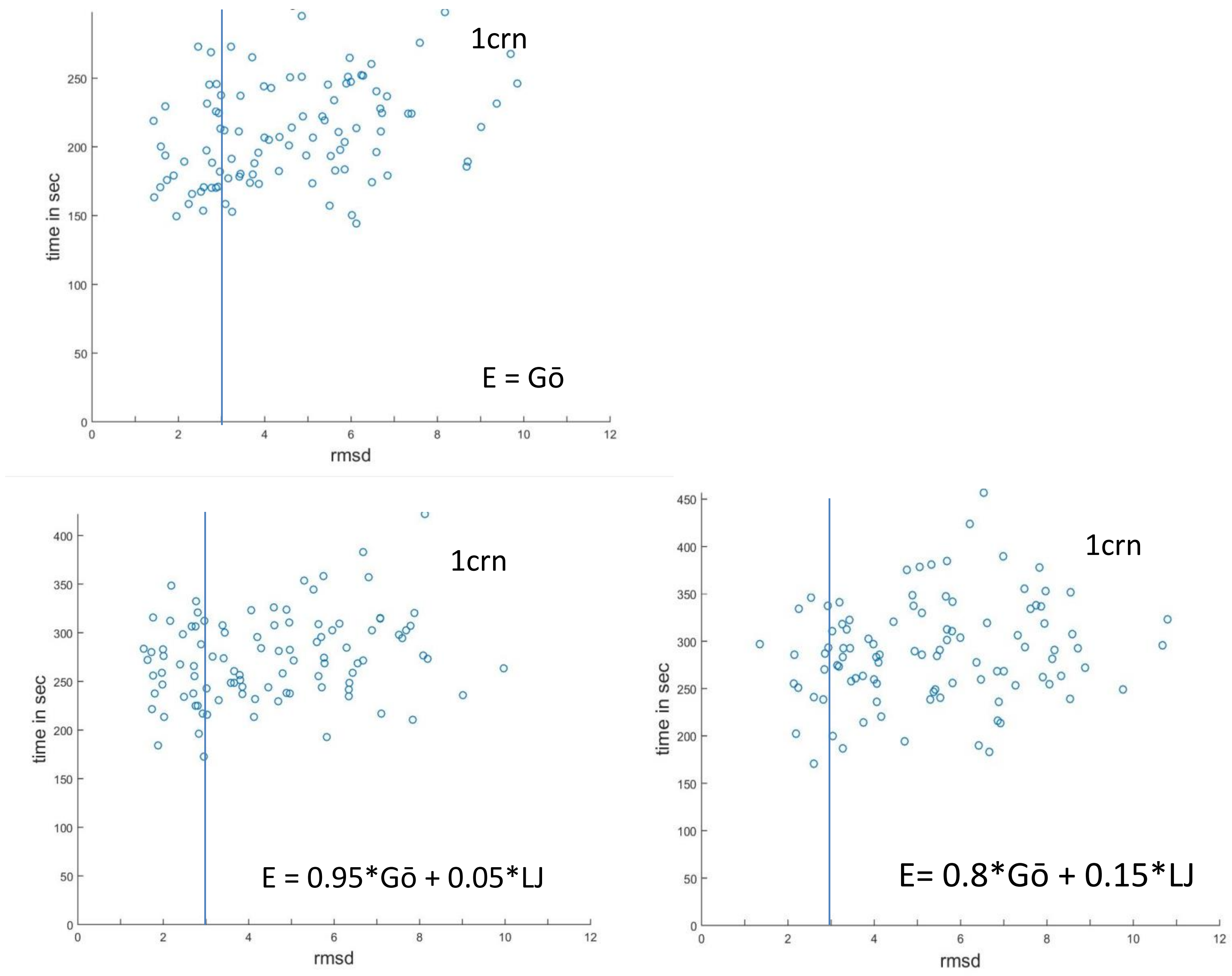


Figure 4.3. Scatter plots RMSD vs folding time for 1crn trajectories folded with E= $w_{G\bar{o}}$\*Gō+w_{LJ}*LJ potential. The average numbers of folded trajectories with rmsd ≤ 3 Å decreased with no significant change in folding time with the LJ weight increase.

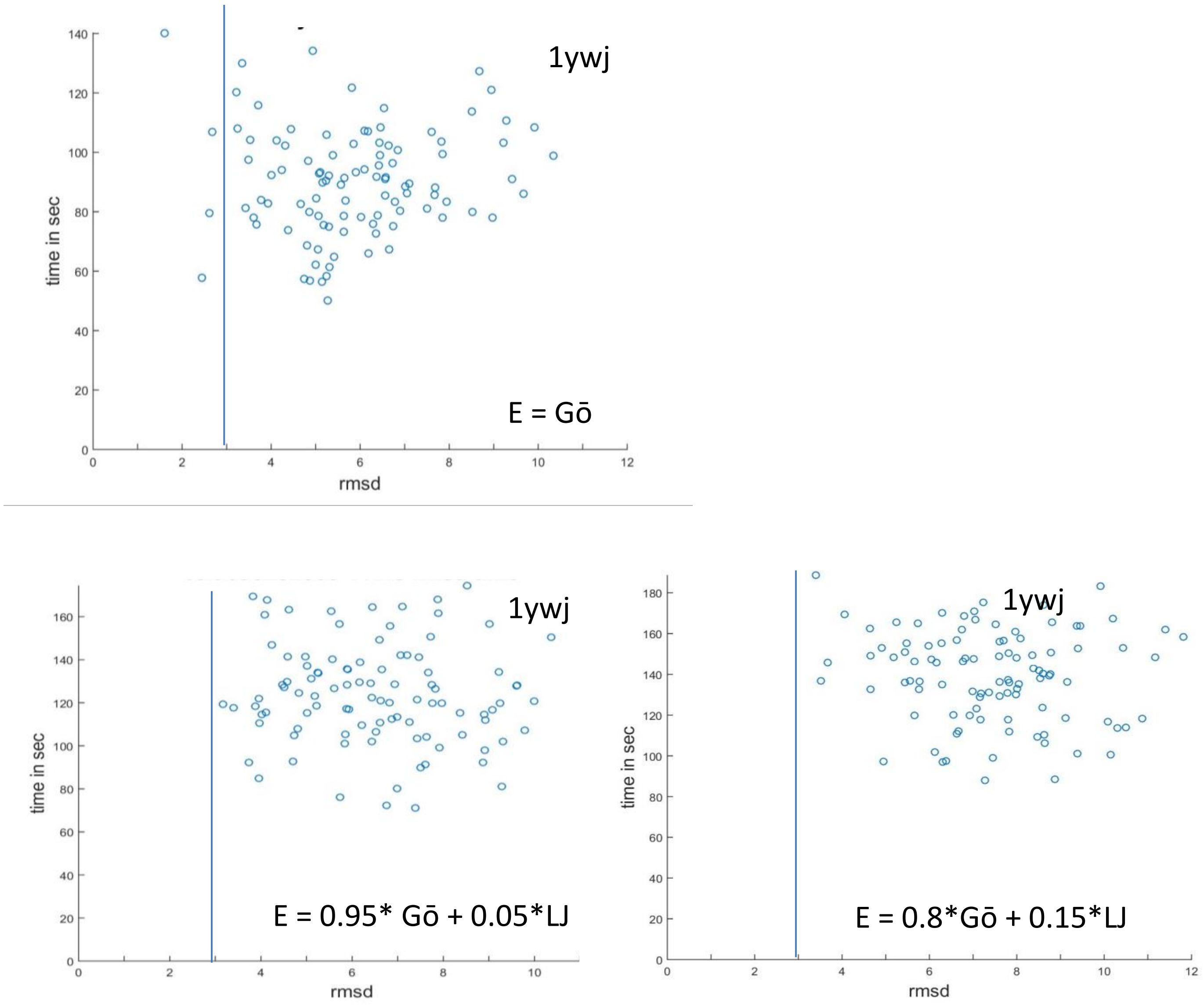


Figure 4.4. Scatter plots RMSD vs folding time for 1ywj trajectories folded with E= $w_{G\bar{o}}$\*Gō+w_{LJ}*LJ potential. The average numbers of folded trajectories with rmsd ≤ 3 Å decreased with no significant change in folding time with the LJ weight increase.

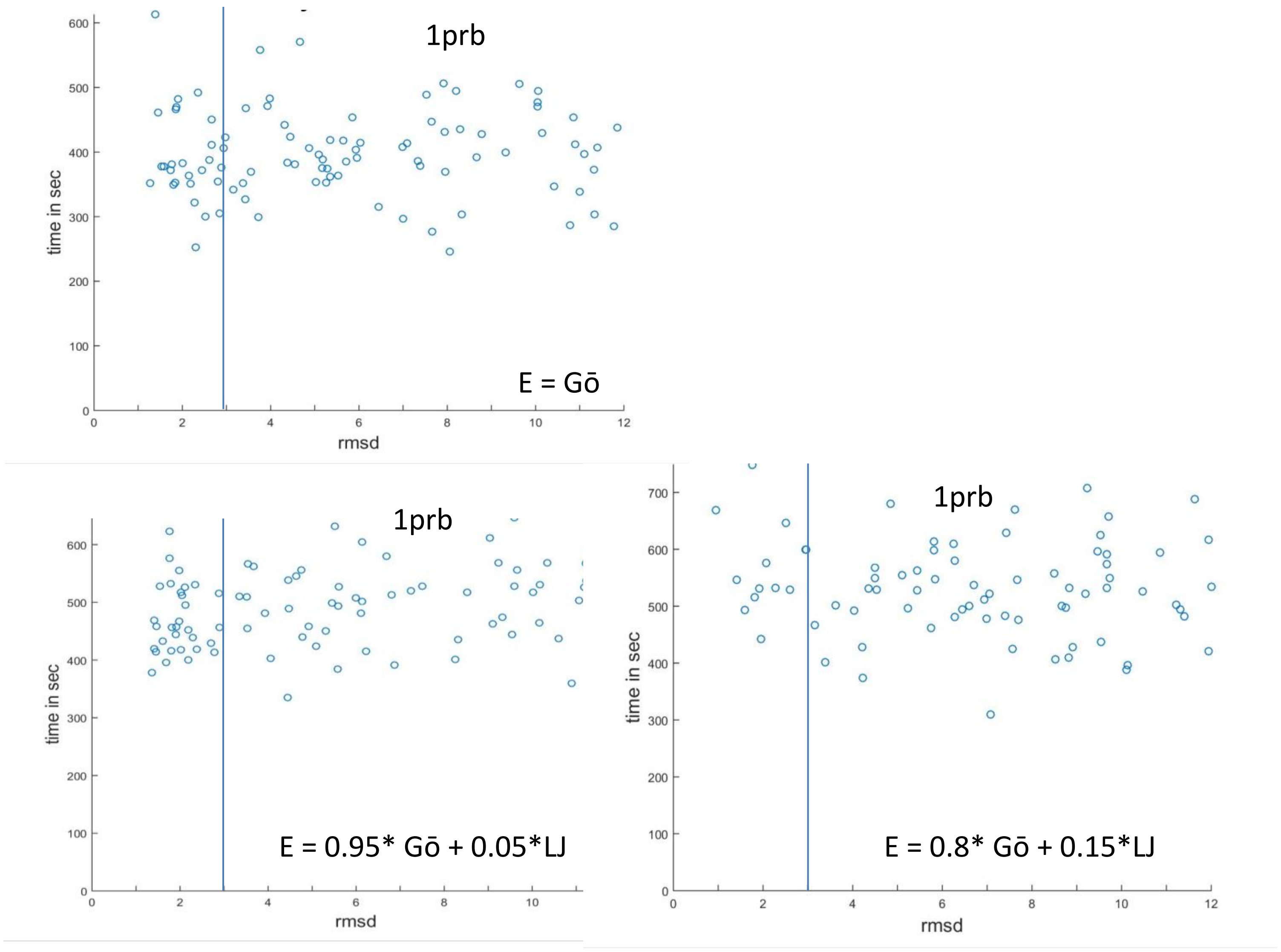
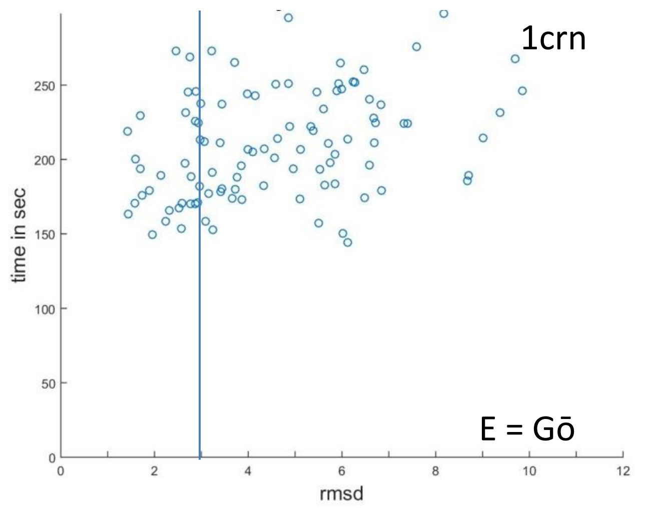
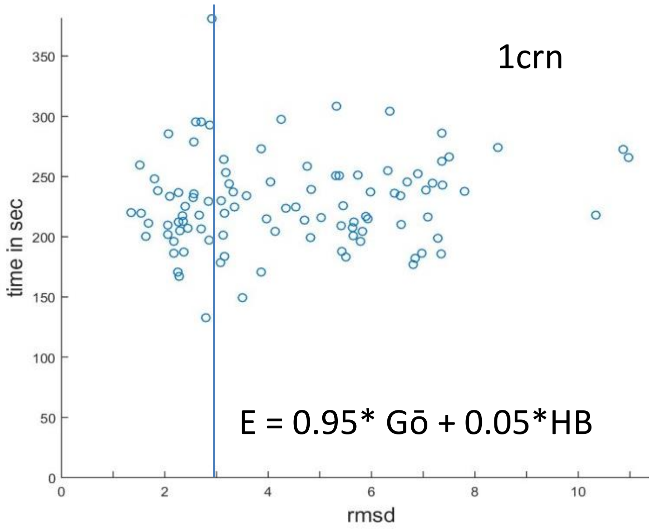


Figure 4.5. Scatter plots RMSD vs folding time for 1prb trajectories folded with E= $w_{Gō}$*Gō+w_{LJ}*LJ potential. The average numbers of folded trajectories with rmsd ≤ 3 Å decreased with no significant change in folding time with the LJ weight increase.





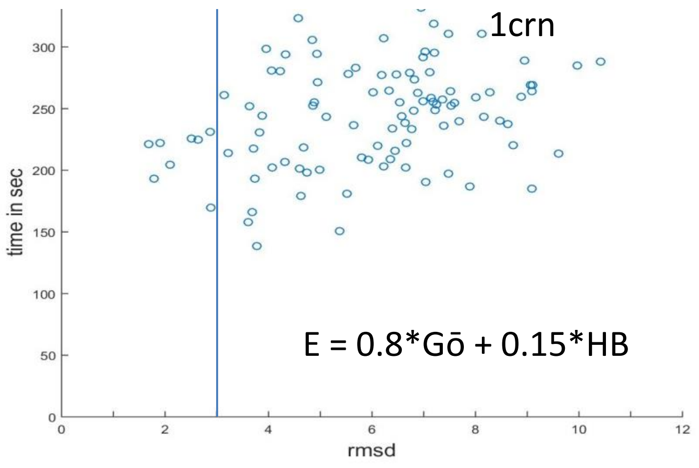
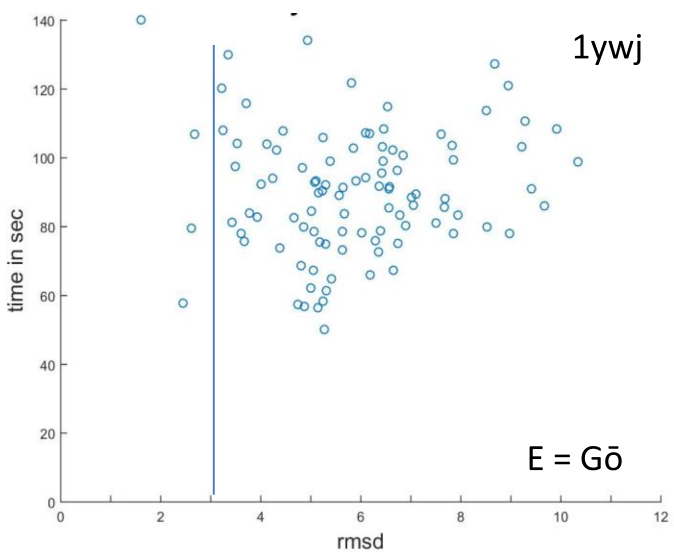
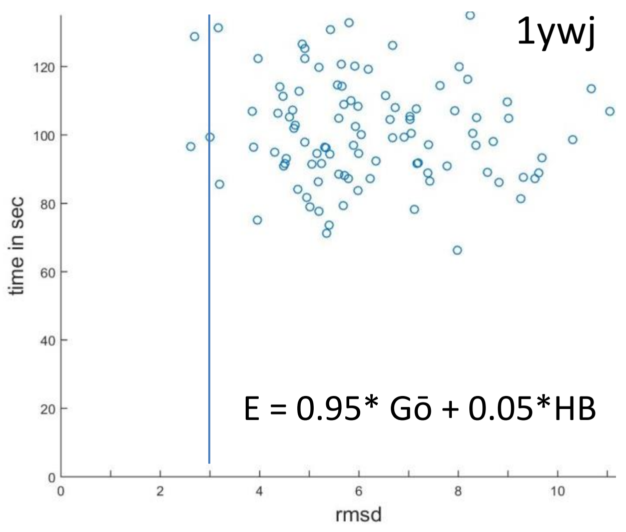


Figure 4.6 Scatter plots RMSD vs folding time for 1crn trajectories folded with E= wGō*Gō+wHB*HB potential. The average numbers of folded trajectories with rmsd ≤ 3 Å decreased with no significant change in folding time with the HB weight increase.





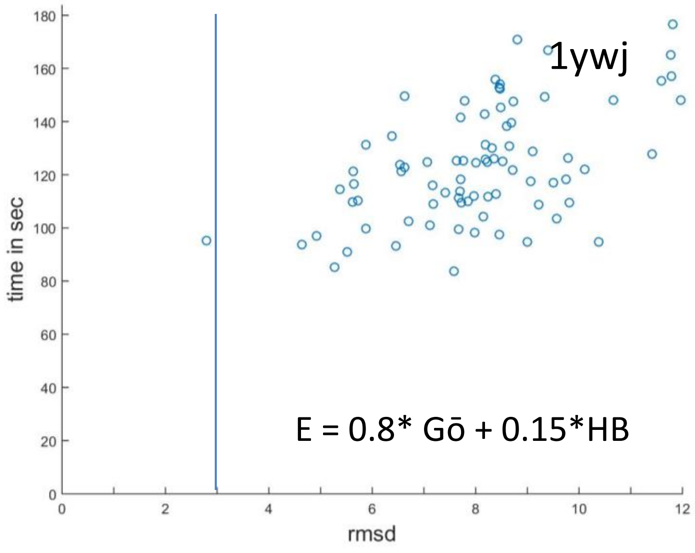


Figure 4.7. Scatter plots RMSD vs folding time for 1ywj trajectories folded with E= $w_{G\bar{o}}$\*Gō+w_{HB}*HB potential. The average numbers of folded trajectories with rmsd ≤ 3 Å decreased with no significant change in folding time with the HB weight increase.

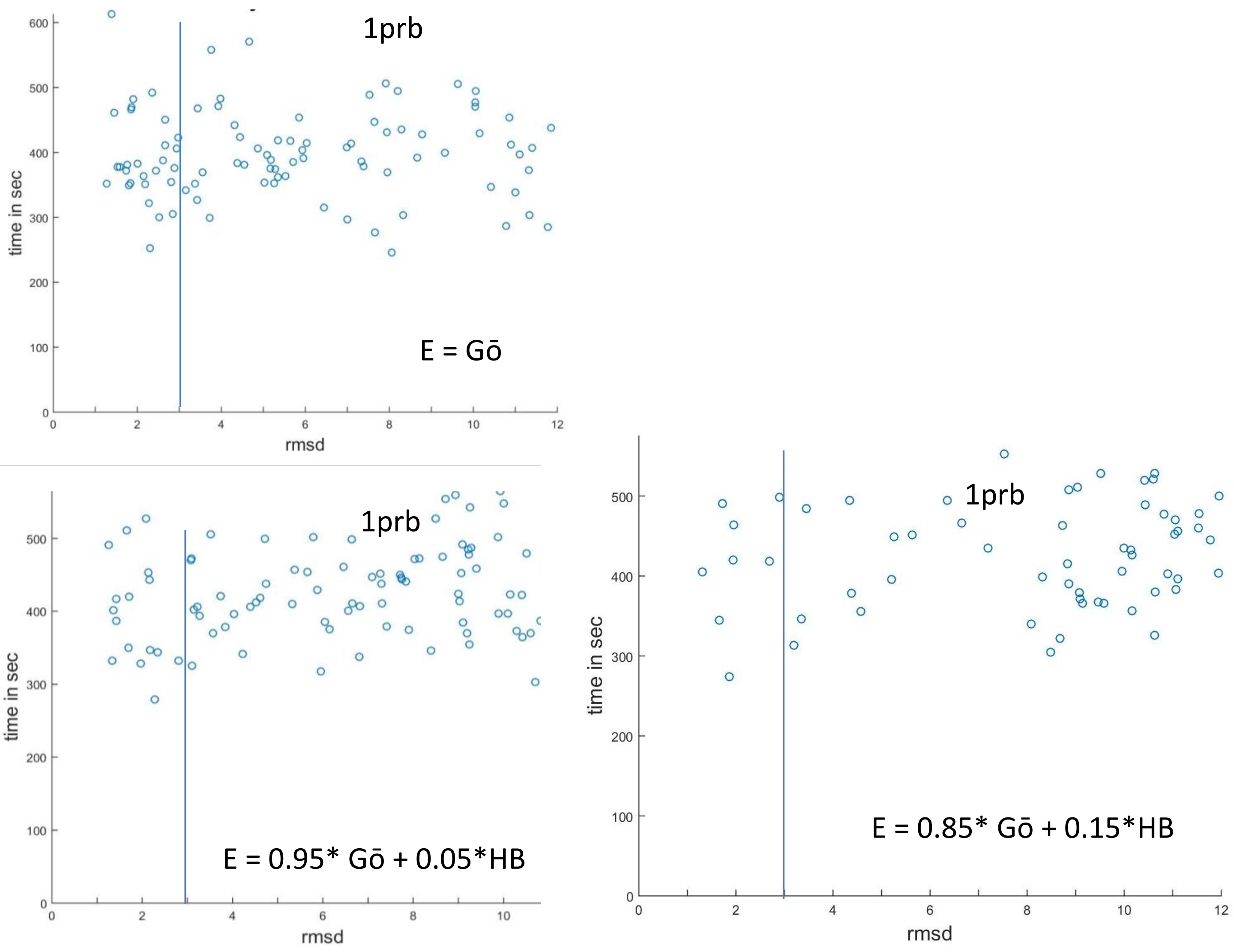
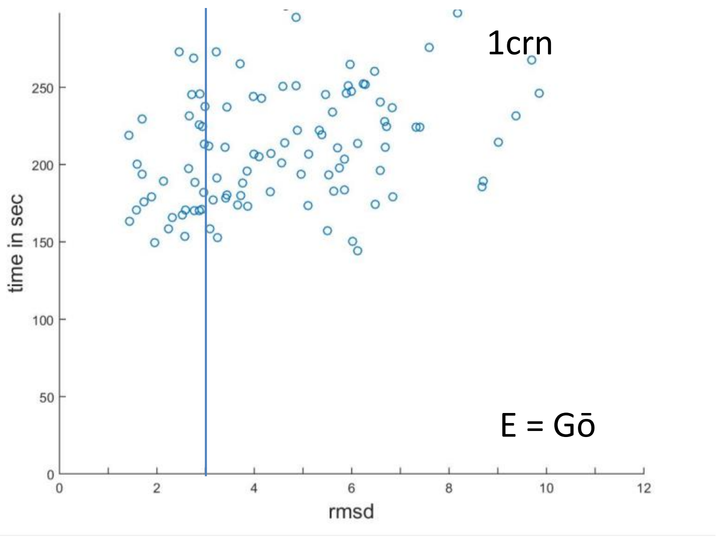
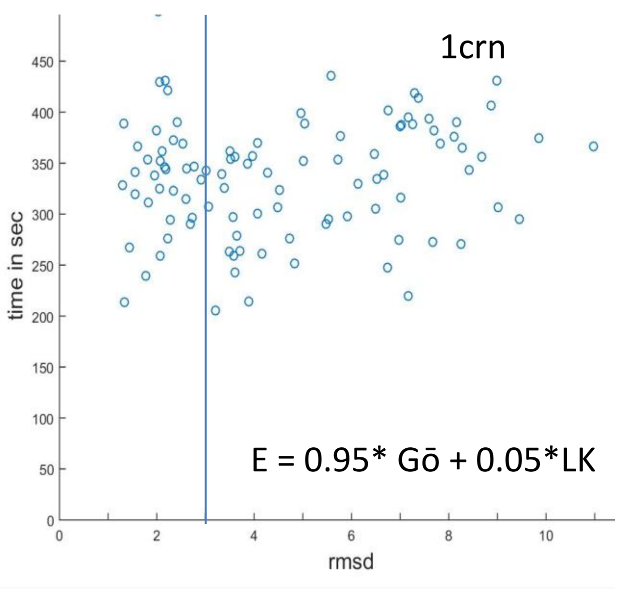


Figure 4.8. Scatter plots RMSD vs folding time for 1prb trajectories folded with E= $w_{G\bar{o}}$\*Gō+w_{HB}*HB potential. The average numbers of folded trajectories with rmsd ≤ 3 Å decreased with no significant change in folding time with the LJ weight increase.





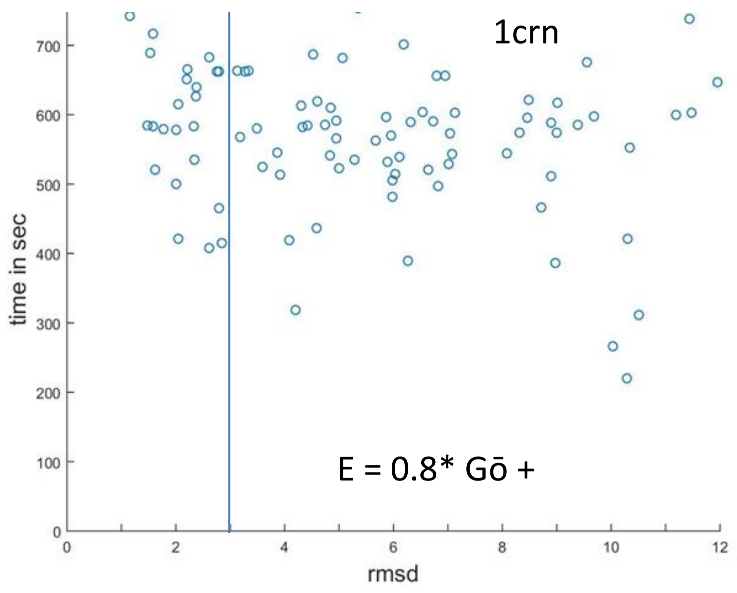
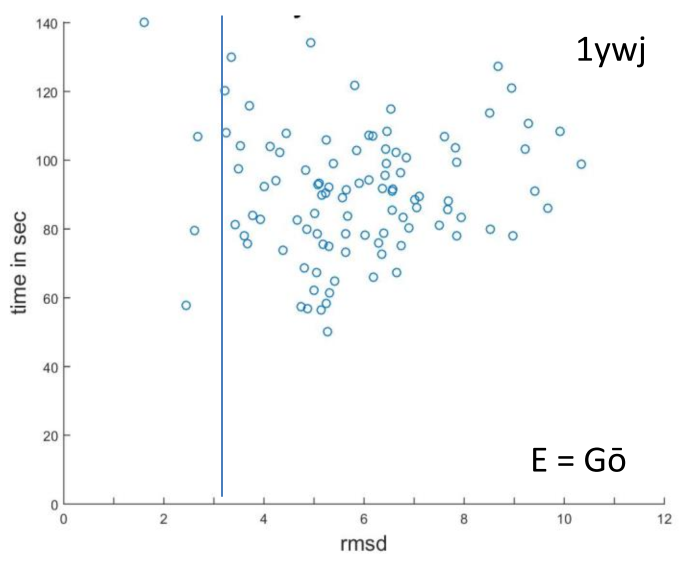
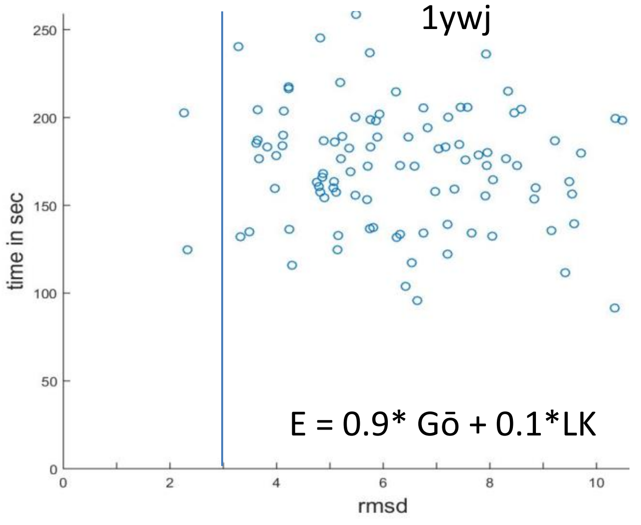


Figure 4.9. Scatter plots RMSD vs folding time for 1crn trajectories folded with E= $w_{Gō}$*Gō+w_{LK}*LK potential. The average numbers of folded trajectories with rmsd ≤ 3 Å did not change with the LK weight increase. Folding time increase was observed.





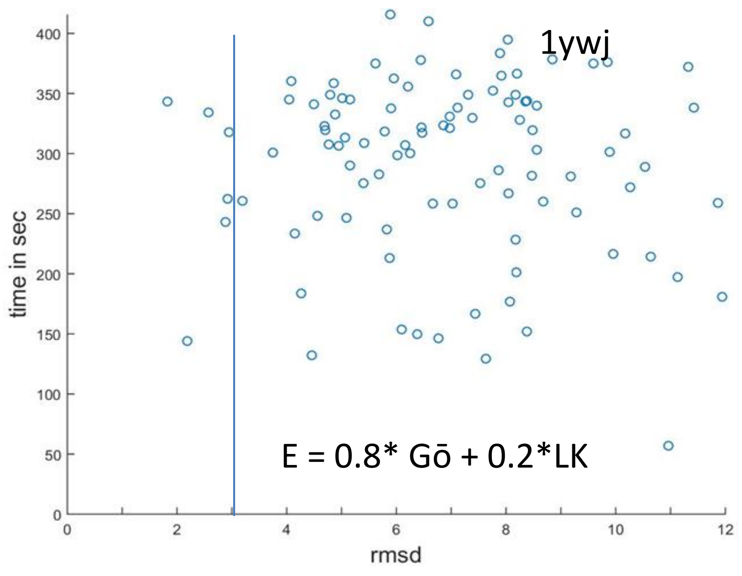


Figure 4.10. Scatter plots RMSD vs folding time for 1ywj trajectories folded with E= wGō*Gō+wLJ*LK potential. The average numbers of folded trajectories with rmsd ≤ 3 Å did not change with the LK weight increase. Folding time increase was observed.

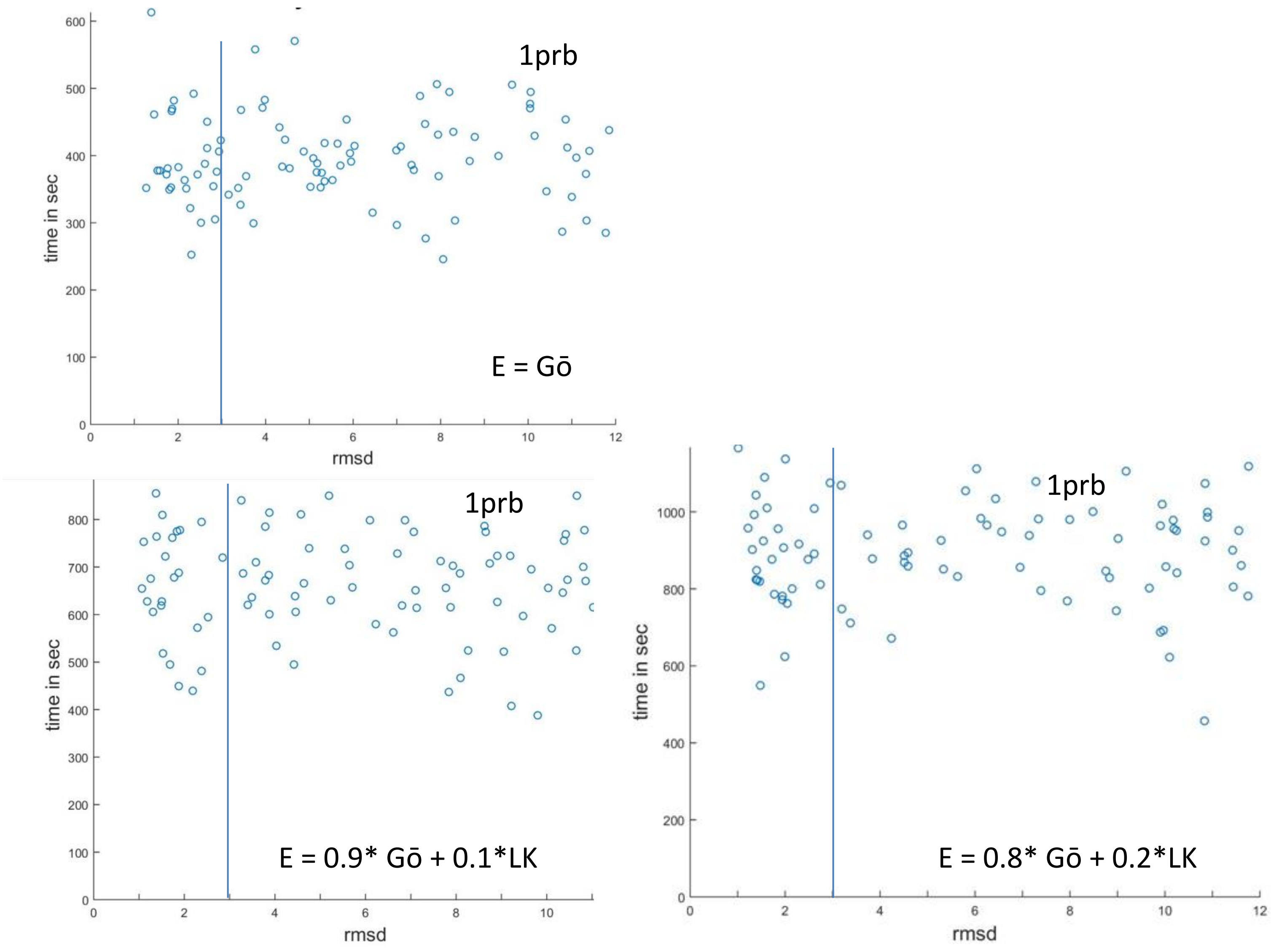


Figure 4.11. Scatter plots RMSD vs folding time for 1prb trajectories folded with E= $w_{Gō}$*Gō+w_{LK}*LK potential. The average numbers of folded trajectories with rmsd ≤ 3 Å did not change with the LK weight increase. Folding time increase was observed.

$E = w_{G\bar{o}} * G\bar{o} + w_{LJ} * LJ + w_{LK} * LK + w_{HB} * HB$	1crn	1ywj	1prb
(1,0,0,0)	35	6	21
(0.8, 0, 0.1, 0.1)	24	1	18
(0.8, 0, 0.15, 0.05)	11	0	4
(0.8, 0, 0.05, 0.15)	28	5	27
(0.8, 0.05, 0.15, 0)	9	1	5
(0.8, 0.05, 0.1, 0.05)	21	0	9
(0.8, 0.05, 0.05, 0.1)	22	3	19
(0.8, 0.05, 0, 0.15)	29	3	22
(0.8, 0.1, 0.1, 0)	10	0	3
(0.8, 0.1, 0, 0.1)	20	4	16
(0.8, 0.1, 0.05, 0.05)	27	0	13
(0.8, 0.15, 0.05, 0)	10	0	6
(0.8, 0.15, 0, 0.05)	17	1	9

Table 4.1. The folding results for energy terms weights taken on a grid. 1st column list weights combinations; 2nd, 3rd and 4th list proteins folded trajectories with RMSD ≤ 3 Å.

4.5 The proposed energy optimization algorithm finds several combinations of weights

I demonstrated that the inclusion of all three energy terms reduced the Gō potential weight to 0.8. I tested further reduction of the Gō potential weight running our Monte Carlo based optimization algorithm. Within fifty iterations I found seventeen combinations of weights with the Gō potential weight below 0.8 (**Table 4.2**). For each of those combinations I repeated simulations five times. The combinations with the Gō potential weight greater 0.54 gave reliable results, which meant the average number of folded trajectories for every protein exceeded three. Two combinations with the lowest Gō potential weight and the highest number of folded trajectories for all tested proteins turned out to be outliers; all five repeated simulations gave zero trajectories for all proteins.

4.6 Correlation between the number of folded trajectories and energy terms weights

In this section I addressed the question: whether the number of folded trajectories correlated with the energy weights or were due to pure chance alone? I used Canonical Correlation Analysis (CCA) to explore the relationships between two random multivariate variables, 3-variable set associated with the number of folded trajectories for the three tested proteins (1crn, 1prb and 1ywj) and 4-variable set that represented weights for four energy terms (the Gō potential, the LJ potential, the LK solvation potential and hydrogen bonding potential) respectively.

1crn	1prb	1ywj	$w_{G\bar{o}}$	w_{LJ}	w_{LK}	w_{HB}
24	19	16	0.502633	0.358179	0.0141239	0.125064
33	38	3	0.503473	0.158081	0.125194	0.213252
15	3	4	0.518735	0.110758	0.110646	0.259862
33	18	3	0.534161	0.273634	0.0219685	0.170237
13	3	5	0.538649	0.403826	0.012412	0.045113
5	4	4	0.540561	0.304886	0.0379914	0.116562
7	12	4	0.541796	0.182007	0.264814	0.011383
4	6	8	0.628141	0.228282	0.0158063	0.127771
15	10	3	0.658026	0.196777	0.120762	0.024435
35	31	7	0.671296	0.156325	0.0448453	0.127533
21	11	3	0.671544	0.0786879	0.182639	0.067129
32	18	8	0.707932	0.0446181	0.214877	0.032574
30	12	3	0.714463	0.0128168	0.165884	0.106837
25	16	7	0.729089	0.0550044	0.14114	0.074767
29	24	3	0.731645	0.0150575	0.209363	0.043934
27	17	4	0.797442	0.0631897	0.0487766	0.090592

Table 4.2. Folded trajectories of training set's proteins and energy terms weights derived with novel energy weights optimization algorithms.

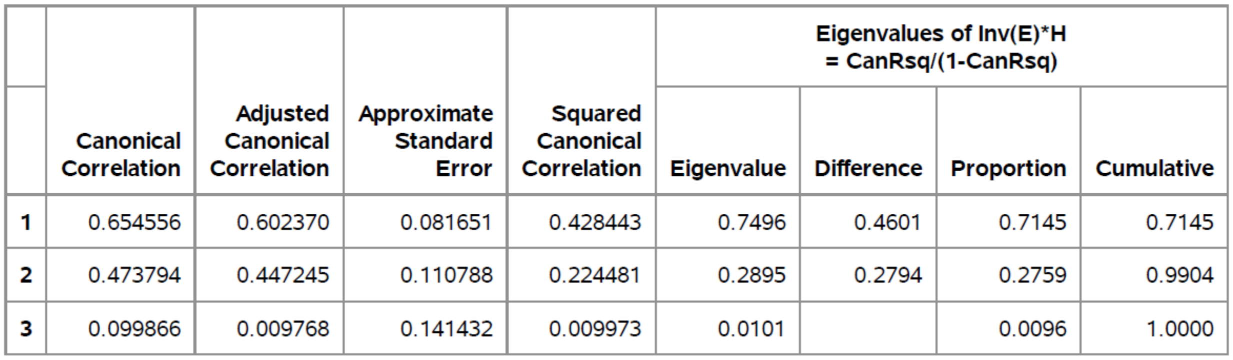
To collect the data for the statistical test, I generated a sample of 50 random 1-by-4 vectors $w^i = ({w^i}_{G\bar{o}}, {w^i}_{LJ}, {w^i}_{LK}, {w^i}_{HB})$, $i = 1 \dots 50$ for energy weights. Vectors were drawn from the Dirichlet distribution with α = 1 and ${w^i}_{G\bar{o}} + {w^i}_{LJ} + {w^i}_{LK} + {w^i}_{HB} = 1$. For each weight vector from the sample I ran 100 trajectories for each protein to calculate the number of folded trajectories for the folding statistics. CCA was carried out in SAS. SAS reported (**Table 4.3**, **a)** - **b)**) the significant canonical correlation γ = 0.654556 with the Wilks' lambda *Λ = 0.43883230; F= 3.47; d.f. = 12, 114.06; p-value = 0.0002* for the first canonical pair, which means that there is a linear correlation between two sets of variables, the energy terms weights and the numbers folded proteins trajectories. Canonical coefficients for the linear correlation are listed in **Table 4.4** in the first columns for each random multivariate variable. Since the Gō potential weight dominated over the other energy terms weights, I ran CCA test for the same weights set with each vector component divided by its Gō weight $w^i = (1, {w^i_{LJ}}/{w^i_{G\bar{o}}}, {w^i_{LK}}/{w^i_{G\bar{o}}}, {w^i_{HB}}/{w^i_{G\bar{o}}})$ to get better sense how the LJ potential, the LK solvation potential and hydrogen bonding potential influence protein structure prediction. The first canonical pair (U_1, V_1) (**Table 4.5**) is represented as:

$U_1 = -0.07 \cdot 1crn - 0.04 \cdot 1prb + 0.13 \cdot 1ywj \quad (13)$

$V_1 = 3.36 \cdot LJ + 2.43 \cdot LK + 5.92 \cdot HB \quad (14)$

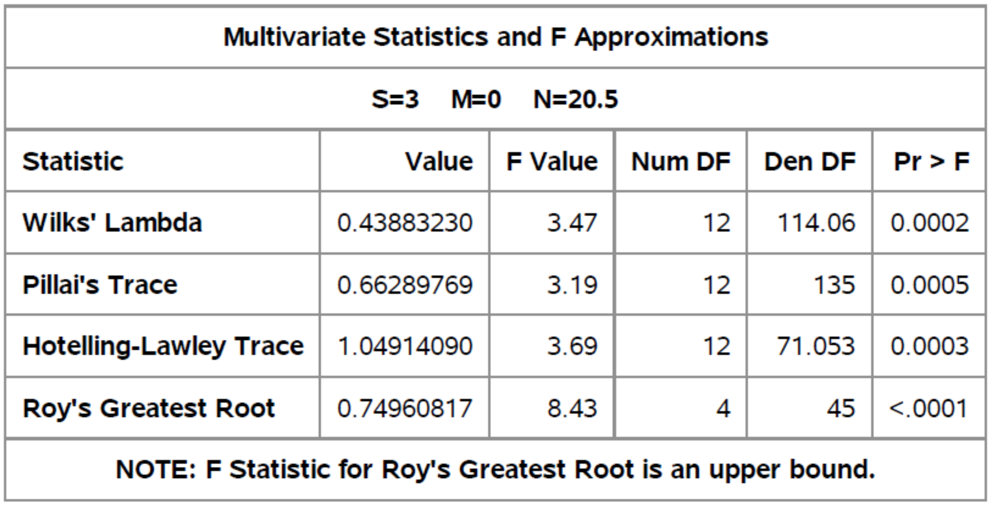
As seen from the values of the canonical coefficients for V_1 protein structure prediction is mostly influences by the LK solvation energy and the LJ potential and to a lesser extent by hydrogen bonding potential.

a)



					Eigenvalues of Inv(E)*H = CanRsq/(1-CanRsq)			
	Canonical Correlation	Adjusted Canonical Correlation	Approximate Standard Error	Squared Canonical Correlation	Eigenvalue	Difference	Proportion	Cumulative
1	0.654556	0.602370	0.081651	0.428443	0.7496	0.4601	0.7145	0.7145
2	0.473794	0.447245	0.110788	0.224481	0.2895	0.2794	0.2759	0.9904
3	0.099866	0.009768	0.141432	0.009973	0.0101		0.0096	1.0000

b)



Multivariate Statistics and F Approximations

S=3 M=0 N=20.5

Statistic	Value	F Value	Num DF	Den DF	Pr > F
Wilks' Lambda	0.43883230	3.47	12	114.06	0.0002
Pillai's Trace	0.66289769	3.19	12	135	0.0005
Hotelling-Lawley Trace	1.04914090	3.69	12	71.053	0.0003
Roy's Greatest Root	0.74960817	8.43	4	45	<.0001

NOTE: F Statistic for Roy's Greatest Root is an upper bound.

Table 4.3 CCA statistics. **a)** Canonical correlation values for 1st, 2nd and 3rd canonical pairs are listed in the first column; **b)** only first 1st canonical pair that has a small Wilk's Lambda value 0.43883230 with p-value 0.0002

a)

Raw Canonical Coefficients for the FOLDED Variables			
	folded1	folded2	folded3
crambin	0.0760690428	-0.118228255	-0.014279988
album	0.0315490716	0.2769424613	0.1195087026
wwdom	-0.047544892	-0.026834682	-0.956488512

b)

Raw Canonical Coefficients for the WEIGHTS Variables			
	weights1	weights2	weights3
GOP	-61.03644685	-29.04409837	151.90481145
LJ	-66.40000738	-33.2027084	143.2834597
LK	-67.80193122	-40.12046958	148.20905099
HB	-72.52305424	-29.05568642	145.96301184

Table 4.4. CCA raw canonical coefficients. a) the first column lists coefficients for 1st canonical variable U_1 which is a linear combination of three random variables that represent folded trajectories for three proteins 1crn (crambin), 1prb (album) and 1ywj (wwdom). b) the first column lists coefficients for 1st canonical variable V_1 which is a linear combination of four random variables that represent energy weights for four energy terms: the Gō potential (GOP), the Lennard Jones potential (LJ), the Lazaridis-Karplus solvation potential (LK) and hydrogen bonding potential (HB).

a)

Raw Canonical Coefficients for the FOLDED Variables			
	folded1	folded2	folded3
crambin	-0.070477343	0.1244951507	0.0130557754
album	-0.049909518	-0.276234669	-0.081875786
wwdom	0.1301304213	0.2814560106	0.6644845845

b)

Raw Canonical Coefficients for the WEIGHTS Variables			
	weights1	weights2	weights3
GOP	.	.	.
LJ	3.3602623616	2.2988760264	-5.39187919
LK	2.4310964263	5.8042170545	2.0399212988
HB	5.9218753657	-0.987508148	1.8863399657

Table 4.5. CCA scaled raw canonical coefficients. Scaling each energy weights combinations to its Gō potential weight simplifies analysis of contributions of the Lennard-Jones potential (LJ), the Lazaridis –Karplus solvation (LK) and hydrogen bonding (HB) potentials to protein structure prediction a) the first column lists coefficients for 1st canonical variable U_1 which is a linear combination of three random variables that represent folded trajectories for three proteins 1crn (crambin), 1prb (album) and 1ywj (wwdom). b) the first column lists coefficients for 1st canonical variable V_1 which is a linear combination of three random variables that represent energy weights for three energy terms: the Lennard Jones potential (LJ), the Lazaridis-Karplus solvation potential (LK) and hydrogen bonding potential (HB).

4.7 Application of optimal weights to predict structure of all-α and α/β proteins

Finally, I conducted experiments to investigate whether the optimized weights can be used to predict structure of proteins different from those that was used in the weight optimization procedure. I chose three small proteins that represent three different folds (**Table 4.6**). For each reliable combination of weights (**Table 4.7**) I ran 100 trajectories for each protein. All simulations were started from fully extended chains (see Methods section for more details). Eleven weights combinations gave at least two folded trajectories with RMSD less than 4 Å for 1i2t, an *all α* protein (**Table 4.2**). The best prediction for the weight combination $w_{Go} = 0.658026, w_{LJ} = 0.196777, w_{LK} = 0.120762, w_{HB} = 0.0244345$ with the smallest weight for the Gō potential had RMSD to native structure 1.38 Å (**Figure 4.11**); the *all-α* protein was folded within 9000 Monte Carlo steps (**Figure 4.11**, **c)**). For 2p5k, an *α/β* protein, two weights combinations gave one folded trajectory with RMSD less than 4.5 Å. The best 2p5k final structure for the weight combination $w_{Go} = 0.658026, w_{LJ} = 0.196777, w_{LK} = 0.120762, w_{HB} = 0.0244345$ had RMSD 4.43 Å (**Figure 4.12**) and was folded within 9000 Monte Carlo steps (**Figure 4.12, c)**). As expected, there were no trajectories for 1zlm, an *all β* protein, with RMSD below 8 Å for any combinations of weights listed in **Table 4.7**.

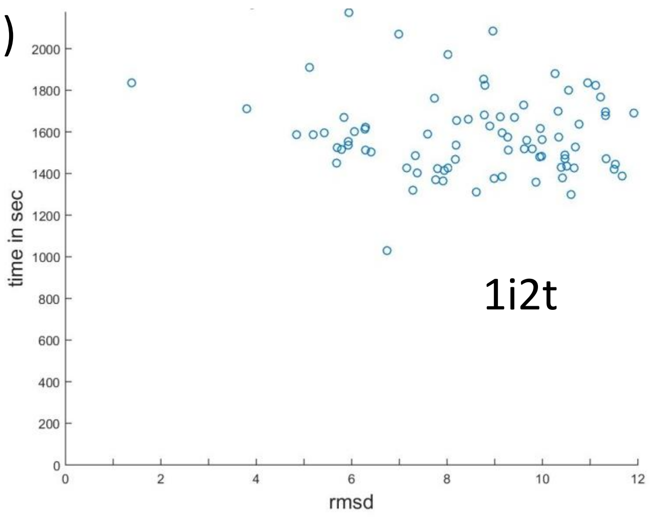
Protein	PDB ID code	Chain	Class	# Residues
Arginine repressor	2p5k	A	α/β	64
HYD protein	1i2t	A	all-α	61
Osteoclast stimulating factor 1	1zlm	A	all -β	58

Table 4.6 Protein structures used to test optimized weights. The 3 proteins belong to different SCOP classes and have comparable sequence lengths.

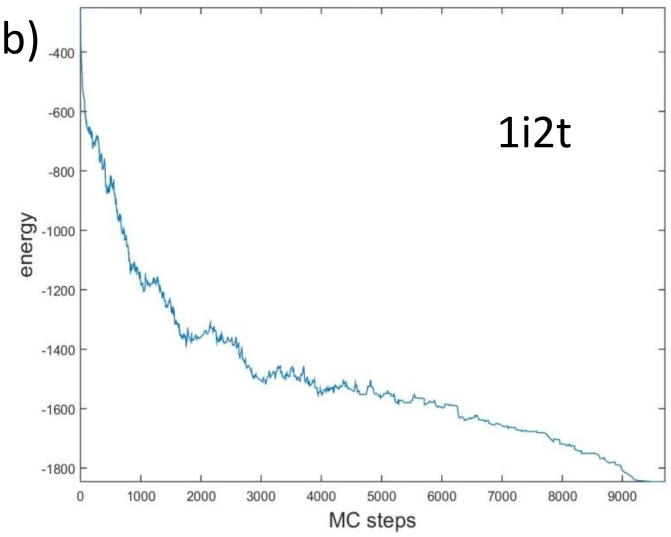
2p5k rmsd ≤ 4.5 Å	1i2t rmsd ≤ 4Å	$w_{G\bar{o}}$	w_{LJ}	w_{LK}	w_{HB}
0	0	0.540561	0.304886	0.0379914	0.116562
0	0	0.541796	0.182007	0.264814	0.011383
0	0	0.628141	0.228282	0.0158063	0.127771
0	2	0.658026	0.196777	0.120762	0.024434
0	0	0.671296	0.156325	0.0448453	0.127533
1	5	0.671544	0.0786879	0.182639	0.067128
0	4	0.707932	0.0446181	0.214877	0.032573
0	2	0.714463	0.0128168	0.165884	0.106837
0	3	0.729089	0.0550044	0.14114	0.074767
0	5	0.731645	0.0150575	0.209363	0.043934
2	8	0.797442	0.0631897	0.0487766	0.090592

Table 4.7. Prediction results for reliable weights for proteins not included into the training set. Folded trajectories derived with optimized energy terms weights.

a)



b)



c) 1i2t

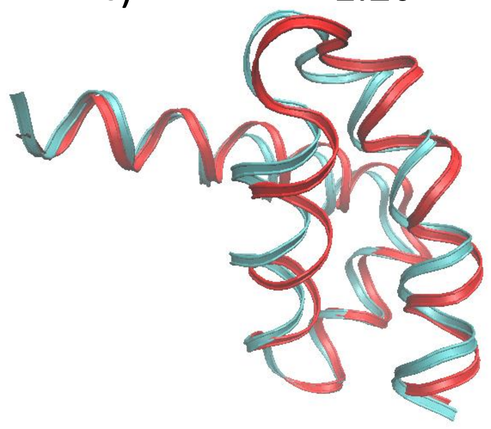
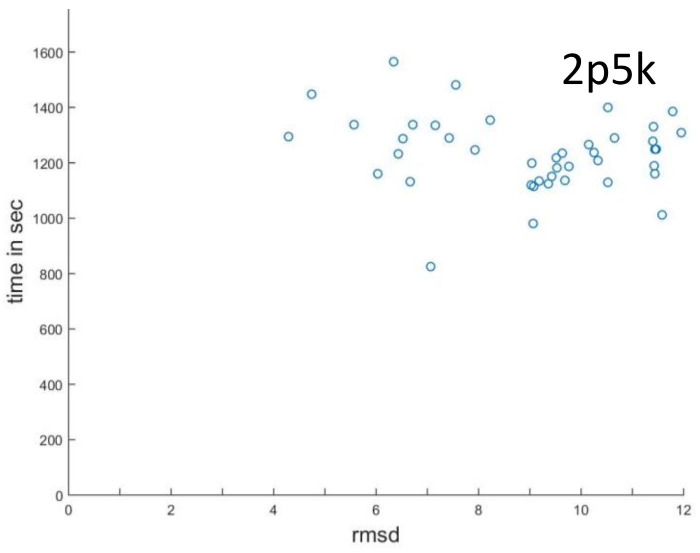
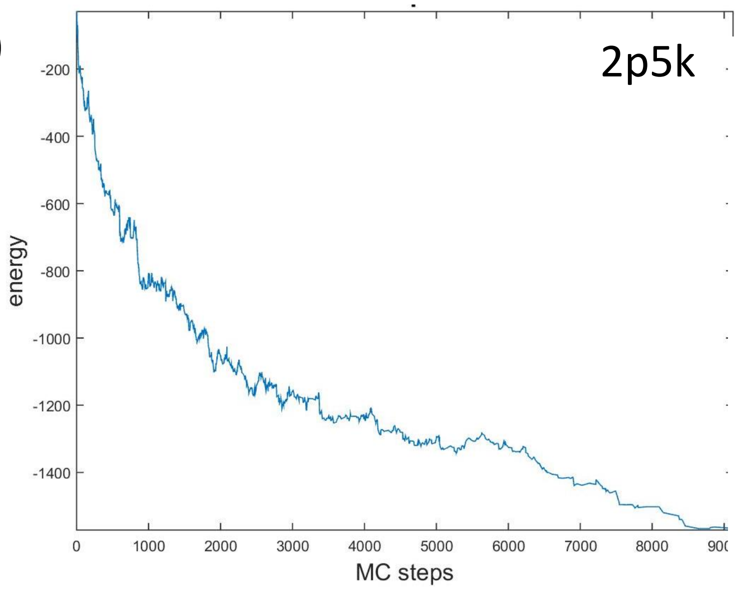


Figure 4.11. **Prediction results for 1i2t.** Optimized energy weights with $w_{G\bar{o}}$ = 0.658 , w_{LJ}= 0.196777, w_{LK} = 0.120762 , w_{HB}= 0.0244345 predict structure of all-α protein 1i2t. a) Scatter plot RMSD vs folding time for 100 trajectories (only those with rmsd ≤12Å are visualized) ; b) energy profile of trajectory with final structure rmsd 1.38 Å; c) superposition of 1i2t native structure (in cyan) with predicted structure (red) with RMSD 1.38Å

a)


b)


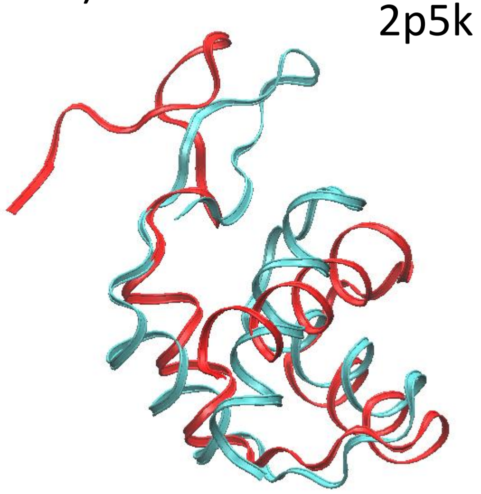
c)


Figure 4.12. **Prediction results for 2p5k.** Optimized energy weights with $w_{G\bar{o}}$ = 0.671544 , w_{LJ}= 0.182639 , w_{LK} = 0.182639 , w_{HB}= 0.0671288 predict α/β protein 2p5k. a) Scatter plot RMSD vs folding time for 100 trajectories (only those with rmsd ≤12Å are visualized); b) energy profile of trajectory with final structure rmsd 4.23 Å; c) superposition of 2p5k native structure (in cyan) with predicted structure (red) with RMSD 4.23Å

Chapter 5

Discussion and Conclusions

There are two widely used approaches that extract energy parameters for protein structure prediction. Linear programming (24, 58) performs optimizations on a large number of parameters and constraints with the assumption that the energy of the native state is lower than all alternative conformations. Z-score minimization (33, 54, 63, 92) requires a large energy gap between the native state and some reference state, represented by an ensemble of compact structures. Our approach is conceptually different; I target a fast protein folding pathway, which is found by a bias toward the native state Gō model scoring function. Since I start all simulations from a fully extended polypeptide chain with the sampling procedure (as any Monte Carlo based sampling) governed by a series of particular changes in dihedrals defined by a researcher, sampling distributions and pseudo-random number generators, I can say that the proposed algorithm is “pathway” directed.

Thus, I proposed a new algorithm for finding the optimal weights of energy terms. I developed a fast sampling algorithm with a Gō potential as a scoring function and assumed the ability of the algorithm to find fast folding pathways. If proteins can be folded with the Gō potential, then the addition of physics- and knowledge-based potentials should improve the protein structure prediction. The structural information encoded in the Lennard-Jones, the Lazaridis-Karplus solvation and hydrogen bonding potentials but missed in the Gō potential alone should help to sustain the adequate number of the folded trajectories and folding speed. Thus, if the energy terms are effective, I should be able to eliminate the Gō potential completely with the proposed optimization procedure. The algorithm I developed found several combinations of weights that predicted successfully the structures of *all-α* and *α/β* proteins not included in the optimization procedure.

I demonstrated that each tested energy term alone was able to reduce the Gō potential weight to 0.8. The Lennard-Jones potential to some extent boosted the number of folded trajectories for 1crn and 1prb and reduced that number for 1ywj. The increase of the hydrogen bonding potential weight gradually decreased the number of folded trajectories for all proteins The Lazaridis-Karplus solvation potential was beneficial for all proteins. Contrary to hydrogen bonding and the Lennard–Jones potentials, the solvation energy kept the number of folded trajectories at the same level. However, for all proteins I observed an increase in folding time.

Even though I showed that one added energy term could reduce the Gō potential weight, it was unclear whether this was due to the extra information brought by a potential or the introduction of computational noise. To answer this question, I needed to reduce the Gō potential weight by adding all three energy terms, the Lennard-Jones, the Lazaridis-Karplus solvation and hydrogen bonding potentials, and to get acceptable percentage of folded trajectories and folding time for each of three proteins. To simplify the task, I tested weights for energy terms chosen from a grid $0.8 \leq w_{G\bar{o}} < 1,\ 0 <$

$w_{LJ/solv/HB} \leq 0.2$. I was shown that the addition of several energy terms indeed allowed the reduction of the Gō potential weight while sustaining reasonable folding time and an adequate number of folded trajectories for all proteins. The values for the weights also implied that the better results were collected for the weights that gave the best performance for a single energy term addition.

I applied our energy weights optimization algorithm to perform a rigorous search for weights for four energy terms, the Gō potential, the Lennard-Jones Potential, the Lazaridis-Karplus potential, by minimizing the Gō potential weight. Sixteen weights combinations were found that satisfied our definition of successful simulations. I encountered two outliers with the Gō potential weight below 0.54 that produced unusually high numbers of folded trajectories for all proteins. The outcome in those cases did not correlate with the weights that gave the best performance when only one energy term was added to the Gō potential. Also, four out of eleven reliable combinations of weights had values for the solvation energy weights that were lower than weights for the Lazaridis-Karplus solvation and hydrogen bonding potentials.

The linear correlation between the numbers of folded trajectories and the energy term weights was confirmed by a CCA statistical test. Moreover, it was shown that the Lazaridis-Karplus and the Lennard-Jones potentials played a more significant role than hydrogen bonding potential in protein structure prediction. CCA might be viewed as an instrument for the assessment of potency of energy terms used to compose an energy function. Adding a newly designed energy term to the Gō potential and testing it with different weights provides a new tool to evaluate its effectiveness.

Finally, I applied the derived weights to predict proteins not included into the optimization procedure. The algorithm demonstrated better performance for all-α and α/β proteins than for all β protein. Proteins with β-sheets are challenging for *ab initio* structure prediction for several reasons: 1) the difficulty of efficiently sampling long-range strand pairings; 2) a very high entropic cost once β-strand pairings formed, which disallows further perturbations to refine a structure; 3) and the high number of alternative nonlocal β-sheet topologies that expands the conformational search (11, 60). Our algorithm utilizes a simple backbone move that limits the ability of the sampling algorithm to maximize the number of hydrogen bonds that form β-sheets. The energy minimization routine froze conformations without any attempts to align β-strands when a substantial number of hydrogen bonds was found. Hydrogen bonding potential alone was not able to fix the problem. To deal with this problem, Bradley and Baker (11) proposed a new multilevel sampling method to β-sheet structure prediction that overcomes this difficulty by reformulating structure generation in terms of a folding tree composed of peptide segments and long-range connections. Nonlocal connections in this tree allow them to explicitly sample alternative β-strand pairings while simultaneously exploring local conformational space using backbone torsion-space moves. This method uses an iterative, energy-biased resampling approach that selects nonlocal pairing from previous iterations and explores them while stochastically disfavoring local ones. The method can sample the nonlocal interactions to navigate the search into the promising areas of the huge conformational space. Therefore, future improvements to our algorithm will require a new procedure that improves β-sheet prediction with effective hydrogen bonding potential. I also intend to carry out similar studies on larger sets of energy terms and benchmark proteins to eliminate the Gō potential completely.

In this work, I demonstrated a fast and effective way to optimize energy functions for protein structure prediction and the ability to fold proteins with an all-atom, *ab initio* algorithm without involving extra information derived from PDB databases (fragments libraries and templates), using ordinary computational resources.

Appendices

Mersenne Twister C/C++ implementation

I tested several implementations of Mersenne Twister algorithms that generate random numbers. The version of Takuji Nishimura and Makoto Matsumoto showed itself to be very reliable for different Operational Systems and C/C++ compilers.

```
A C-program for MT19937, with initialization improved 2002/2/10.
   Coded by Takuji Nishimura and Makoto Matsumoto.
   This is a faster version by taking Shawn Cokus's optimization,
   Matthe Bellew's simplification, Isaku Wada's real version.

   Before using, initialize the state by using init_genrand(seed)
   or init_by_array(init_key, key_length).

   Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
   All rights reserved.

/*
   C++ codes by Kohei Takeda (k-tak@letter.or.jp)
   Redistribution terms are the same as the original ones above.
*/

#ifndef ___MERSENNE_TWISTER_RNG___
#define ___MERSENNE_TWISTER_RNG___

#include <ctime>
#include <cstdlib>
#include <cassert>


struct Mt32Traits
{
      typedef unsigned int        UINTTYPE;
      typedef signed int          INTTYPE;
      static const int                   INTTYPE_BITS  = 32;
      static const unsigned int          MAXDOUBLEVAL  = 4294967295U; //2^32-1
      static const size_t         NN            = 624;
      static const size_t         MM            = 397;
      static const unsigned int          INITVAL              = 1812433253U;
      static const unsigned int          ARRAYINITVAL_0       = 19650218U;
      static const unsigned int          ARRAYINITVAL_1       = 1664525U;
      static const unsigned int          ARRAYINITVAL_2       = 1566083941U;

      static unsigned int twist(const unsigned int& u, const unsigned int& v)
      {
        static unsigned int mag01[2] = {0U, 0x9908b0dfU};
        return ((((u & 0x80000000U) | (v & 0x7fffffffU)) >> 1) ^ mag01[v&1]);
      }

      static unsigned int temper(unsigned int y)
      {
        y ^= (y >> 11);
```

```
          y ^= (y << 7) & 0x9d2c5680U;
          y ^= (y << 15) & 0xefc60000U;
          y ^= (y >> 18);

          return y;
        }
};

struct Mt64Traits
{
        typedef unsigned long long   UINTTYPE;
        typedef signed long long            INTTYPE;
        static const int                    INTTYPE_BITS  = 64;
        static const unsigned long long     MAXDOUBLEVAL  = 9007199254740991ULL; //
2^53-1
        static const size_t          NN            = 312;
        static const size_t          MM            = 156;
        static const unsigned long long     INITVAL              =
6364136223846793005ULL;
        static const unsigned long long     ARRAYINITVAL_0       = 19650218ULL;
        static const unsigned long long     ARRAYINITVAL_1       =
3935559000370003845ULL;
        static const unsigned long long     ARRAYINITVAL_2       =
2862933555777941757ULL;

        static unsigned long long twist(const unsigned long long& u, const unsigned
long long& v)
        {
          static unsigned long long mag01[2] = {0ULL, 0xB5026F5AA96619E9ULL};
          return ((((u & 0xFFFFFFFF80000000ULL) | (v & 0x7FFFFFFFULL)) >> 1) ^
mag01[v&1]);
        }

        static unsigned long long temper(unsigned long long y)
        {
          y ^= (y >> 29) & 0x5555555555555555ULL;
          y ^= (y << 17) & 0x71D67FFFEDA60000ULL;
          y ^= (y << 37) & 0xFFF7EEE000000000ULL;
          y ^= (y >> 43);

          return y;
        }
};


template <typename Traits>
class MtRng
{
   public:
        typedef typename Traits::UINTTYPE   UINTTYPE;
        typedef typename Traits::INTTYPE    INTTYPE;

   protected:
        // member variables
        UINTTYPE*             state_;
size_t          left_;
        UINTTYPE*             next_;
```

```
protected:
   void nextState()
   {
     UINTTYPE *p = state_;
     size_t j;

     left_ = Traits::NN;
     next_ = state_;

     for (j=Traits::NN-Traits::MM+1; --j; p++)
        *p = p[Traits::MM] ^ Traits::twist(p[0], p[1]);

     for (j=Traits::MM; --j; p++)
        *p = p[Traits::MM-Traits::NN] ^ Traits::twist(p[0], p[1]);

     *p = p[Traits::MM-Traits::NN] ^ Traits::twist(p[0], state_[0]);
   }

public:
   MtRng()
   {
     left_ = 1;
     next_ = NULL;
     state_ = (UINTTYPE*)malloc(sizeof(UINTTYPE) * Traits::NN);
     init((UINTTYPE)time(NULL));
   }

   MtRng(UINTTYPE seed)
   {
     left_ = 1;
     next_ = NULL;
     state_ = (UINTTYPE*)malloc(sizeof(UINTTYPE) * Traits::NN);
     init(seed);
   }

   MtRng(UINTTYPE initkeys[], size_t keylen)
   {
     left_ = 1;
     next_ = NULL;
     state_ = (UINTTYPE*)malloc(sizeof(UINTTYPE) * Traits::NN);
     init(initkeys, keylen);
   }

   virtual ~MtRng()
   {
     if (state_) {
        free(state_);
     }
   }

   void init(UINTTYPE seed)
   {
     assert(sizeof(UINTTYPE)*8 == (size_t)Traits::INTTYPE_BITS);

     state_[0]= seed;
     for (size_t j=1; j<Traits::NN; j++) {
        state_[j]
```

```
              = (Traits::INITVAL * (state_[j-1] ^ (state_[j-1] >>
(Traits::INTTYPE_BITS-2)))
                + (UINTTYPE)j);
        }
        left_ = 1;
      }

      void init(UINTTYPE initkeys[], size_t keylen)
      {
        init(Traits::ARRAYINITVAL_0);

        size_t i = 1;
        size_t j = 0;
        size_t k = (Traits::NN > keylen ? Traits::NN : keylen);

        for (; k; k--) {
           state_[i]
              = (state_[i]
                ^ ((state_[i-1] ^ (state_[i-1] >> (Traits::INTTYPE_BITS-2)))
                   * Traits::ARRAYINITVAL_1))
              + initkeys[j] + (UINTTYPE)j; /* non linear */

           i++;
           j++;

           if (i >= Traits::NN) {
              state_[0] = state_[Traits::NN-1];
              i = 1;
           }
           if (j >= keylen) {
              j = 0;
           }
        }

        for (k=Traits::NN-1; k; k--) {
           state_[i]
              = (state_[i]
                ^ ((state_[i-1] ^ (state_[i-1] >> (Traits::INTTYPE_BITS-2)))
                   * Traits::ARRAYINITVAL_2))
              - (UINTTYPE)i; /* non linear */

           i++;

           if (i >= Traits::NN) {
              state_[0] = state_[Traits::NN-1];
              i = 1;
           }
        }

        /* MSB is 1; assuring non-zero initial array */
        state_[0] = (UINTTYPE)1 << (Traits::INTTYPE_BITS-1);
        left_ = 1;
      }

      /* generates a random number on [0,2^bits-1]-interval */
      UINTTYPE getUint()
      {
        if (--left_ == 0) nextState();
```

```
          return Traits::temper(*next_++);
        }

        /* generates a random number on [0,2^(bits-1)-1]-interval */
        INTTYPE getInt()
        {
          if (--left_ == 0) nextState();
          return (INTTYPE)(Traits::temper(*next_++)>>1);
        }

        /* generates a random number on [0,1]-real-interval */
        double getReal1()
        {
          if (--left_ == 0) nextState();
          if (Traits::INTTYPE_BITS > 53) {
             return (
                (double)(Traits::temper(*next_++)>>(Traits::INTTYPE_BITS-53))
                * (1.0 / 9007199254740991.0)
                );
          } else {
             return (
                (double)Traits::temper(*next_++) *
(1.0/(double)Traits::MAXDOUBLEVAL)
                );
          }
        }

        /* generates a random number on [0,1)-real-interval */
        double getReal2()
        {
          if (--left_ == 0) nextState();
          if (Traits::INTTYPE_BITS > 53) {
             return (
                (double)(Traits::temper(*next_++)>>(Traits::INTTYPE_BITS-53))
                * (1.0 / 9007199254740992.0)
                );
          } else {
             return (
                (double)Traits::temper(*next_++) *
(1.0/((double)Traits::MAXDOUBLEVAL+1.0))
                );
          }
        }

        /* generates a random number on (0,1)-real-interval */
        double getReal3()
        {
          if (--left_ == 0) nextState();
          if (Traits::INTTYPE_BITS > 52) {
             return (
                ((double)(Traits::temper(*next_++)>>(Traits::INTTYPE_BITS-52)) +
0.5)
                * (1.0 / 4503599627370496.0)
                );
          } else {
             return (
                ((double)Traits::temper(*next_++) + 0.5) *
(1.0/((double)Traits::MAXDOUBLEVAL+1.0))
```

```
                );
            }
        }


};

typedef MtRng<Mt32Traits> MtRng32;
typedef MtRng<Mt64Traits> MtRng64;
```

Program Parameters, Program Running and Programs to analyze data

Configuration files (with extension .cfg) are employed to set the program's parameters. All parameter files are located in /home/safronova/clOOPS-0.9.6/prj/AbInitioSimple/cfg directory. For *ab initio* routine (ab initio plugin), AbInitio.cfg is used. Each energy plugin also has its own configuration file: gop.cfg corresponds to the Gō potential, chlk.cfg corresponds to the Lazaridis-Karplus solvation potential, chlj2.cfg is used for the Lennard-Jones potential and BMKhbond-B.cfg is designated for hydrogen bonding potential. PDB structures for proteins are located in

/home/safronova/clOOPS-0.9.6/prj/AbInitioSimple/nat

In AbInitio.cfg a researcher should define energy terms and their weights that are going to be used in simulations. All parameters for Monte Carlo algorithm, Simulated Annealing are set through AbInitio.cfg.

To start the program that runs N trajectories for each protein from the list Test_Poteins_lst the following command is used:

```
./run_oops -simInfo GO -i Test_Protein_lst -nTrj 100
```

Where -simInfo option is a name of an experiment, –i option is a file name that contains pdb ids for proteins and –nTrj option is used to set the number of trajectories for each protein from a list in Test_Protein_lst file. Executable file run_oops is located in

/home/safronova/clOOPS-0.9.6/src/PL-tools/C++/AbInitioSimple/src. A source and a header files are located in

/home/safronova/clOOPS-0.9.6/src/PL-tools/C++/AbInitioSimple/run_oops

To run the optimization procedure the following command is used:

./optimize_energy_weights -prevGOw 1 -lGO 0 -hGO 1 -nextSim 0 -nRun 100 -percent 3 -i Test_Protein_lst -nTrj 100

Where –prevGOw is used to set a previous value for the Gō weight. If simulations are just started this value is equal 1, options - lGO and –hGO set the range of the Gō potential weight to be tested. To eliminate the Go weight –lGO should be set to 0 and –hGO should be set to 1. Option –nextSim sets a simulation step to start with, the total number of steps is set to 100. Option –i sets the name for a file that contains proteins pdb ids. Option –percent serves to set the percentage of successful trajectories with final structures rmsd ≤ 3 Å to define success.

To prepare data for analysis and visualization sort_files program must be run.

qsub -b y -j y -cwd sort_files Test_Protein_lst Test_Time_lst SimulationName

where Test_Protein_lst is the name of a file that contains proteins' pdb ids, Test_Time_lst is the name of a file that contains proteins' expected folding times, and the last parameter is a simulation name (used with option –simName for run_oops and optimize_energy_weights programs). The program creates folder Results_SimulationName in trajectories output folder (see run_opps.cpp file's initiation section fot TRJ_DIR). The data contained in Results_SimulationName is used by MATLAB program to create scatter plots and energy profiles for successful trajectories.

Program run_oops

```
//run_oops.cpp
#include <cstdio>
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
#include <cassert>
#include <list>
#include <stdlib.h>
#include <unistd.h>/// for unix
#include <ctime>
using namespace std;

string PRJ_DIR = "/home/safronova/clOOPS-0.9.6/prj/AbInitioSimple";
//string TRJ_DIR = "/home/safronova_1/clOOPS-0.9.6/prj/AbInitioSimple/trj";
string TRJ_DIR = "/home/safronova_3/trj";
string CFG =  PRJ_DIR + "/cfg/AbInitioFold.cfg";
string PDB_DIR = PRJ_DIR + "/nat";
//string TRJ_DIR = PRJ_DIR + "/trj";
//string CMD_NAME = "qsub -b y -j y -cwd oops";
//string CMD_NAME = "qsub -b y -j y -q fast.q -cwd oops";
string CMD_NAME = "qsub -b y -j y -cwd -N oopsbatch oops";
string sCB = "";
string CMD_NEW_DIR = "mkdir -m a=rwx -p";


// Parameters.
string InputLst, SimInfo;
int NTrjs, NSimulationSteps = 1000000;

int StrToInt(const string &str) { return atoi(str.c_str()); }

/* reverse:  reverse string s in place */
void reverse(char s[])
{
    int c, i, j;

    for (i = 0, j = strlen(s)-1; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}


/* itoa:  convert n to characters in s */
void itoa(int n, char s[])
{
    int i, sign;
    if ((sign = n) < 0)  /* record sign */
        n = -n;          /* make n positive */
    i = 0;
    do {       /* generate digits in reverse order */
        s[i++] = n % 10 + '0';   /* get next digit */
    } while ((n /= 10) > 0);     /* delete it */
    if (sign < 0)
```

```
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

void PrintHelp()
{
       const char *const PROG_USAGE = "run_oops -i <input file without ext and path> -
simInfo <string:date[additional info]> -n <number of simulations>";

    const char *const PROG_DESC = "run_oops runs n simulations(n copies of oops
program)";

    cout << "USAGE: " << PROG_USAGE << '\n';
    cout << PROG_DESC << '\n';

    exit(0);
}

void PrintError(string err_str, int err_code)
{
    cout << "run_oops error: " << err_str << '\n';
    cout << "Try run_oops --help to get some more information." << '\n';
    exit(err_code);
}

void ReadParams(int argc, char *argv[])
{
    string args, par, val;
    int i, NextArgIsVal = -1;

    for (i = 1; i < argc; i++)
    {
        args = argv[i];

        if (-1 < NextArgIsVal)
        {
            switch (NextArgIsVal)
            {
                case 0:
                    InputLst = args;
                    break;
                case 1:
                    SimInfo = args;
                    break;
                case 2:
                    NTrjs = StrToInt(args);
                    break;
                            case 3:
                    NSimulationSteps = StrToInt(args);
                    break;

            }
            NextArgIsVal = -1;
        }
        else
        {
            NextArgIsVal = -1;
            if (args == "--help") PrintHelp();else
                if (args == "-sCB") sCB = " -sCB";else
            if (args == "-i") NextArgIsVal = 0; else
            if (args == "-simInfo") NextArgIsVal = 1; else
            if (args == "-nTrj") NextArgIsVal = 2; else
                if (args == "-nSim") NextArgIsVal = 3; else
                   PrintError("unrecognized option", 1);
          }
        }

}
int main (int argc, char *argv[])
{
       ReadParams (argc, argv);

       ifstream in (InputLst.c_str());
       if(!in){cout << "can't open InputLst \n"; exit (1);}

       list<string> ProtLst;
       list<string>::iterator lstIter;

       string line, prot_name;
       while (getline (in, line)){
              //we assume a line doesn't contain spaces
              prot_name = line;
              ProtLst.push_back (prot_name);
       }
       in.close ();

       char buffer0 [33];
       buffer0 [0]= '\0';
       itoa (NSimulationSteps, buffer0);

       for (lstIter = ProtLst.begin (); lstIter != ProtLst.end (); lstIter ++){

              string trj_full_dir = TRJ_DIR + "/" + SimInfo + "/" + *lstIter;
              string cmd_dir = CMD_NEW_DIR + " " + trj_full_dir;
              system (cmd_dir.c_str());
              string trj = trj_full_dir + "/" + *lstIter + "_" + SimInfo; // number will
be added in cycle
              string pdb_file = PDB_DIR + "/" + *lstIter + ".pdb";
              string cmd_files = CMD_NAME + " -c " + CFG + " -i " + pdb_file +  + " -n "
+ buffer0 + " -o " + trj;

              char buffer1 [33];
              char buffer2 [33];

              string FileNameLst = *lstIter + "_" + SimInfo + "_lst";
              ofstream FileLst(FileNameLst.c_str());
              if(!FileLst)PrintError("can't open list file", 3);

              string cmd;
              int seed = time(NULL);

              for ( int i = 1; i <= NTrjs; i++ )
              {
                     seed += i;
                     buffer1[0]= '\0';
```

```
                        buffer2[0]= '\0';
                        itoa (i,buffer1);
                        itoa (seed,buffer2);
                        cmd = cmd_files + "_" + buffer1 + " -seed " + buffer2 + sCB ;
                        FileLst << trj + "_" + buffer1 + "_stat" << endl;
                        system (cmd.c_str ());
                        sleep (1);
                }


                FileLst.close ();
        }
        return 0;
}
```

Data Analysis Program

```
//sort_files.cpp

#include <stdio.h>
#include <cstdlib>

#include <cstdio>
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
#include <cassert>
#include <list>
#include <stdlib.h>
#include <unistd.h>/// for unix
#include <ctime>
#include <utility>
#include <string>
#include <vector>
#include <sstream>

using namespace std;
string TOOL_DIR = "/home/safronova/clOOPS-0.9.6/src/PL-tools/C++/AbInitioSimple";
string RMSD_DIR = TOOL_DIR + "/calc_rmsd/bin/";
string TIME_DIR = TOOL_DIR + "/combine_files/";

string PRJ_DIR = "/home/safronova/clOOPS-0.9.6/prj/AbInitioSimple";
string TRJ_DIR = "/home/safronova_3/trj";
string CFG =  PRJ_DIR + "/cfg/AbInitioFold.cfg";
string PDB_DIR = PRJ_DIR + "/nat";

string CMD_RMSD_NAME = RMSD_DIR +"calc_rmsd -cfg " + PRJ_DIR + "/cfg/Calc_RMSD.cfg";
//string CMD_RMSD_NAME = RMSD_DIR +"calc_rmsd ";

string CMD_TIME_NAME = TIME_DIR +"combine_files";
string CMD_NEW_DIR = "mkdir -m a=rwx -p";


using namespace std;

double RmsdMax = 3;
double TimeMax = 300;

int StrToInt(const string &str) { return atoi(str.c_str()); }
void Copy (const vector<pair<int, pair<double, double> > > vec, vector<pair<int,
pair<double, double> > >& vec_cpy )

{
	for (int i = 0; i < vec.size(); i ++)
		vec_cpy.push_back(vec[i]);
}

void SortByRmsd (vector<pair<int, pair<double, double> > >& vec)
{
	pair<int, pair<double, double> > temp;
	//sort array
	for(int i = 0; i < vec.size(); i++)
	{
		for (int j = 0; j < vec.size()-1; j++)
	 {
	     if (vec[j].second.first > vec[j+1].second.first)
			{
				temp = vec[j];
				vec[j] = vec[j+1];
				vec[j+1] = temp;
			}
	 }/*End inner for loop*/
	}/*End outer for loop*/
}

void SortByTime (vector<pair<int, pair<double, double> > >& vec)
{
	pair<int, pair<double, double> > temp;
	//sort array
	for(int i = 0; i < vec.size(); i++)
	{
		for (int j = 0; j < vec.size()-1; j++)
	 {
	     if (vec[j].second.second > vec[j+1].second.second)
			{
				temp = vec[j];
				vec[j] = vec[j+1];
				vec[j+1] = temp;
			}
	 }/*End inner for loop*/
	}/*End outer for loop*/
}

void FindBestCandidates(
	const vector<pair<int, pair<double, double> > > vec_rmsd,
	const vector<pair<int, pair<double, double> > > vec_time,
	vector<pair<int, pair<double, double> > >& vec_canditates)
{


	int count_candidates = 0;
	for(int i = 0; i < vec_rmsd.size()/3 && count_candidates < 6; i++)
	{


		for(int j = 0; j < vec_time.size()/3 ; j++)
		{


			if (vec_rmsd[i].first == vec_time[j].first )
			{
				count_candidates ++;
				vec_canditates.push_back(vec_time[j]);

			}
		}
	}
}
```

```
void ReplaceSubStr(string fn, string from, string to)
{
       ifstream file (fn.c_str());
       if(!file){cout << "cannot open file from ReplaceSubStr()\n"; exit (1);}
       string line;
       vector<string> line_vec;
       while (getline (file,line))
       {
              line.replace(line.find(from), from.size(), to);
              line_vec.push_back(line);
       }
       file.close();
       ofstream new_file(fn.c_str());
       for(int i = 0; i < line_vec.size(); i ++)
              new_file << line_vec[i]<< endl;

       new_file.close();

}

void Output_file_names (ofstream& file, vector<string> vec, vector<pair<int, pair<double,
double> > > vec_sorted)
{
       int j = 0;

       for (int i= 0; i < 3 && i< vec_sorted.size(); i ++ )
       {
              j = vec_sorted[i].first - 1;
              file << vec[j] << endl;
       }
}

void Copy_Files(string file_with_fn, string to_dir )
// file names in file_with_fn must be with full-path-names

{
       ifstream file(file_with_fn.c_str());
       if(!file){cout << "cannot open file_with_fn from Copy_Files()\n"; exit (1);}

       string line;
       while (getline (file, line)){
              string cmd = "cp " + line + " " + to_dir;
              system(cmd.c_str());
       }
    file.close();
}

void DeleteExtraSpaceInDatFiles(string dir, string file_with_names_fn)
{
       ifstream file_with_names (file_with_names_fn.c_str());
       if(!file_with_names){cout << "cannot open " << file_with_names_fn << endl; exit
(1);}
       string line, line2;
       vector<string> files_vec;

       while (getline (file_with_names,line))
       {
              string name = dir + "/" + line;
              ifstream dat_file (name.c_str());
              if(!dat_file){cout << "cannot open " << name <<  " from
DeleteExtraSpaceInDatFiles()"  << endl; exit (1);}

              vector<double> x_vec, y_vec, in_vec, out_vec, in_out_vec;
              getline (dat_file,line2);
              while (getline (dat_file,line2))
              {
                     double x,y, in, out, in_out;
                     istringstream ist_data(line2);
                     ist_data >> x; x_vec.push_back(x);
                     ist_data >> y; y_vec.push_back(y);
                     ist_data >> in; in_vec.push_back(in);
                     ist_data >> out; out_vec.push_back(out);
                     ist_data >> in_out; in_out_vec.push_back(in_out);
              }
              dat_file.close();
              if (x_vec.size() != y_vec.size()) {cout << name << ": x_vec.size() !=
y_vec.size()" << endl; exit(1);}
              if (y_vec.size() != in_vec.size()) {cout << name << ": y_vec.size() !=
in_vec.size()" << endl; exit(1);}
              if (in_vec.size() != out_vec.size()) {cout << name << ": in_vec.size() !=
out_vec.size()" << endl; exit(1);}
              if (out_vec.size() != in_out_vec.size()) {cout << name << ": out_vec.size()
!= In_out_vec.size()" << endl; exit(1);}

              ofstream dat_file_new (name.c_str());
              if(!dat_file_new){cout << "cannot open for w-mode" << name <<  " from
DeleteExtraSpaceInDatFiles()"  << endl; exit (1);}
              for(int i = 0; i < x_vec.size() ; i ++)
              {
                     dat_file_new << x_vec[i] << " " << y_vec[i] << " "<< in_vec[i] << "
" << out_vec[i] << " "<< in_out_vec[i] <<endl;
              }
              dat_file_new.close();
       }
}

//1st parameter - Protein names file
//2nd parameter -Time Max for quadrants file name
//3nd parameter - simulation name

//              |      |
//  t           |  II  |  III
//  i           |       |
//  m  TimeMax |______|________
//  e           |  I   |  IV
//              |______|________
//              0     RmsdMax
//                     rmsd
// How to run:
int main (int argc, char *argv[])
{//1
       if (argc != 4) {cout << "wrong number of parameters" << endl; exit (1);}


       ifstream proteins (argv[1]);
       if(!proteins){cout << "cannot open proteins file \n"; exit (1);}
```

```
        ifstream time_limit (argv[2]);
        if(!time_limit){cout << "cannot open time_limit file \n"; exit (1);}

        string line_proteins, line_time, line;
        double number;

        string  results_dir = TRJ_DIR + "/" +  argv[3] + "/" + "Results_"  + argv[3];
        string cmd_dir = CMD_NEW_DIR + " " + results_dir;
        system (cmd_dir.c_str());

        string all_files_for_matlab_fn = results_dir + "/all_files_for_matlab";
        ofstream all_files_for_matlab (all_files_for_matlab_fn.c_str());
        if(!all_files_for_matlab){cout << "all_files_for_matlab file \n"; exit (1);}

        string all_rmsd_time_fn = results_dir + "/all_rmsd_time_files";
        ofstream all_rmsd_time (all_rmsd_time_fn.c_str());
        if(!all_rmsd_time){cout << "rmsd file \n"; exit (1);}

        string all_rmsd_time_short_fn = results_dir + "/all_rmsd_time_files_short";
        ofstream all_rmsd_time_short (all_rmsd_time_short_fn.c_str());
        if(!all_rmsd_time_short){cout << "rmsd file \n"; exit (1);}

        all_files_for_matlab << "all_rmsd_time_files_short" << endl;

        //reads rmsd and time files for proteins from the list
        while (getline (proteins,line_proteins)){//2

                //read proteins list file
                string cmd;

                //read protein name
                string protein_name;
                istringstream ist_protein(line_proteins);
                ist_protein >> protein_name;

                cout << protein_name<< ":" << endl;

                // create name that reflects protein-simulation names

                string FileNameLst = protein_name + "_" + argv[3] + "_lst";//this file and
its name are generated in run_oops.cpp
                string trj_full_dir = TRJ_DIR + "/" +  argv[3] + "/" + protein_name;//
where are trj and dat files

                //create required folders: Results_SimName, Results_SimName/Best;
Results_SimName/I; ... Results_SimName/IV

                string dir_name = results_dir + "/" + protein_name;//new
                //string dir_name = trj_full_dir + "/Results_" + argv[3];//old

                cmd_dir = CMD_NEW_DIR + " " + dir_name + "/Best";
                system (cmd_dir.c_str());
                cmd_dir = CMD_NEW_DIR + " " + dir_name + "/I";
                system (cmd_dir.c_str());
                cmd_dir = CMD_NEW_DIR + " " + dir_name + "/II";
                system (cmd_dir.c_str());
                cmd_dir = CMD_NEW_DIR + " " + dir_name + "/III";
                system (cmd_dir.c_str());
                cmd_dir = CMD_NEW_DIR + " " + dir_name + "/IV";
                system (cmd_dir.c_str());

                //generate file names for output rmsds and times
                string in_rmsd_fn = dir_name + "/" + FileNameLst + "_rmsd";
                string in_time_fn = dir_name + "/" + FileNameLst + "_time";
                //string rmsd_time_fn = dir_name + "/" + FileNameLst + "_rmsd_time";//for
matlab scatter plot//old
                string rmsd_time_fn = results_dir + "/" +  protein_name +
"_rmsd_time";//for matlab scatter plot
                all_rmsd_time << rmsd_time_fn << endl;
                all_rmsd_time_short <<protein_name + "_rmsd_time" << endl;
                //cp file with trajectories names to Result folder, in this case we can
clear folder with jobs otput

                cmd = "cp " + TOOL_DIR + "/src/" + FileNameLst  + " " + dir_name;
                system(cmd.c_str());

                //run combine_files program (TIME)

                cmd = CMD_TIME_NAME + " " + dir_name + "/" + FileNameLst + " " +
in_time_fn;

                //test
                cout << "time cmd = " << cmd <<endl;
                system(cmd.c_str());


                // modify FileNameLst file: replace all "_stat" to "_f.pdb" for rmsd
calculation
                string modify_fn =  dir_name + "/" + FileNameLst;


                ReplaceSubStr(modify_fn, "_stat", "_f.pdb");
                //run RMSD calculation program
                string pdb_file = PDB_DIR + "/" +  protein_name + ".pdb";
                cmd = CMD_RMSD_NAME + " -str " + pdb_file + " -list " + modify_fn ;

                system(cmd.c_str());

                //open rmsd file
                ifstream in_rmsd (in_rmsd_fn.c_str());
                if(!in_rmsd){cout << "rmsd file \n"; exit (1);}

                //open time file
                ifstream in_time (in_time_fn.c_str());
                if(!in_time) {cout << "time file \n"; exit (1);}

                //read time file and fills TimeMax for each protein/rmsdMax and TimaMax
define quadrants
                double time_from_file;
                getline (time_limit,line_time);
                istringstream ist_time(line_time);
                ist_time >> time_from_file;
                TimeMax = time_from_file;

                // read rmsd and time file for each protein
```

```
            vector<double> rmsd_vec;
            vector<double> time_vec;
            //istringstream ist;
            double rmsd;

            while (getline (in_rmsd, line)){
                    istringstream ist(line);
                    ist >> rmsd;
                   // number = atof(word_number.c_str());
                 rmsd_vec.push_back(rmsd);
            }

            in_rmsd.close ();
            double time;

            //test
            cout << "Time vector values:" << endl;
            while (getline (in_time, line)){
                   istringstream ist(line);
                   ist >> time;
                   cout << time << endl;
                   //number = atof(word_number.c_str());
                   time_vec.push_back(time);
            }
            in_time.close ();

            if(rmsd_vec.size() != time_vec.size() ){
                   cout << "rmsd vector size (" <<rmsd_vec.size()<< ")" << " != time
vector size (" <<time_vec.size()<< ")"<< endl;
                   exit(1);
            }

            ofstream rmsd_time (rmsd_time_fn.c_str());
            if (!rmsd_time) {cout << "cannot open rmsd_time file" << endl; exit(1);}
            for (int i = 0; i < rmsd_vec.size(); i++ )
            {
                   rmsd_time <<  rmsd_vec[i] << " " <<  time_vec[i] << endl;
            }


            // sort to quadrants
            vector<pair<int, pair<double, double> > > I, II, III, IV, All;
            int size = rmsd_vec.size();

            for (int i = 0; i < size; i ++ )
            {//3
                   pair<double, double> rmsd_time;
                   pair <int, pair <double, double> > index__rmsd_time;

                rmsd_time.first = rmsd_vec[i];
                rmsd_time.second = time_vec[i];
                index__rmsd_time.first = i + 1;
                   index__rmsd_time.second =  rmsd_time;

                   // list with a triad (trj number, rmsd, time)
                   All.push_back(index__rmsd_time);

                   if ((rmsd_vec[i] <= RmsdMax) && (time_vec[i] <= TimeMax))
```

```
                            I.push_back(index__rmsd_time);
                     if ((rmsd_vec[i] <= RmsdMax) && (time_vec[i] > TimeMax))
                            II.push_back(index__rmsd_time);
                     if ((rmsd_vec[i] > RmsdMax) && (time_vec[i] > TimeMax))
                            III.push_back(index__rmsd_time);
                     if ((rmsd_vec[i] > RmsdMax) && (time_vec[i] <= TimeMax))
                            IV.push_back(index__rmsd_time);

              }//_3

           //copy
              vector<pair<int, pair<double, double> > > I_t, II_t, III_t, IV_t, All_t;
              Copy(All, All_t);Copy(I, I_t); Copy(II, II_t); Copy(III, III_t); Copy(IV,
IV_t);
              cout <<"All_t.size() =  " << All_t.size() << endl;
              cout <<"I_t.size() =  " << I_t.size() << endl;
              cout <<"II_t.size() =  " << II_t.size() << endl;
              cout <<"III_t.size() =  " << III_t.size() << endl;
              cout <<"IV_t.size() =  " << IV_t.size() << endl;

              //sort
               SortByRmsd(All); SortByRmsd(I);  SortByRmsd(II);  SortByRmsd(III);
SortByRmsd(IV);
               SortByTime(All_t); SortByTime(I_t);  SortByTime(II_t);  SortByTime(III_t);
SortByTime(IV_t);


               //Find the best candidates
               vector<pair<int, pair<double, double> > > best_candidates;
               FindBestCandidates(All, All_t, best_candidates);

               //print all results sorted by rmsd to files
              string out_all_fn = dir_name + "/" + FileNameLst + "_all";
              ofstream out_all (out_all_fn.c_str());
              if(!out_all){cout << "all info file can't be open\n"; exit (1);}

              string out_best_fn = dir_name + "/" +  FileNameLst +"_best";
              ofstream out_best_candidates (out_best_fn.c_str());
              if(!out_best_candidates){cout << "best_candidates file can't be open\n";
exit (1);}


              out_best_candidates << "The best candidates (" <<best_candidates.size() <<
")"<< endl;
              for (int i = 0; i< best_candidates.size(); i ++ )
                     out_best_candidates << best_candidates[i].first << ":  (" <<
best_candidates[i].second.first << ", " <<  best_candidates[i].second.second << ")" <<
endl;

              out_all << "// Sorted by RMSD" << endl;


              out_all << "// I quadrant (" <<I.size() << ")"<< endl;
              for (int i = 0; i< I.size(); i ++ )
                     out_all << I[i].first << ":  (" <<  I[i].second.first << ", " <<
I[i].second.second << ")" << endl;
```

```
              out_all << "// II quadrant (" <<II.size() << ")"<< endl;
              for (int i = 0; i< II.size(); i ++ )
                     out_all << II[i].first << ":  (" <<  II[i].second.first << ", " <<
II[i].second.second << ")" << endl;


              out_all << "// III quadrant (" <<III.size() << ")"<< endl;
              for (int i = 0; i< III.size(); i ++ )
                     out_all << III[i].first << ":  (" <<  III[i].second.first << ", " <<
III[i].second.second << ")" << endl;


              out_all << "// IV quadrant (" <<IV.size() << ")"<< endl;
              for (int i = 0; i< IV.size(); i ++ )
                     out_all << IV[i].first << ":  (" <<  IV[i].second.first << ", " <<
IV[i].second.second << ")" << endl;
              out_best_candidates.close();
              out_all.close();


       string trj_fn =  dir_name + "/" + FileNameLst;
       //put .dat files to vector<string>
       vector<string> final_pdb_files_names;
       vector<string> dat_files_names;
       vector<string> dat_files_names_without_dir;
       ifstream trj_files (trj_fn.c_str());
       if(!trj_files){cout << "cannot open file with trj_f names\n"; exit (1);}
       while (getline (trj_files, line)){
              final_pdb_files_names.push_back(line);
              line.replace(line.find("_f.pdb"), 6, ".dat");
              dat_files_names.push_back(line);

              size_t line_len = line.size();
              string find_str = "/"+ protein_name + "/";
              line.erase(0, line.find (find_str)+protein_name.size()+2);
              dat_files_names_without_dir.push_back(line);
       }
       trj_files.close();


       //We need to create foders in Results:
       //1.SimName_best:
       //      _f.pdb//later
       //   .dat
       //2. SimName_I/II/III/IV (max 3trj in each category)
       //      _f.pdb//later
       //   .dat

       //these files contain lists with file names required for matlab analysis

       //AND COPY DAT FILES TO RESULT , f.pdb and trj
       //
       string matlab_out_best_energy_profile_fn =dir_name + "/Best/" + FileNameLst +
"_best_energy_profile_matlab";
       ofstream matlab_out_best_energy_profile
(matlab_out_best_energy_profile_fn.c_str());
       if(!matlab_out_best_energy_profile){cout <<
"best_matlab_out_best_energy_profile_matlab info file can't be open\n"; exit (1);}
```

```
	string matlab_out_best_energy_profile_fn_short =dir_name + "/Best/" + FileNameLst
+ "_best_energy_profile_matlab_short";
	ofstream matlab_out_best_energy_profile_short
(matlab_out_best_energy_profile_fn_short.c_str());
	if(!matlab_out_best_energy_profile_short){cout <<
"best_matlab_out_best_energy_profile_matlab_short info file can't be open\n"; exit (1);}
	all_files_for_matlab << protein_name + "/Best/" + FileNameLst +
"_best_energy_profile_matlab_short" << endl;

	Output_file_names (matlab_out_best_energy_profile, dat_files_names,
best_candidates);
	Output_file_names (matlab_out_best_energy_profile_short,
dat_files_names_without_dir, best_candidates);

	matlab_out_best_energy_profile.close();
	matlab_out_best_energy_profile_short.close();
	Copy_Files(matlab_out_best_energy_profile_fn, dir_name + "/Best/" );
	DeleteExtraSpaceInDatFiles(dir_name + "/Best",
matlab_out_best_energy_profile_fn_short);
	//
	string matlab_out_I_best_energy_profile_fn = dir_name + "/I/" + FileNameLst +
"_I_energy_profile_matlab";
	ofstream matlab_out_I_best_energy_profile
(matlab_out_I_best_energy_profile_fn.c_str());
	if(!matlab_out_I_best_energy_profile){cout << "I_energy_profile_matlab info file
can't be open\n"; exit (1);}

	string matlab_out_I_best_energy_profile_fn_short = dir_name + "/I/" + FileNameLst
+ "_I_energy_profile_matlab_short";
	ofstream matlab_out_I_best_energy_profile_short
(matlab_out_I_best_energy_profile_fn_short.c_str());
	if(!matlab_out_I_best_energy_profile_short){cout << "I_energy_profile_matlab_short
info file can't be open\n"; exit (1);}
	all_files_for_matlab << protein_name + "/I/" + FileNameLst +
"_I_energy_profile_matlab_short" << endl;

	Output_file_names (matlab_out_I_best_energy_profile, dat_files_names, I);
	Output_file_names (matlab_out_I_best_energy_profile_short,
dat_files_names_without_dir, I);

	matlab_out_I_best_energy_profile.close();
	matlab_out_I_best_energy_profile_short.close();
	Copy_Files(matlab_out_I_best_energy_profile_fn, dir_name + "/I/" );
	DeleteExtraSpaceInDatFiles(dir_name + "/I",
matlab_out_I_best_energy_profile_fn_short);
	//
	string matlab_out_II_best_energy_profile_fn = dir_name + "/II/" + FileNameLst +
"_II_energy_profile_matlab";
	ofstream matlab_out_II_best_energy_profile
(matlab_out_II_best_energy_profile_fn.c_str());
	if(!matlab_out_II_best_energy_profile){cout <<
"II_best_matlab_out_best_energy_profile_matlab info file can't be open\n"; exit (1);}

	string matlab_out_II_best_energy_profile_fn_short = dir_name + "/II/" +
FileNameLst + "_II_energy_profile_matlab_short";
	ofstream matlab_out_II_best_energy_profile_short
(matlab_out_II_best_energy_profile_fn_short.c_str());
```

```
	if(!matlab_out_II_best_energy_profile_short){cout << 
"II_best_matlab_out_best_energy_profile_matlab_short info file can't be open\n"; exit 
(1);}
	all_files_for_matlab << protein_name + "/II/" + FileNameLst + 
"_II_energy_profile_matlab_short" << endl;

	Output_file_names (matlab_out_II_best_energy_profile, dat_files_names, II);
	Output_file_names (matlab_out_II_best_energy_profile_short, 
dat_files_names_without_dir, II);

	matlab_out_II_best_energy_profile.close();
	matlab_out_II_best_energy_profile_short.close();
	Copy_Files(matlab_out_II_best_energy_profile_fn, dir_name + "/II/" );
	DeleteExtraSpaceInDatFiles(dir_name + "/II", 
matlab_out_II_best_energy_profile_fn_short);
	//
	string matlab_out_III_best_energy_profile_fn = dir_name + "/III/" +  FileNameLst + 
"_III_energy_profile_matlab";
	ofstream matlab_out_III_best_energy_profile 
(matlab_out_III_best_energy_profile_fn.c_str());
	if(!matlab_out_III_best_energy_profile){cout << 
"III_best_matlab_out_best_energy_profile_matlab info file can't be open\n"; exit (1);}

	string matlab_out_III_best_energy_profile_fn_short = dir_name + "/III/" + 
FileNameLst + "_III_energy_profile_matlab_short";
	ofstream matlab_out_III_best_energy_profile_short 
(matlab_out_III_best_energy_profile_fn_short.c_str());
	if(!matlab_out_III_best_energy_profile_short){cout << 
"III_best_matlab_out_best_energy_profile_matlab_short info file can't be open\n"; exit 
(1);}
	all_files_for_matlab << protein_name + "/III/" + FileNameLst + 
"_III_energy_profile_matlab_short" << endl;

	Output_file_names (matlab_out_III_best_energy_profile, dat_files_names, III);
	Output_file_names (matlab_out_III_best_energy_profile_short, 
dat_files_names_without_dir, III);

	matlab_out_III_best_energy_profile.close();
	matlab_out_III_best_energy_profile_short.close();
	Copy_Files(matlab_out_III_best_energy_profile_fn, dir_name + "/III" );
	DeleteExtraSpaceInDatFiles(dir_name + "/III", 
matlab_out_III_best_energy_profile_fn_short);
	//
	string matlab_out_IV_best_energy_profile_fn = dir_name + "/IV/" + FileNameLst + 
"_IV_energy_profile_matlab";
	ofstream matlab_out_IV_best_energy_profile 
(matlab_out_IV_best_energy_profile_fn.c_str());
	if(!matlab_out_IV_best_energy_profile){cout << 
"IV_best_matlab_out_best_energy_profile_matlab info file can't be open\n"; exit (1);}

	string matlab_out_IV_best_energy_profile_fn_short = dir_name + "/IV/" + 
FileNameLst + "_IV_energy_profile_matlab_short";
	ofstream matlab_out_IV_best_energy_profile_short 
(matlab_out_IV_best_energy_profile_fn_short.c_str());
	if(!matlab_out_IV_best_energy_profile_short){cout << 
"IV_best_matlab_out_best_energy_profile_matlab_short info file can't be open\n"; exit 
(1);}
```

```
       all_files_for_matlab << protein_name + "/IV/" + FileNameLst +
"_IV_energy_profile_matlab_short" << endl;

       Output_file_names (matlab_out_IV_best_energy_profile, dat_files_names, IV);
       Output_file_names (matlab_out_IV_best_energy_profile_short,
dat_files_names_without_dir, IV);

       matlab_out_IV_best_energy_profile.close();
       matlab_out_IV_best_energy_profile_short.close();
       Copy_Files(matlab_out_IV_best_energy_profile_fn, dir_name + "/IV" );
       DeleteExtraSpaceInDatFiles(dir_name + "/IV",
matlab_out_IV_best_energy_profile_fn_short);
       }//_2
       all_rmsd_time.close();
       all_rmsd_time_short.close();
       all_files_for_matlab.close();
}//_1
```

MATLAB programs to visualize data

To create scatter plots (rmsd vs time) and energy profiles for successful trajectories the following MATLAB program should be run:

CreateScatterPlots_new('C:\Users\sasha\Desktop\TEMP_WORK', 'newprot-10', 'all_files_for_matlab', 'Test_Time_lst', 'Energy_LST')

Where the first parameter is the name of a directory where Results_SimulationName are located (sort_files command). The second parameter is a simulation name to be analyzed, the third parameter is the name of the file in Results_SimulationName folder, the forth parameter is a file with expected folding times for proteins, and the last parameter is a file with energy terms names used for simulations. Files Test_Time_lst, Energy_LST must be located in the folder that was used for the first parameter.

```
function CreateScatterPlots_new( dir, sim_name, file, time_file_name,
energy_file_name )
% HOW to run:
%CreateScatterPlots('C:\Users\Aleksandra Safronova\Desktop\oops\Matlab',
%                     'TestInertia2', 'all_files_for_matlab',
'Test_Time_lst')
% file Test_Time_lst must be in dir
names_to_join = {'Results_', sim_name};
joined_name = strjoin(names_to_join, '');
%joined_name = [names_to_join{:}];

full_name = {dir, joined_name, file};
full_name_str = strjoin(full_name, '\');
%full_name = {dir,'\' ,joined_name,'\' ,file};
%full_name_str = [full_name{:}];

L = readtable(full_name_str,'ReadVariableNames',false,'Format', '%s');
MAT_NAMES = L.Var1;

full_time_file_name = {dir,time_file_name};
full_time_file_name_str = strjoin(full_time_file_name, '\');

full_energy_file_name = {dir,energy_file_name};
full_energy_file_name_str = strjoin(full_energy_file_name, '\');
```

```
%full_time_file_name = {dir,'\',time_file_name};
%full_time_file_name_str = [full_time_file_name{:}];

%full_energy_file_name = {dir,'\',energy_file_name};
%full_energy_file_name_str = [full_energy_file_name{:}];

%generates scatter plots
scatter_files = {dir, joined_name, MAT_NAMES{1}};
full_scatter_files =  strjoin(scatter_files, '\');
%scatter_files = {dir,'\',joined_name, MAT_NAMES{1}};
%full_scatter_files =  [scatter_files{:}];
L = readtable(full_scatter_files,'ReadVariableNames',false,'Format', '%s');
NAMES = L.Var1;
len = length(NAMES);

L = readtable(full_time_file_name_str,'ReadVariableNames',false,'Format',
'%f%s');
TIMES = L.Var1;

for n = 1:len
    C = {dir, joined_name,  NAMES{n}};
    name = strjoin(C,'\');
    %C = {dir,'\' ,joined_name,'\' , NAMES{n}};
    %name = [C{:}];
    %CallScatter( name, sim_name,TIMES(n));
    CallScatter( name, sim_name);
end

%generates energy profiles and compactness
len = length(MAT_NAMES);
for n = 2:len
   %for windows only
    C = strsplit (MAT_NAMES{n},'/');
    %//C_size = length(C);
    %//str_file = {C{1},C{2},C{3}};
    str_file2 = strjoin(C, '\');
    %str_file2 = strjoin(C, '\');
    C = {dir, joined_name, str_file2};
    %//C = {dir, joined_name,  MAT_NAMES{n}};
    name = strjoin(C,'\');
    CallEnergyProfile_new( dir, joined_name, name,
full_energy_file_name_str);
end
end




function CallScatter( name, sim_name )

T = readtable(name,'Delimiter','space','ReadVariableNames',false, 'Format',
'%f%f');
RMSD =T.Var1;
TIME = T.Var2;
```

```
h=figure('visible','off');
scatter(RMSD, TIME);
axis([0,12, 0, inf]);

h2=xlabel('rmsd');
set(h2, 'FontSize', 16);
set(gca,'XTick',[0:1:12]);
set(gca,'XTickLabel',{'0','', '2','', '4','', '6','', '8','', '10','',
'12'});

h3 = ylabel('time in sec');
set(h3, 'FontSize', 16);
C = strsplit (name,'\');
C_size = length(C);
str_file = C{C_size};
str_file2 = strrep(str_file, '_', ' ');
C = {sim_name, str_file2};
str_file3 = strjoin(C, ' ');

h4 = title(str_file3);
set(h4, 'FontSize', 18) ;
set(h4,'FontWeight','bold');

%set(h, 'Color', 'white'); % white bckgr
%export_fig( h, ...      % figure handle
%    name,... % name of output file without extension
%    '-painters', ...      % renderer
%    '-jpg', ...           % file format
%    '-r150' );             % resolution in dpi
%savefig(h,name);
saveas(h,name,'fig');
saveas(h,name,'jpg');

close(h);

end



function CallEnergyProfile_new( dir, joined_name,name,energies_file_name )
T = readtable(name,'ReadVariableNames',false,'Format', '%s');
C = strsplit (name,'\');
C_size = length(C);
prot_cat = {C{C_size-2}, C{C_size-1} };
prot_cat_str = strjoin(prot_cat, '\');

DAT_NAMES = T.Var1;

len = length(DAT_NAMES);
for n = 1:len
    full_name = {dir, joined_name, prot_cat_str, DAT_NAMES{n} };
    full_name_str = strjoin(full_name, '\');
    CallEnergyProfilePlot_new( full_name_str,energies_file_name);
end
end
```

```
unction CallEnergyProfilePlot( name, energies_file_name )

L = readtable(energies_file_name,'ReadVariableNames',false,'Format', '%s');
ENERGIES = L.Var1;

T = readtable(name,'ReadVariableNames',false,'Delimiter','space','Format',
'%f%f%f%f%f');
X =T.Var1;
Y = T.Var2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h=figure('visible','off');
plot(X, Y);
axis([0,inf, -inf, inf]);

h_1=xlabel('MC steps');
set(h_1, 'FontSize', 16);

h_2 = ylabel('energy');
set(h_2, 'FontSize', 16);

C = strsplit (name,'\');
C_size = length(C);
str_file = C{C_size};
str_file2 = strrep(str_file, '_', ' ');
CC = {C{C_size - 1}, str_file2};
title_name = strjoin(CC, ' ');

h_3 = title(title_name);
set(h_3, 'FontSize', 18) ;
set(h_3,'FontWeight','bold');
file_name = name;
energy_file_name = strrep(file_name, '.dat', '_energy');

saveas(h,energy_file_name,'fig');
saveas(h,energy_file_name,'jpg');
close(h);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for n = 1:length(ENERGIES)
energy_term =T.(n+2);
h2=figure('visible','off');
plot(X, energy_term);
%hold all;
axis([0,inf, -inf, inf]);

h2_1=xlabel('MC steps');
set(h2_1, 'FontSize', 16);
h2_2 = ylabel('Energy');
set(h2_2, 'FontSize', 16);
```

```
CCC = {ENERGIES{n}, title_name};
energy_title = strjoin(CCC, ' ');
h2_3 = title(energy_title);
set(h2_3, 'FontSize', 18) ;
set(h2_3,'FontWeight','bold');

en_file_name = strrep(file_name, '.dat', ENERGIES{n});
saveas(h2,en_file_name,'fig');
saveas(h2,en_file_name,'jpg');
close(h2);
end
end
```

Bibliography

1. **Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, and Lipman DJ**. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res* 25: 3389-3402, 1997.
2. **Anfinsen CB**. Principles that govern the folding of protein chains. *Science* 181: 223-230, 1973.
3. **Bairoch A AR, Wu CH, et al**. The Universal Protein Resource [UniProt]. *Nucleic Acids Res* D154-159, 2005.
4. **Berman HM, Bhat TN, Bourne PE, Feng Z, Gilliland G, Weissig H, and Westbrook J**. The Protein Data Bank and the challenge of structural genomics. *Nature structural biology* 7 Suppl: 957-959, 2000.
5. **Binder KH, D. W**. Monte Carlo Simulation in Statistical Physics. *2nd edit, Springer- Verlag, Berlin* 1992.
6. **Bonneau R, Strauss CE, Rohl CA, Chivian D, Bradley P, Malmstrom L, Robertson T, and Baker D**. De novo prediction of three-dimensional structures for major protein families. *Journal of molecular biology* 322: 65-78, 2002.
7. **Bonneau R, Tsai J, Ruczinski I, Chivian D, Rohl C, Strauss CE, and Baker D**. Rosetta in CASP4: progress in ab initio protein structure prediction. *Proteins* Suppl 5: 119-126, 2001.
8. **Bower MJ, Cohen FE, and Dunbrack RL, Jr.** Prediction of protein side-chain rotamers from a backbone-dependent rotamer library: a new homology modeling tool. *Journal of molecular biology* 267: 1268-1282, 1997.
9. **Bowie JU, and Eisenberg D**. An evolutionary approach to folding small alpha-helical proteins that uses sequence information and an empirical guiding fitness function. *Proceedings of the National Academy of Sciences of the United States of America* 91: 4436-4440, 1994.
10. **Bowie JU, Luthy R, and Eisenberg D**. A method to identify protein sequences that fold into a known three-dimensional structure. *Science* 253: 164-170, 1991.
11. **Bradley P, and Baker D**. Improved beta-protein structure prediction by multilevel optimization of nonlocal strand pairings and local backbone conformation. *Proteins* 65: 922-929, 2006.
12. **Bradley P, Misura KM, and Baker D**. Toward high-resolution de novo structure prediction for small proteins. *Science* 309: 1868-1871, 2005.
13. **Brooks BR BR, Olafson BD, et al.** CHARMM: a program for macromolecular energy, minimization, and dynamics calculations. *J Comput Chem* 4: 187–217, 1983.
14. **Brylinski M, Konieczny L, and Roterman I**. Hydrophobic collapse in (in silico) protein folding. *Computational biology and chemistry* 30: 255-267, 2006.
15. **Canutescu AA, Shelenkov AA, and Dunbrack RL, Jr.** A graph-theory algorithm for rapid protein side-chain prediction. *Protein science : a publication of the Protein Society* 12: 2001-2014, 2003.
16. **Case DA PD, Caldwell JA, et al.** AMBER 5.0. *University of California, San Francisco, CA* 1997.
17. **Chen VB, Arendall WB, 3rd, Headd JJ, Keedy DA, Immormino RM, Kapral GJ, Murray LW, Richardson JS, and Richardson DC**. MolProbity: all-atom structure validation for macromolecular crystallography. *Acta crystallographica Section D, Biological crystallography* 66: 12-21, 2010.
18. **Chivian D, Kim DE, Malmstrom L, Schonbrun J, Rohl CA, and Baker D**. Prediction of CASP6 structures using automated Robetta protocols. *Proteins* 61 Suppl 7: 157-166, 2005.
19. **Clementi C**. Coarse-grained models of protein folding: toy models or predictive tools? *Current opinion in structural biology* 18: 10-15, 2008.
20. **Clementi C, Nymeyer H, and Onuchic JN**. Topological and energetic factors: what determines the structural details of the transition state ensemble and "en-route" intermediates for protein folding? An investigation for small globular proteins. *Journal of molecular biology* 298: 937-953, 2000.
21. **Colubri A**. OOPS software. *http://protlibuchicagoedu/oopshtml*

22. **Cong Q, Kinch LN, Pei J, Shi S, Grishin VN, Li W, and Grishin NV**. An automatic method for CASP9 free modeling structure prediction assessment. *Bioinformatics (Oxford, England)* 27: 3371-3378, 2011.

23. **Creighton TE**. Protein Folding. *1st edit, WH Freeman and Company, New York* 1992.

24. **Crippen GM**. Easily searched protein folding potentials. *Journal of molecular biology* 260: 467-475, 1996.

25. **Das R, Qian B, Raman S, Vernon R, Thompson J, Bradley P, Khare S, Tyka MD, Bhat D, Chivian D, Kim DE, Sheffler WH, Malmstrom L, Wollacott AM, Wang C, Andre I, and Baker D**. Structure prediction for CASP7 targets using extensive all-atom refinement with Rosetta@home. *Proteins* 69 Suppl 8: 118-128, 2007.

26. **Dill KA**. Additivity principles in biochemistry. *The Journal of biological chemistry* 272: 701-704, 1997.

27. **Dill KA**. Dominant forces in protein folding. *Biochemistry* 29: 7133-7155, 1990.

28. **Dill KA, and Chan HS**. From Levinthal to pathways to funnels. *Nature structural biology* 4: 10-19, 1997.

29. **Dinner AR, Sali A, Smith LJ, Dobson CM, and Karplus M**. Understanding protein folding via free-energy surfaces from theory and experiment. *Trends in biochemical sciences* 25: 331-339, 2000.

30. **Earl DJ, and Deem MW**. Parallel tempering: theory, applications, and new perspectives. *Physical chemistry chemical physics : PCCP* 7: 3910-3916, 2005.

31. **Faraggi E, Xue B, and Zhou Y**. Improving the prediction accuracy of residue solvent accessibility and real-value backbone torsion angles of proteins by guided-learning through a two-layer neural network. *Proteins* 74: 847-856, 2009.

32. **Go N, and Abe H**. Noninteracting local-structure model of folding and unfolding transition in globular proteins. I. Formulation. *Biopolymers* 20: 991-1011, 1981.

33. **Goldstein RA, Luthey-Schulten ZA, and Wolynes PG**. Optimal protein-folding codes from spin-glass theory. *Proceedings of the National Academy of Sciences of the United States of America* 89: 4918-4922, 1992.

34. **Gront D, Blaszczyk M, Wojciechowski P, and Kolinski A**. BioShell Threader: protein homology detection based on sequence profiles and secondary structure profiles. *Nucleic Acids Res* 40: W257-262, 2012.

35. **Gu H, Kim D, and Baker D**. Contrasting roles for symmetrically disposed beta-turns in the folding of a small protein. *Journal of molecular biology* 274: 588-596, 1997.

36. **Gu H, Yi Q, Bray ST, Riddle DS, Shiau AK, and Baker D**. A phage display system for studying the sequence determinants of protein folding. *Protein science : a publication of the Protein Society* 4: 1108-1117, 1995.

37. **Hayes B**. Prototeins. *American Scientist* 86: 216, 1998.

38. **Head-Gordon T, and Brown S**. Minimalist models for protein folding and design. *Current opinion in structural biology* 13: 160-167, 2003.

39. **Huang YM, and Bystroff C**. Expanded explorations into the optimization of an energy function for protein design. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM* 10: 1176-1187, 2013.

40. **J.Tooze CBa**. Introduction to protein structure. *New York Garland Publishing* 1999.

41. **Kallberg M, Wang H, Wang S, Peng J, Wang Z, Lu H, and Xu J**. Template-based protein structure modeling using the RaptorX web server. *Nature protocols* 7: 1511-1522, 2012.

42. **Kamisetty H, Ovchinnikov S, and Baker D**. Assessing the utility of coevolution-based residue-residue contact predictions in a sequence- and structure-rich era. *Proceedings of the National Academy of Sciences of the United States of America* 110: 15674-15679, 2013.

43. **Kinch L, Yong Shi S, Cong Q, Cheng H, Liao Y, and Grishin NV**. CASP9 assessment of free modeling target predictions. *Proteins* 79 Suppl 10: 59-73, 2011.
44. **Kinch LN, Li W, Monastyrskyy B, Kryshtafovych A, and Grishin NV**. Evaluation of free modeling targets in CASP11 and ROLL. *Proteins* 2015.
45. **Klepeis JL, Wei Y, Hecht MH, and Floudas CA**. Ab initio prediction of the three-dimensional structure of a de novo designed protein: a double-blind case study. *Proteins* 58: 560-570, 2005.
46. **Klimov DK, and Thirumalai D**. Factors governing the foldability of proteins. *Proteins* 26: 411-441, 1996.
47. **Kolinski A, and Skolnick J**. Assembly of protein structure from sparse experimental data: an efficient Monte Carlo model. *Proteins* 32: 475-494, 1998.
48. **Korst EAJ**. Simulated annealing and Boltzmann Machines. *Wiley & Sons* 1989.
49. **Kortemme T, Morozov AV, and Baker D**. An orientation-dependent hydrogen bonding potential improves prediction of specificity and structure for proteins and protein-protein complexes. *Journal of molecular biology* 326: 1239-1259, 2003.
50. **Lazaridis T, and Karplus M**. Effective energy function for proteins in solution. *Proteins* 35: 133-152, 1999.
51. **Leaver-Fay A, O'Meara MJ, Tyka M, Jacak R, Song Y, Kellogg EH, Thompson J, Davis IW, Pache RA, Lyskov S, Gray JJ, Kortemme T, Richardson JS, Havranek JJ, Snoeyink J, Baker D, and Kuhlman B**. Scientific benchmarks for guiding macromolecular energy function improvement. *Methods in enzymology* 523: 109-143, 2013.
52. **Levinthal C**. *in Proceedings of a Meeting Held at Allerton House, Monticello, IL (Univ of Illinois Press, Urbana)* 1969.
53. **Lindahl E HB, van der Spoel D** GROMACS 3.0: a package for molecular simulation and trajectory analysis. *J Mol Model* 7: 306–317, 2001.
54. **Liwo A, Arlukowicz P, Czaplewski C, Oldziej S, Pillardy J, and Scheraga HA**. A method for optimizing potential-energy functions by a hierarchical design of the potential-energy landscape: application to the UNRES force field. *Proceedings of the National Academy of Sciences of the United States of America* 99: 1937-1942, 2002.
55. **Liwo A, Khalili M, and Scheraga HA**. Ab initio simulations of protein-folding pathways by molecular dynamics with the united-residue model of polypeptide chains. *Proceedings of the National Academy of Sciences of the United States of America* 102: 2362-2367, 2005.
56. **Ma J, Peng J, Wang S, and Xu J**. A conditional neural fields model for protein threading. *Bioinformatics (Oxford, England)* 28: i59-66, 2012.
57. **MacKerell AD, Bashford D, Bellott M, Dunbrack RL, Evanseck JD, Field MJ, Fischer S, Gao J, Guo H, Ha S, Joseph-McCarthy D, Kuchnir L, Kuczera K, Lau FT, Mattos C, Michnick S, Ngo T, Nguyen DT, Prodhom B, Reiher WE, Roux B, Schlenkrich M, Smith JC, Stote R, Straub J, Watanabe M, Wiorkiewicz-Kuczera J, Yin D, and Karplus M**. All-atom empirical potential for molecular modeling and dynamics studies of proteins. *The journal of physical chemistry B* 102: 3586-3616, 1998.
58. **Maiorov VN, and Crippen GM**. Contact potential that recognizes the correct folding of globular proteins. *Journal of molecular biology* 227: 876-888, 1992.
59. **Mariani V, Biasini M, Barbato A, and Schwede T**. lDDT: a local superposition-free score for comparing protein structures and models using distance difference tests. *Bioinformatics (Oxford, England)* 29: 2722-2728, 2013.
60. **Max N, Hu C, Kreylos O, and Crivelli S**. BuildBeta--a system for automatically constructing beta sheets. *Proteins* 78: 559-574, 2010.
61. **Metropolis N, Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E**. Equation of State Calculations by Fast Computing Machines. *J Chem Phys* 21: 1087, 1953.

62. **Mirny LA, Abkevich VI, and Shakhnovich EI**. How evolution makes proteins fold quickly. *Proceedings of the National Academy of Sciences of the United States of America* 95: 4976-4981, 1998.
63. **Mirny LA, and Shakhnovich EI**. How to derive a protein folding potential? A new approach to an old problem. *Journal of molecular biology* 264: 1164-1179, 1996.
64. **Moult J, Fidelis K, Zemla A, and Hubbard T**. Critical assessment of methods of protein structure prediction (CASP)-round V. *Proteins* 53 Suppl 6: 334-339, 2003.
65. **Murzin AG, Brenner SE, Hubbard T, and Chothia C**. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of molecular biology* 247: 536-540, 1995.
66. **Nauli S, Kuhlman B, and Baker D**. Computer-based redesign of a protein folding pathway. *Nature structural biology* 8: 602-605, 2001.
67. **Onuchic JN, Luthey-Schulten Z, and Wolynes PG**. Theory of protein folding: the energy landscape perspective. *Annual review of physical chemistry* 48: 545-600, 1997.
68. **Ovchinnikov S, Kamisetty H, and Baker D**. Robust and accurate prediction of residue-residue interactions across protein interfaces using evolutionary information. *eLife* 3: e02030, 2014.
69. **Ovchinnikov S, Kim DE, Wang RY, Liu Y, DiMaio F, and Baker D**. Improved de novo structure prediction in CASP11 by incorporating Co-evolution information into rosetta. *Proteins* 2015.
70. **Pande VS, and Rokhsar DS**. Folding pathway of a lattice model for proteins. *Proceedings of the National Academy of Sciences of the United States of America* 96: 1273-1278, 1999.
71. **Park SH, O'Neil KT, and Roder H**. An early intermediate in the folding reaction of the B1 domain of protein G contains a native-like core. *Biochemistry* 36: 14277-14283, 1997.
72. **Pertsemlidis A, Zelinka J, Fondon JW, 3rd, Henderson RK, and Otwinowski Z**. Bayesian statistical studies of the Ramachandran distribution. *Statistical applications in genetics and molecular biology* 4: Article35, 2005.
73. **Pierluigi Crescenzi DG, Christos H. Papadimitriou, Antonio Piccolboni, and Mihalis Yannakakis**. On the Complexity of Protein Folding. *Journal of Computational Biology* 5: 423-466, 1998.
74. **Plaxco KW, Millett IS, Segel DJ, Doniach S, and Baker D**. Chain collapse can occur concomitantly with the rate-limiting step in protein folding. *Nature structural biology* 6: 554-556, 1999.
75. **Plaxco KW, Simons KT, and Baker D**. Contact order, transition state placement and the refolding rates of single domain proteins. *Journal of molecular biology* 277: 985-994, 1998.
76. **Ponder JW, and Case DA**. Force fields for protein simulations. *Advances in protein chemistry* 66: 27-85, 2003.
77. **Raman S, Vernon R, Thompson J, Tyka M, Sadreyev R, Pei J, Kim D, Kellogg E, DiMaio F, Lange O, Kinch L, Sheffler W, Kim BH, Das R, Grishin NV, and Baker D**. Structure prediction for CASP8 with all-atom refinement using Rosetta. *Proteins* 77 Suppl 9: 89-99, 2009.
78. **Rohl CA**. Protein structure estimation from minimal restraints using Rosetta. *Methods in enzymology* 394: 244-260, 2005.
79. **Sali A, Shakhnovich E, and Karplus M**. How does a protein fold? *Nature* 369: 248-251, 1994.
80. **Scalley ML, Yi Q, Gu H, McCormack A, Yates JR, 3rd, and Baker D**. Kinetics of folding of the IgG binding domain of peptostreptococcal protein L. *Biochemistry* 36: 3373-3382, 1997.
81. **Schafer NP, Kim BL, Zheng W, and Wolynes PG**. Learning To Fold Proteins Using Energy Landscape Theory. *Israel journal of chemistry* 54: 1311-1337, 2014.
82. **Schwede T, Kopp J, Guex N, and Peitsch MC**. SWISS-MODEL: An automated protein homology-modeling server. *Nucleic Acids Res* 31: 3381-3385, 2003.
83. **Shapovalov MV, and Dunbrack RL, Jr.** A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure (London, England : 1993)* 19: 844-858, 2011.

84. **Shi S, Pei J, Sadreyev RI, Kinch LN, Majumdar I, Tong J, Cheng H, Kim BH, and Grishin NV**. Analysis of CASP8 targets, predictions and assessment methods. *Database : the journal of biological databases and curation* 2009: bap003, 2009.
85. **Shimada J, Kussell EL, and Shakhnovich EI**. The folding thermodynamics and kinetics of crambin using an all-atom Monte Carlo simulation. *Journal of molecular biology* 308: 79-95, 2001.
86. **Simons KT, Kooperberg C, Huang E, and Baker D**. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. *Journal of molecular biology* 268: 209-225, 1997.
87. **Simons KT, Ruczinski I, Kooperberg C, Fox BA, Bystroff C, and Baker D**. Improved recognition of native-like protein structures using a combination of sequence-dependent and sequence-independent features of proteins. *Proteins* 34: 82-95, 1999.
88. **Soding J**. Protein homology detection by HMM-HMM comparison. *Bioinformatics (Oxford, England)* 21: 951-960, 2005.
89. **Song Y, Tyka M, Leaver-Fay A, Thompson J, and Baker D**. Structure-guided forcefield optimization. *Proteins* 79: 1898-1909, 2011.
90. **Wang JM CP, Kollman PA** How well does a restrained electrostatic potential [RESP] model perform in calculating conformational energies of organic and biological molecules? . *J Comput Chem* 21: 1049–1074, 2000.
91. **Wolynes PG**. Energy landscapes and solved protein-folding problems. *Philosophical transactions Series A, Mathematical, physical, and engineering sciences* 363: 453-464; discussion 464-457, 2005.
92. **Wolynes PG, Onuchic JN, and Thirumalai D**. Navigating the folding routes. *Science* 267: 1619-1620, 1995.
93. **Wu S, Skolnick J, and Zhang Y**. Ab initio modeling of small proteins by iterative TASSER simulations. *BMC biology* 5: 17, 2007.
94. **Xu J, Li M, Lin G, Kim D, and Xu Y**. Protein threading by linear programming. *Pacific Symposium on Biocomputing Pacific Symposium on Biocomputing* 264-275, 2003.
95. **Yang Y, Faraggi E, Zhao H, and Zhou Y**. Improving protein fold recognition and template-based modeling by employing probabilistic-based matching between predicted one-dimensional structural properties of query and corresponding native properties of templates. *Bioinformatics (Oxford, England)* 27: 2076-2082, 2011.
96. **Zemla A**. LGA: A method for finding 3D similarities in protein structures. *Nucleic Acids Res* 31: 3370-3374, 2003.
97. **Zemla A, Venclovas C, Moult J, and Fidelis K**. Processing and analysis of CASP3 protein structure predictions. *Proteins* Suppl 3: 22-29, 1999.
98. **Zhang Y**. Template-based modeling and free modeling by I-TASSER in CASP7. *Proteins* 69 Suppl 8: 108-117, 2007.
99. **Zhang Y, Arakaki AK, and Skolnick J**. TASSER: an automated method for the prediction of protein tertiary structures in CASP6. *Proteins* 61 Suppl 7: 91-98, 2005.
100. **Zhang Y, and Skolnick J**. Automated structure prediction of weakly homologous proteins on a genomic scale. *Proceedings of the National Academy of Sciences of the United States of America* 101: 7594-7599, 2004.
101. **Zhang Y, and Skolnick J**. Tertiary structure predictions on a comprehensive benchmark of medium to large size proteins. *Biophysical journal* 87: 2647-2655, 2004.
102. **Zhang Y, and Skolnick J**. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Res* 33: 2302-2309, 2005.
103. **Zhou H, and Zhou Y**. Fold recognition by combining sequence profiles derived from evolution and from depth-dependent structural alignment of fragments. *Proteins* 58: 321-328, 2005.

104. **Zhou Y, and Karplus M**. Interpreting the folding kinetics of helical proteins. *Nature* 401: 400-403, 1999.