EVOLUTION AS A DESIGN TOOL TO INFORM BIOMOLECULAR ENGINEERING

BY

PRATIK LAHIRI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Agricultural and Biological Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Abstract

Enzyme biotechnology is a critical component of technologies needed for increased sustainable materials processing. Along with the ability to rapidly synthesize proteins through fermentation, there is a need to be able to alter enzyme functionality in specific ways to suit the desired application. For instance, industrial enzymes with increased stability at higher temperatures or altered pH optima can improve productivity in large-scale bioreactors through improved catalytic rates and lowered costs due to cooling and reduced contamination. This research aims to provide the scientific community with a suite of design tools and methodologies for protein engineering experiments and research. Specifically, these methods utilize a foundation of concepts from molecular evolution to provide insight into the innovation process in molecular engineering.

Methods developed in this study aim to increase thermal stability of an enzyme by engineering disulfide bonds and electrostatic salt bridges on the surface of the enzyme. Two methods to assist disulfide engineering were developed- 1) A neural network model to predict disulfide bonds within existing structures from mutual information and a continuous distributed representation of protein sequence. 2) A methodology incorporating statistics on the structural information of disulfide bonds in conjunction with evolutionary patterns to rank order specific design choices. A similar approach was developed for engineering salt bridges. The methodology for developing geometric constraints and utilizing evolutionary patterns to engineer salt bridges was validated with experiments on 1,4 -glucan branching enzymes by collaborators.

The neural network model achieves state of the art accuracy (80%) and in addition, the impact of the protein sequence representation, mutual information and cysteine separation distance on performance of the model were analysed. in a particular disulfide engineering experiment. The trained long short term memory (LSTM) neural network model also serves as a model of disulfide bond sequence motifs so as to develop an understanding of the constraints for disulfide bond formation. The methodology using statistical constraints on disulfide bonds was prototyped as a PyMOL script that identifies potential pairs of residues on the surface of an enzyme for modification to disulfide-capable cysteine residues. This method suggests suggests 85% more stabilising mutations out of 17% fewer suggestions according to evaluations by short Molecular Dynamics simulations using FoldX.

In loving memory of my dearest mother, and to my family for their love and support.

Table of Contents

List of Tables

List of Figures

List Of Abbreviations

BLAST	Basic Local Alignment Search Tool
CSD	Cysteine Separation Distance
CD-HIT	Cluster Database at High Identity with Tolerance
CBOW	Continuous Bag of Words
DbD2	Disulfide by Design 2
DSSP	Define Secondary Structure of Proteins
ET	Evolutionary Trace
GRU	Gated Recurrent Unit
GBE	1,4-α Glucan Binding Enzyme
LSTM	Long Short Term Memory
MD	Molecular Dynamics
MI	Mutual Information
PDB	Protein Data Bank
RNN	Recurrent Neural Network

Chapter 1

Introduction

Enzymes serve an important function in bioprocessing due to their high catalytic efficiency and exquisite substrate specificity. Most enzymes are mesophilic in origin and are functional in a narrow range of environmental conditions. However, there are a number of extreme environments in the biosphere where specialized microorganisms can thrive. Through necessity, these extremophilic (Hough and Danson 1999; Gomes and Steiner 2004) microorganisms produce enzymes capable of functioning in unusual conditions. Using mesophilic enzymes directly at higher than optimal temperatures have several drawbacks: They can irreversibly denature at higher temperatures and lose functionality, or the increased temperature can affect their structure to result in lowered selectivity and the production of undesirable byproducts (Demirjian, et al. 2001).

Enzymes from extremophiles perform the same functions as their mesophilic counterparts, albeit in extreme conditions such as high salt concentrations, at pHs far away from neutral in highly acidic or basic conditions, in non-aqueous media (Hough DW and Danson MJ 1999; Gomes J and Steiner W 2004) and at very low or high temperatures (Demirjian, et al. 2001; Rothschild and Mancinelli 2001). Extremophile enzymes allow bioprocesses to be conducted in environmental conditions that are more suitable to industrial bioprocessing, and are less hospitable to contaminant microorganisms. Their utility motivates the demand for enzymes that are functioning in abiotic conditions (Vielle and Zeikus, 2001). Thermophiles constitute an important subset of extremophiles and are of significant interest in the development of thermostable, industrially-relevant enzymes.

The availability of robust enzymes is always a critical bottleneck in the use of biocatalysts in industrial processes. There are several examples of enzymes derived from extremophiles that have, with or without further bioengineering, been of tremendous value in biotechnology. For instance-

- Thermostable DNA polymerases obtained from thermophilic organisms like Thermus aquaticus or Pyrococcus furiosus are the de facto standard used in DNA amplification, which has widespread utility in industry, medicine and research.

- Similarly, the (+)--lactamase from the thermophilic archaeon Sulfolobus solfataricus MT4 is used in the production of anti-HIV compound, Abacavir (Taylor, et al. 1993). This pharmacophore has further been stabilized by immobilization as a cross-linked, polymerized enzyme (Hickey, et al. 2009).

- L-aminoacylase is used for synthesizing precursors for many pharmaceutical compounds of interest. A thermophilic L-aminocyclase has been extracted from Thermococcus litoralis (Toogood, et al. 2002).

- From the same archeon, a pyroglutamyl carboxyl peptidase which is applicable in industry to cleave the pyroglutamyl group from blocked peptides, allowing them to be N-terminally sequenced. This protease derives its stability, partially, from disulfide bonds between the dimer subunit interfaces (Singleton, et al. 1999).

However, the search for thermophilic counterparts of mesophilic enzymes can be inefficient and expensive, especially when some bioengineering approaches exist. Bioengineering can provide an alternative route to modifying existing enzymes to make them more thermostable. Introducing disulfide bonds to cross-link the surface of an enzyme (Tatu, et al. 1990), or to use nanotechnology to produce cross-linked enzyme aggregates (CLEAs) (Sheldon, 2011), or to use salt bridges to increase the stability through ionic interactions (Lee, et al. 2014), or making the protein backbone more rigid, are some of the strategies that can increase thermostability while maintaining functionality.

Within the toolbox of biotechnology, there exist techniques such as directed evolution, site-directed mutagenesis, gene shuffling and immobilization (Bull, et al. 2000; Iyer and Ananthanarasyn, 2008; Kaul and Asano, 2011) that can result in thermostable variants of already-available enzymes. Extensive research has shown that thermal adaptation is determined by hydrophobicity, number of disulfide bonds, electrostatic interactions, rigidity, and packing density (Delboni, et al.

1995; Chang, et al. 1999; Rosato, et al. 2002; Xiao and Honig 1999; Brian and Dominy, 2004; Jaenicke, 2000; Reetz, et al. 2006). Of these adaptations, one strategy employed by thermophiles to increase stability of proteins is to increase its G, its change in Gibbs Free Energy required to denature an enzyme with temperature (Razvi and Scholtz, 2006). This increase can be achieved at the protein structural level through salt bridges, hydrogen bonds, disulfide bonds or hydrophobic interactions. Disulfide bonds and salt bridges can be introduced via point mutations and their ability to stabilize the structure is influenced by their geometric conformation. Additionally, there are considerable experimental data where they have been used to create thermostable variants of proteins. This data suggests that an increase in melting temperature, Tm of up to around 23 can be achieved (Dombrowski, et al. 2014; Lee, et al. 2014). However, many designed disulfide bonds result in a decreased melting temperature of the protein as well (Dombrowski, et al. 2014), and there is little by way of theory to understand why such approaches sometimes fail.

Chapter 2

Engineering Disulfide Bonds

2.1 Introduction

Disulfide bonds have been linked to increase in chemical stability, reduction in misfolding, resistance to enzymatic activity. Disulfide bonds have a bond energy of 251 KJ/mol and are among the strongest contacts in proteins and contribute to protein recognition, catalytic process and stability (Gngora-Bentez, et al. 2014). Disulfide bonds appear to have statistical properties such as geometric and sequence biases, as well as patterns of evolutionary stability. The structure and sequence biases have previously been used to create machine learning based computational methods for rational design of interactions. Hitherto these methods have been overly restrictive and predict many false negatives (Dani, et al. 2003). In addition to statistical properties, the design of disulfide bonds use mechanistic molecular dynamics (MD) simulations. Recent work on MD simulations shows that regions that unfold early are the most effective targets for improving protein stability (Dombrowski, et al. 2014). Disulfide by Design (DbD2) (Craig and Dombrowski, 2013; Dombrowski, 2003), MODIP (Dani, et al. 2003; Sowdhamini, et al. 1989) mainly rely on the idea of geometric constraints on disulfide bonds (Thornton, 1981; Petersen, et al. 1999) and B factor to determine regions of high mobility/flexibility.

These methods and early experiments in engineering disulfides revealed three heuristics for rational design of disulfide bonds- 1) Mutations in flexible regions of proteins or regions with medium to high mobility make stabilizing disulfide bonds. 2) Stabilizing mutations are more likely to be near the surface of the protein. 3) Large loop lengths (¿25 residues) provide more stability. Experimentally, however, these heuristics are also associated with reports of false positives (Dombrowski, et al. 2014; Dani, et al. 2013; Pellequer, et al. 2006) as high as 50%. This failure rate could

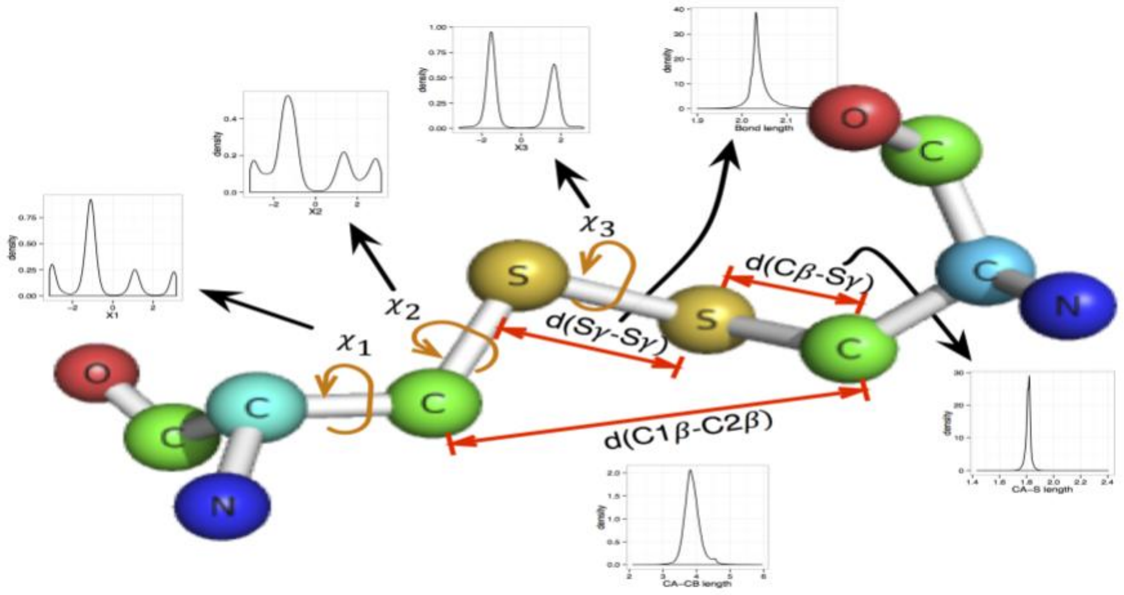

Figure 2.1: An illustration of the distance and angles in a disulfide bond.

be attributed to the restrictive nature of geometric models for engineering disulfide. On the other hand, there have been numerous reports of engineered disulfides with geometric parameters that fall outside the narrow range imposed by current methods.

The objective of this study was to develop additional design rules that incorporate geometric constraints and evolutionary patterns in the engineering of disulfide bonds and to codify these analyses in the form of a computer-aided design tool.

Based on a statistical analysis, DbD2 uses fixed bond length constraints, $C\beta - S\gamma$ and $S\gamma - S\gamma$ bond lengths of 1.81 Åand 2.04 Å, respectively. It also constrains $C\beta - S\gamma - S\gamma$ bond angle to 104.15°. It also uses an energy function based on the $\chi1$, $\chi3$ torsion angles. This method does not completely incorporate the statistical distributions and evolutionary patterns of substitutions. Further, the methodology constraints to maintain the $C\beta - C\beta$ distance of the original residues in the protein structure, thereby allowing no relaxation to it. We propose that introducing cysteines in the structure and then evaluating the $C\beta - C\beta$ distance is a less stringent yet more realistic set of constraints that trades off structural rigidity for thermal stability. A similar statistical analysis was performed in this study on a larger, updated set of proteins and developed a protocol that uses the bond length and torsion angles distribution in addition to substitution scores to generate a probabilistic score for potential disulfide bond mutations. Short Molecular Dynamics (MD) simulations were then performed using the FoldX package (Schymkowitz, et al. 2005) to identify stabilising mutations from the set of potential mutations identified by geometrical parameters.

2.2 Material and Methods

2.2.1 Dataset

All PDB structures with at least one disulfide bond were downloaded (November, 2017) from the RCSB protein data bank and culled to less than 90% sequence similarity using the CD-HIT webserver (Ying, et al. 2010). This dataset had 14407 disulfide bonds. For all PDB files in the dataset, evolutionary trace analysis results were downloaded (November, 2017) from the ET webserver (http://lichtargelab.org/software/ETserver). To evaluate the results, an experimental evaluation dataset of stabilizing, destabilizing and neutral mutations in 11 PDB structures- 2CBA, 1SNO, 2LZM, 1XNB, 1RNB, 9RAT, 1MNP, 3GLY, 1CHH, 2CI2, 4TNC reported in [15] was was selected to used to assess the performance of the developed protocol. Mutations that involve an existing cysteine in this dataset were ignored. The 11 structures result in a total of 23 mutations- 7 destabilizing, 2 neutral and 14 stabilizing as shown in Table 2.1.

2.2.2 Statistical analysis of Geometric and Evolutionary Constraints

On the dataset of all disulfide containing PDB structures, distributions of bond length, torsion angles($\chi1$,$\chi2$,$\chi3$) and backbone angles(ϕ,ψ) were computed. Disulfide cysteine substitution frequencies, i.e BLOSUM like substitution scores for disulfide cysteines were calculated using the multiple sequence alignments generated by the Evolutionary Trace server.

2.2.3 Molecular Dynamics Simulations

All PDB structures from the experimental dataset were repaired using the FoldX RepairPDB command with default parameters. MD simulations to categorize mutations as stabilizing, destabilizing or neutral were carried out using the FoldX BuildModel command on repaired PDB structure files with recommended mutations from our protocol as well as DbD2. Since all mutations were to cysteine, which has 3 rotamers, each mutation was repeated 3 times so that the simulation could test multiple rotamers and report the average change in G as well as the standard deviation to identify convergence problems with the simulations. A mutation was considered stabilising if it has a $\Delta\Delta$G < -0.5 kcal/mol, neutral if it has a -0.5 kcal/mol < $\Delta\Delta$G < 0.5 kcal/mol and destabilising if $\Delta\Delta$G > 0.5 kcal/mol. These cutoffs were chosen because FoldX has an error of 0.5 kcal/mol. We only report whether a mutation was stabilising or destabilising since the $\Delta\Delta$G values reported by FoldX do not correlate well with experimental results but, have been proven to be able to predict if a mutation is stabilising or destabilising (Potapov, et al. 2009).

2.2.4 Disulfide Engineering Protocol

First, PDB structure files were repaired using FoldX using the procedure described above. Using the geometric distributions and substitution frequencies of cysteine residues to other amino acids, a PyMol plugin was developed. This plugin, examines at all possible pairs of residues on the surface of a protein. A surface residue was defined as a residue with greater than 0.5 Å^2 exposed area. From all pairs of surface residues, pairs which had Cγ – Cγ distances greater than 3.5 Å and less than 4.25 Å were identified. For each of these candidates, the plugin introduces cysteines using the pymol mutagenesis wizard and calculates the probability that it could assume one of the three cysteine rotamers using the pymol mutagenesis wizard. For each rotamer, it also calculates the probability scores for Cγ – Cγ distance and χ3 torsion angle. If a mutation has at least one rotamer with both these probabilities >0, it is considered a potential mutation. Additionally, substitution scores for all potential mutations are also reported by adding the scores of the two residues in the mutation. All potential mutations are evaluated using Foldx MD simulations as either being stabilising, neutral or destabilising.

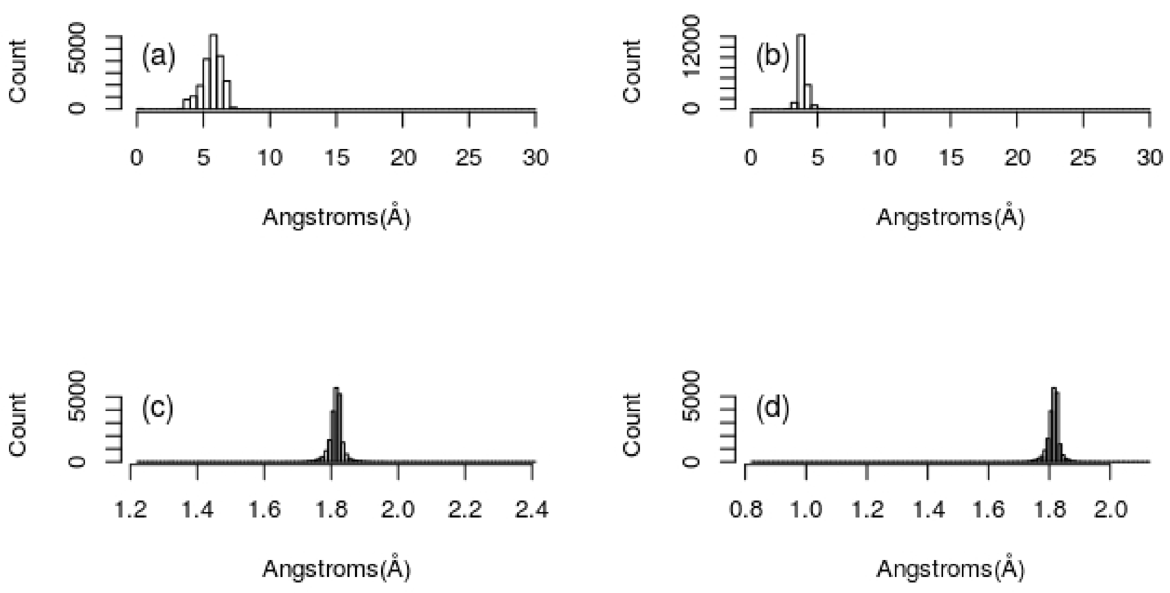

Figure 2.2: Distributions of various distances in a disulfide bond. a) Cα –Cα b) Cβ – Cβ c) Cβ – S of the N-terminal Cys d) Cβ – S of the C-terminal Cys

2.3 Results

2.3.1 Distributions of Disulfide Bond Geometries

The plots of the distances and angles are able to recapitulate previously reported data. Fig. 2.2 shows the distributions of Cα – Cα, Cβ – Cβ and Cβ – S distances in disulfide bonds. A large majority of Cβ – Cβ distance are between the 3.5 – 4.25 Årange. This range serves as a good cutoff for filtering pairs of residues to be evaluated for mutations into disulfide bonds. Fig. 2.3 shows the distributions of the backbone dihedral angles as well as the torsion angles. The $\chi3$ torsion angles show two peaks while the dihedral angles show three peaks corresponding to the three rotamers of cysteine.

2.3.2 Disulfide cysteine substitution

The substitution scores for the cysteine residues in the disulfide bonds show notable differences from the BLOSUM62 scores for cysteine residues (Fig. 2.4). Most often (50% of the time) cysteine residues are deleted. The top 3 residues that a disulfide cysteine is substituted to are Serine,

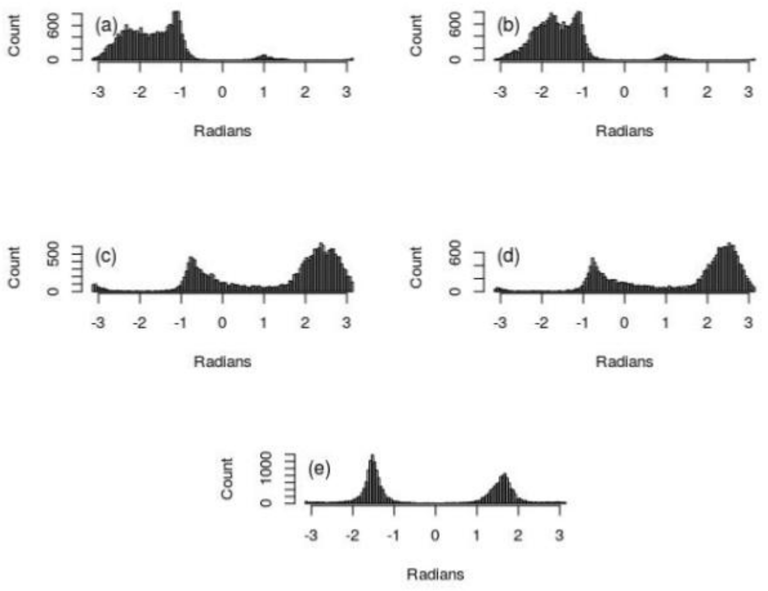

Figure 2.3: Distributions of angles in a disulfide bond. a) ϕ of N-terminal Cys b) ϕ of C-terminal Cys c) ψ of N-terminal Cys d) ψ of C-terminal Cys e) $\chi3$.

Alanine and Glycine. Tryptophan, Proline and Phenylalanine are the least likely substitutions. These patterns suggest that disulfide cysteines are likely to be substituted by amino acids with side chains of similar shape and size as cysteine- minimizing steric hindrance with atoms of spatially neighbouring residues. Thus, a disulfide engineering tool should eliminate geometrically feasible but sterically unfeasible sites and this, in turn, could increase the overall stability of the protein.

2.3.3 PyMol Plugin

The PyMol plugin developed provides an animation as it identifies potential mutations. Screenshots of the interface have been produced in Fig 2.5. Table 2.2 summarizes the results of the evaluation of our protocol and DbD2 based on the results of Foldx simulations and reported experimental results. Our method predicts twice the number of stabilising mutations as compared to DbD2 and includes all their stabilising mutations. Based on these results the suggested protocol for designing disulfide bridges has been presented as a flowchart in Fig. 2.6. Further, detailed results can be found in Appendix A and B.

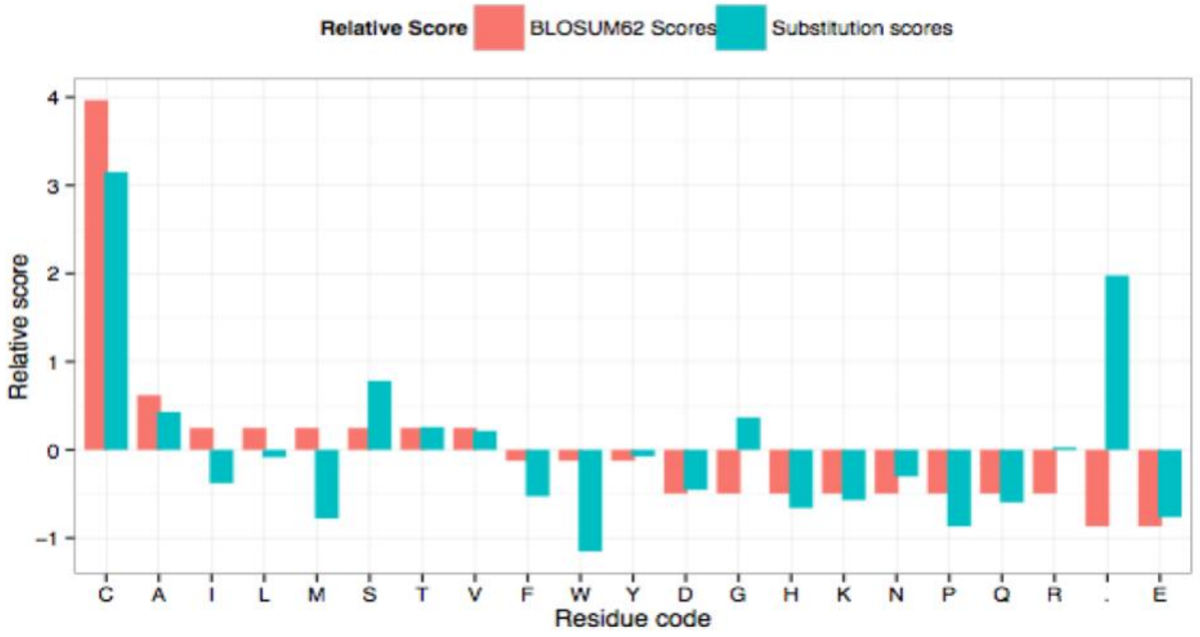

Figure 2.4: Substitution scores for cysteine residues in disulfide bonds and BLOSUM62 substitution scores

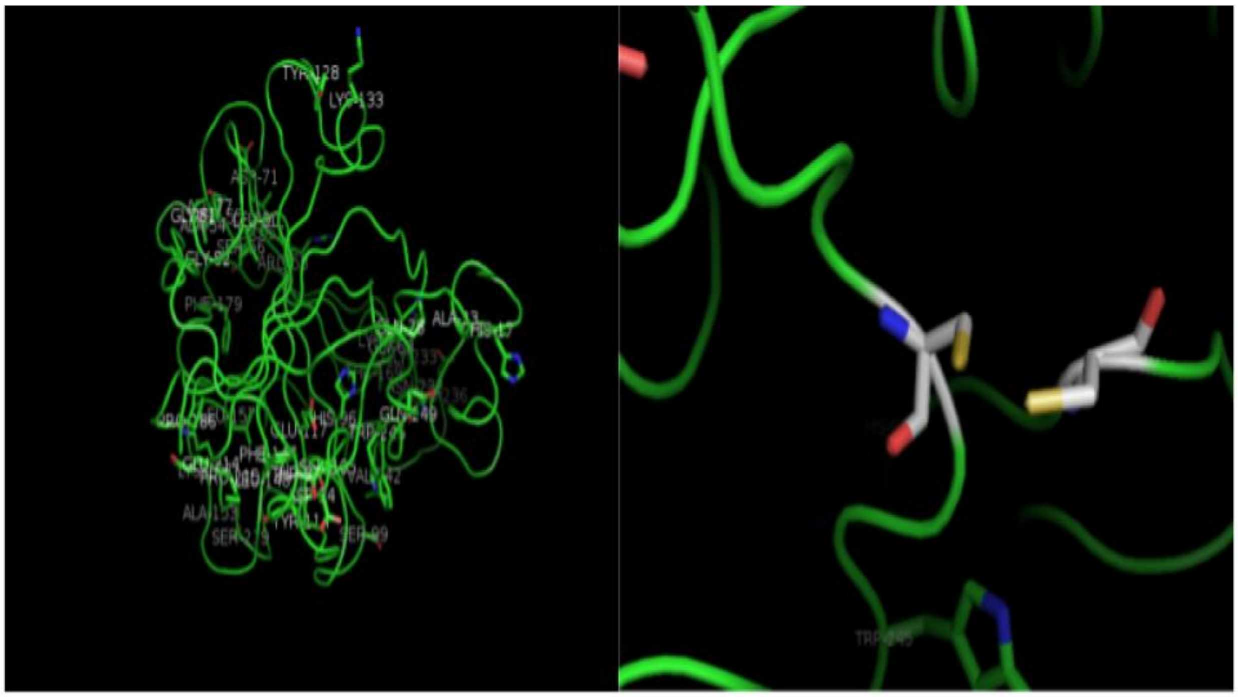

Figure 2.5: Screenshots of the pymol plugin on 2CBA. Left: Identifying surface residues with 3.5 < Cβ-Cβ < 4.25Å, Right: Mutating a residue pair to Cys.

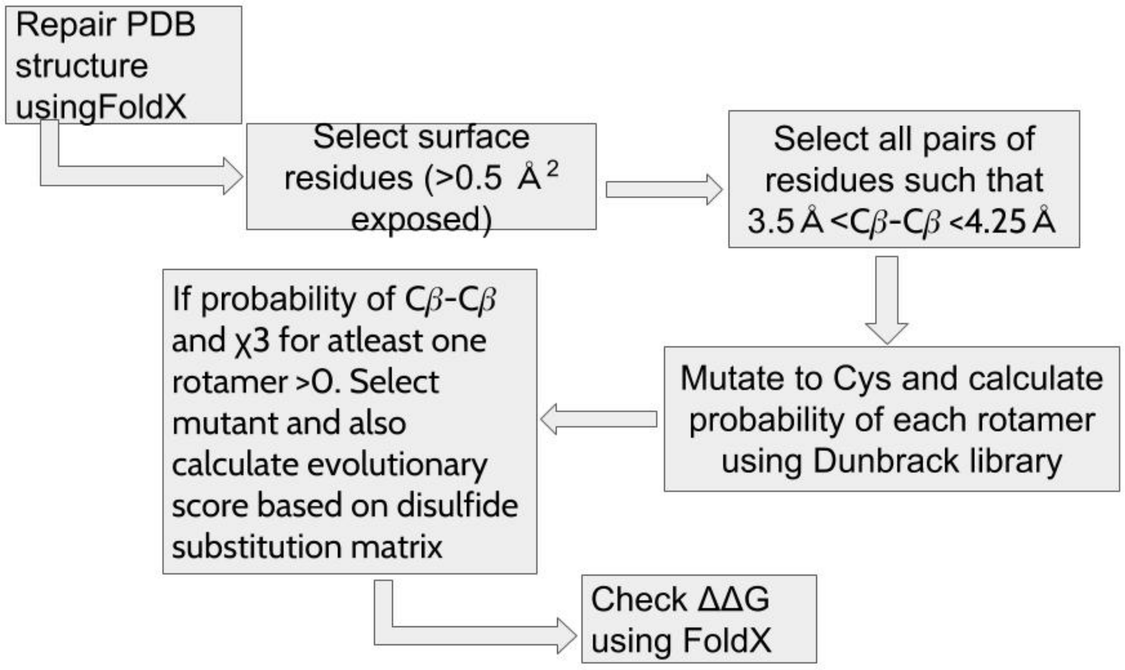
Repair PDB structure usingFoldX
Select surface residues (>0.5 Å² exposed)
Select all pairs of residues such that 3.5 Å <Cβ-Cβ <4.25 Å
Mutate to Cys and calculate probability of each rotamer using Dunbrack library
If probability of Cβ-Cβ and χ3 for atleast one rotamer >0. Select mutant and also calculate evolutionary score based on disulfide substitution matrix
Check ΔΔG using FoldX

Figure 2.6: Suggested protocol for designing disulfide bridges

2.4 Discussion

In this study we developed a method of analyzing an existing protein structure and recommending mutations to cysteinse that result in thermostabilizing disulfide bonds. The recommendations are based on geometric constraints of disulfide bonds using distributions of disulfide bond geometry from PDB structures. However, instead of strictly adhering to these distributions, we designed reasonable cutoffs as filters based on these distributions. We compared the performance of our method to Disulfide by Design 2 (DbD2) based on known mutations in an experimental dataset. Both methods perform poorly in identifying disulfide mutations that have been experimentally proven to be stabilising. Since any method based on geometrical constraints cannot capture the flexibility in a protein chain, our methods impose overly restrictive constraints. This is evidenced by the fact that both methods do not suggest any mutations that have been experimentally proven to be destabilising. Combined with the very few reported experimental results, a low rate of predicting experimentally stabilising mutations is expected. Earlier methods have evaluated their methods by predicting existing disulfide bonds. Since >85% of disulfide bonds in PDB structures have perfect geometry (Dani, et al. 2003), it is likely that disulfide bonds are a dominant factor in reshaping the rest of the protein in its presence. In order to determine if a suggested mutation produces a stabilizing change in protein conformation, FoldX was used on the results of both DbD2 and this method as a relatively realistic comparison method. The method proposed here is able to suggest 85% more number of stabilising disulfide mutations according to MD simulations by Foldx compared to DbD2 and includes all the stabilising mutations suggested by DbD2. Since both the methods suggest approximately the same number of mutations, our method is much more accurate by this method of evaluation. Nonetheless, more extensive testing of these methods is required on a larger dataset. An analysis of the reasons for the poor performance in predicting experimentally verified stabilising mutations could inform future efforts. However, since experimental data is scarce, we must be careful not to base the entire method on experimental data.Such an method will not generalise well to unseen data. In our method, we believe that the cutoffs for C-C distances and identifying surface residues may be relaxed even further.

Table 2.1: Summary of Experimental Evaluation Data

PDB ID	**Mutations**	**S/D/N**$^{\alpha}$
2CBA	Ala38–Ala258	D
	Ser29–Ser197	D
	Leu60–Ser173	S
1SNO	Asp77–Asn118	D
	Gly79–Asn118	S
	Gln80Lys116	S
2LZM	Thr21–Thr142	S
	Ser90–Gln122	S
	Ser90–Gln122	D
	Asp127–Arg154	D
1XNB	Val98–Ala152	S
	Ser100–Asn148	S
1RNB	Ala43–Ser80	S
	Thr70–Ser92	D
	Ser85–His102	S
9RAT	Ser24–Ser87	N
	Ala4–Val118	S
1MNP	Ala48–Ala63	N
3GLY	Asn20–Ala27	S
	Thr72–Ala471	N
1CHH	Val20–Thr102	D
2CI2	Thr22–Val82	S
4TNC	Met48–Met82	S

^{a}: S-Stabilizing, N-Neutral, D-Destabilizing

Table 2.2: Summary of Results on Experimental Evaluation Data

PDB ID	**No. of stabilizing$^{\alpha}$ mutations in this protocol/-Total**	**No. of stabilizing$^{\alpha}$ mutations by DbD2/Total**	**TP$^{\beta}$/Total in this study**	**TP$^{\beta}$/Total in DbD2**
2CBA	4/24	0/29	0/1	0/1
1SNO	1/15	1/19	0/2	0/2
2LZM	0/16	0/15	0/2	0/2
1XNB	0/23	0/23	0/2	0/2
1RNB	1/9	0/12	0/2	0/2
9RAT	1/8	1/14	0/1	0/1
1MNP	4/43	3/54	0/1	0/0
3GLY	1/38	1/60	1/1	1/1
1CHH	0/8	0/0	0/0	0/0
2CI2	0/2	0/3	0/1	0/1
4TNC	1/8	1/5	0/1	0/1

$^{\alpha}$: Stabilising according to Foldx MD simulations.
$^{\beta}$: TP- number of stabilising mutations predicted that have been experimentally reported.

Chapter 3

Sequence Effects for Engineering Disulfide Bonds

3.1 Introduction

Models based on stereochemical criteria are unable to fully encapsulate the flexibility of protein structure formation since they focus on the geometry of the the two cysteine amino acids in the disulfide bond. Using sequence we can learn longer range interactions. In fact, there are very fast and accurate methods that can predict bonding state of cysteines from sequence information alone (Fariselli and Casadio, 2001; Chen, et al. 2004; Savojardo, et al. 2013). These methods reveal that when analyzing multiple sequence alignments of query sequences positions that disulfide cysteines show a clear pattern of coevolution (Gbel, et al. 1994). This observation has been utilized to create multiple scores for quantifying co-evolution and use them to predict residues in contact in proteins to the extent of creating contact maps of proteins from the sequence alone (Jones, et al. 2012; Savojardo, et al. 2013). We propose that combining structure, sequence and evolution based features will lead to a more robust disulfide engineering algorithm.

As earlier mentioned, disulfide bonds, formed between two cysteine amino acids are generally long-range bonds in protein structures and constrain the conformational space of a protein structure. It is also well known that there exist general short range regularities in the primary structure of proteins (Vonderviszt, et al. 1986). Presumably, the neighboring residues have strong and probably deterministic influence to the chemical property of cysteine in forming disulfide bond (Muskal, et al. 1990). So, there has been considerable research in solving a problem closely related to disulfide engineering- predicting disulfide bonding pattern and methods developed so far focus on the local sequence context. Machine learning algorithms such as support vector machine (SVM) (Chen, et al. 2006), kernel method (Vincent, et al. 2008), correlated mutation analysis (Rubenstein and

Fiser, 2008; Raimondi, et al. 2015) and support vector regression (SVR) (Savojardo, et al. 2011; Savojardo, et al. 2013; Song, et al. 2007); and novel features such as protein subcellular localization (Savojardo, et al. 2011) correlated mutations (Savojardo, et al. 2013) and context-based features (Yaseen and Li, 2013) have also been applied. There are two issues with all these methods-

1) They do not scale to large datasets.

2) To circumvent this bottleneck, these methods train and test on reduced datasets where sequence similarity between protein chains is <35%.

However, we found that the local sequence context (which is the only sequence input to these methods) for such a dataset has sequence similarity as high as 80%. So, the sequence similarity restriction in these methods is meaningless.

However, approaches to the disulfide bonding pattern problem can be adapted to engineering disulfide bond. Instead of predicting bonding patterns, algorithms that are trained on predicting whether two cysteines are bonded using sequence and evolution based features, will expand the narrow constraints learned by purely geometric and structural methods.

To solve the issues raised above, we implemented a long short term memory (LSTM) neural network which scales well with large datasets. An LSTM would also able to learn sequence context based features, much like the simple context free and context sensitive languages it has been proven to be able to learn (Gers and Schmidhuber, 2001). An LSTM is a type of a recurrent neural network (RNN). Recurrent neural networks are designed to process sequential information like humans do while reading text. Hence, the most common applications of RNNs are in natural language processing. An RNN is made of the same recurring unit unfolded over the sequence of inputs (as show in Fig. 3.1).

The equations governing an RNN are-

$$s_t = f(Ux_t + Ws_{t-1})$$

$$o_t = softmax(Vs_t)$$

Where,

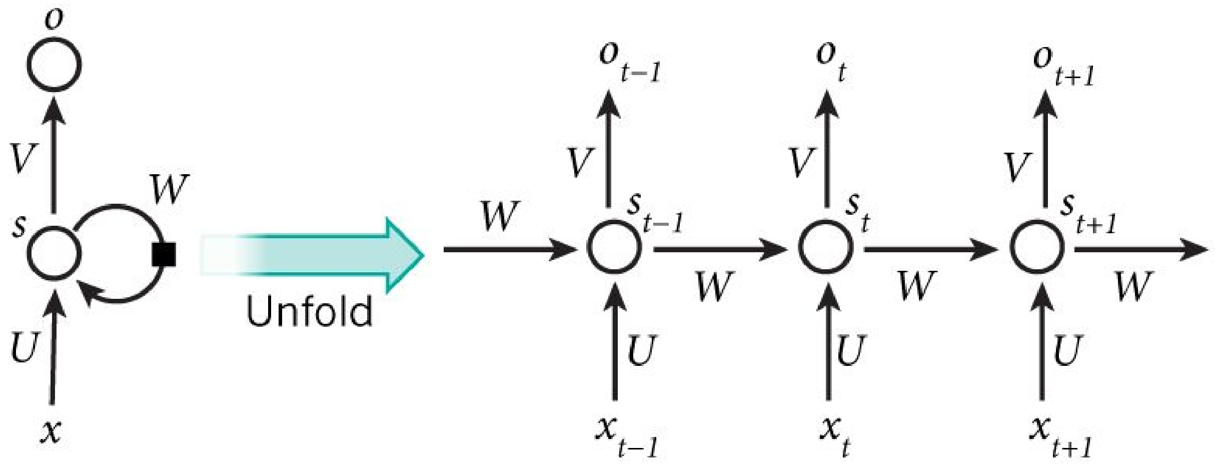

Figure 3.1: A recurrent neural network and the unfolding in time of the computation involved in its forward computation from Nature.

- s_t is the hidden state at time t.
- x_t is the input at time step t.
- o_t is the output at time step t.

The hidden state, s_t captures the memory of the previous input and the weights U,V,W are shared across all steps (each value in the input sequence). Theoretically, RNNs should be able to store information from arbitarilly long sequences. In practice, they are able to handle only short sequences because of vanishing gradients due to multiplication of many small numbers while calculating the gradient. To solve this an LSTM introduces a gating mechanism. It uses input, output and forget gates to selectively remember pertinent information at every step and output information relevant to the next state. The equations describing an LSTM are-

$$i = \sigma(x_t U^i + s_{t-1} W^i)$$

$$f = \sigma(x_t U^f + s_{t-1} W^f)$$

$$o = \sigma(x_t U^o + s_{t-1} W^o)$$

$$g = tanh(x_t U^g + s_{t-1} W^g)$$

$$c_t = c_{t-1} \circ f + g \circ i$$

$$s_t = tanh(c_t) \circ o$$

Here $\circ$ refers to elementwise multiplication. i,f,o are the input, forget and output gates respectively. The weights U,V,W from Fig. 3.1 have 4 sub-parts corresponding to the input, forget, output gates and the "candidate" hidden state.

As discussed in the introduction, the success of the various approaches significantly depends on the the features used. This can also be thought of as a data representation problem. In natural language processing, an arbitrary encoding scheme provides no useful information to the system regarding the relationships that may exist between the individual symbols. Hence, for example, the model cannot leverage anything it has learned about the word cat to help understand dog. Word2Vec is the current state of the art method to generate word embeddings from a text corpus(Mikolov, et al. 2013). There are two methods of generating this embedding- the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. CBOW predicts target words from source context words, while the skip-gram predicts source context-words from the target words. CBOW has the disadvantage that it treats the whole context as one observation and so it smooths over some of the information in the corpus. However, skip-gram treats each context-target pair as a new observation. This can be advantageous when we have a large dataset for training. An extension to word2vec for proteins has been proposed (Asgari and Mofrad, 2015). This embedding encodes amino acid 3 grams in a 100d vector space and has been trained on the set of all protein sequences in the SWISS-PROT database. These features capture biophysical characteristics of amino acid 3-grams. Hence, they can be used to learn the local structural characteristics of disulfide bonds to be able to better predict connectivity.

3.2 Material and Methods

3.2.1 Dataset

All PDB structures with at least one disulfide bond were downloaded from the RCSB protein data bank and culled to less than 90% sequence similarity using the CD-HIT webserver (Ying, et al.

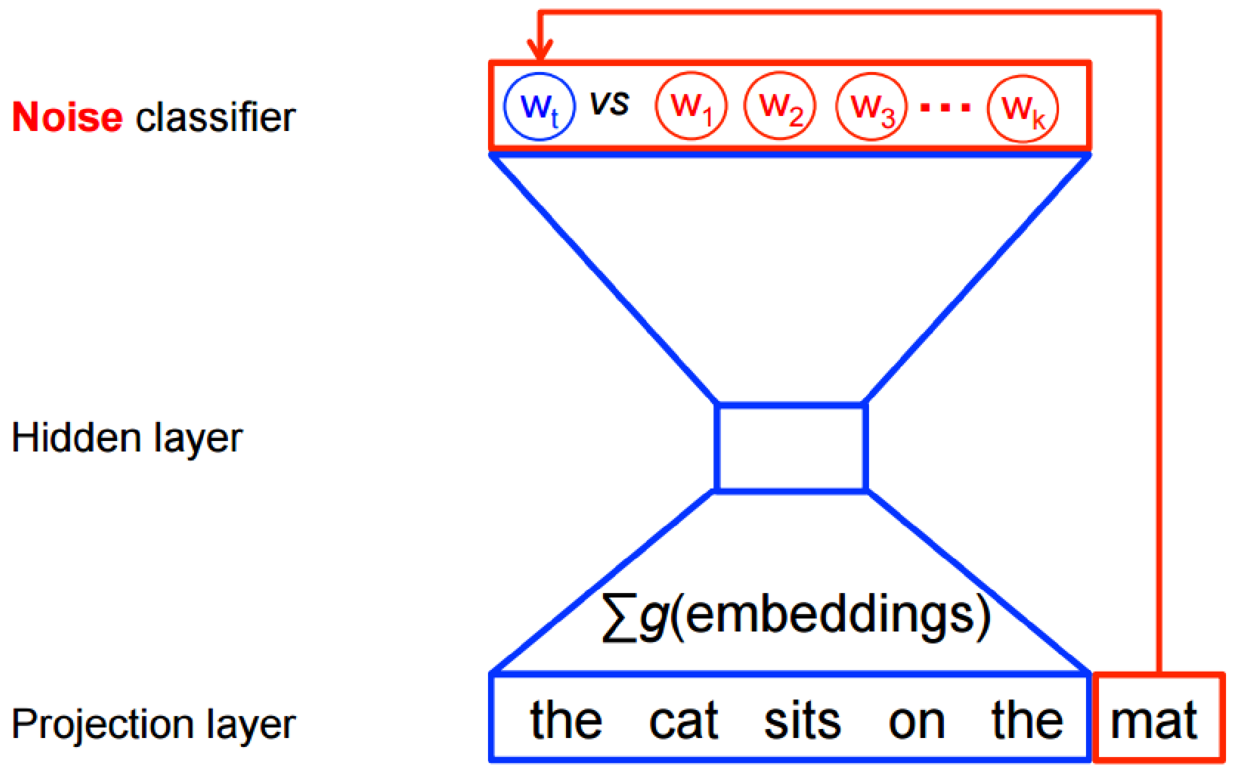

Figure 3.2: Skip-gram model for Word2Vec
(`https://www.tensorflow.org/tutorials/word2vec`)

2010). This dataset had 14407 disulfide bonds. For all PDB files in the dataset, evolutionary trace analysis results were downloaded from the ET webserver (`http://lichtargelab.org/software/ETserver`). The multiple sequence alignments generated by the server as part of the evolutionary trace analysis were used for all evolutionary analyses. The ProtVec (Asgari and Mofrad, 2015) word embeddings were downloaded from harvard dataverse `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/JMFHTN`.

3.2.2 Evolutionary stability of local sequence context

Here we define local sequence context as the ± 3 residues in the primary sequence around the disulfide cysteines. This yields 12 positions in the primary sequence for each disulfide bond. For all protein chains in our dataset, their multiple sequence alignments from section 3.2.1 above, were used to calculate the sequence similarity of the local sequence context at the aligned positions of homologous sequences.

3.2.3 Features in LSTM

To represent amino acid sequences as input vectors to the LSTM/GRU model, the ProtVec representation was used. The 6 residues upstream and downstream of each cysteine in a cysteine pair were split into 3-grams using a sliding window to generate a vector of length 22. Since disulfide bonds have been well documented to have a covariance pattern, we used MIp (Dunn, et al. 2008) which is the best measure of covariation that compensates for random noise and phylogeny to identify truly coevolving positions. Previous methods have used cysteine-separation-distance (CSD) as a feature to predict disulfide bonds and a statistical analysis of disulfide bonds does reveal particular sequence separation preferences in disulfide bonds(Tsai, et al. 2005) and following the results of that paper, in our model we also used the logarithm of the cysteine separation distance.

3.2.4 Training the LSTM

LSTM and GRU(Gated Recurrent Unit) networks in unidirectional and bidirectional configurations with a varying number of hidden units (50,100,150), dropout, recurrent dropout and regularisation were trained. As positive examples, sequence context, mutual information and cysteine separation

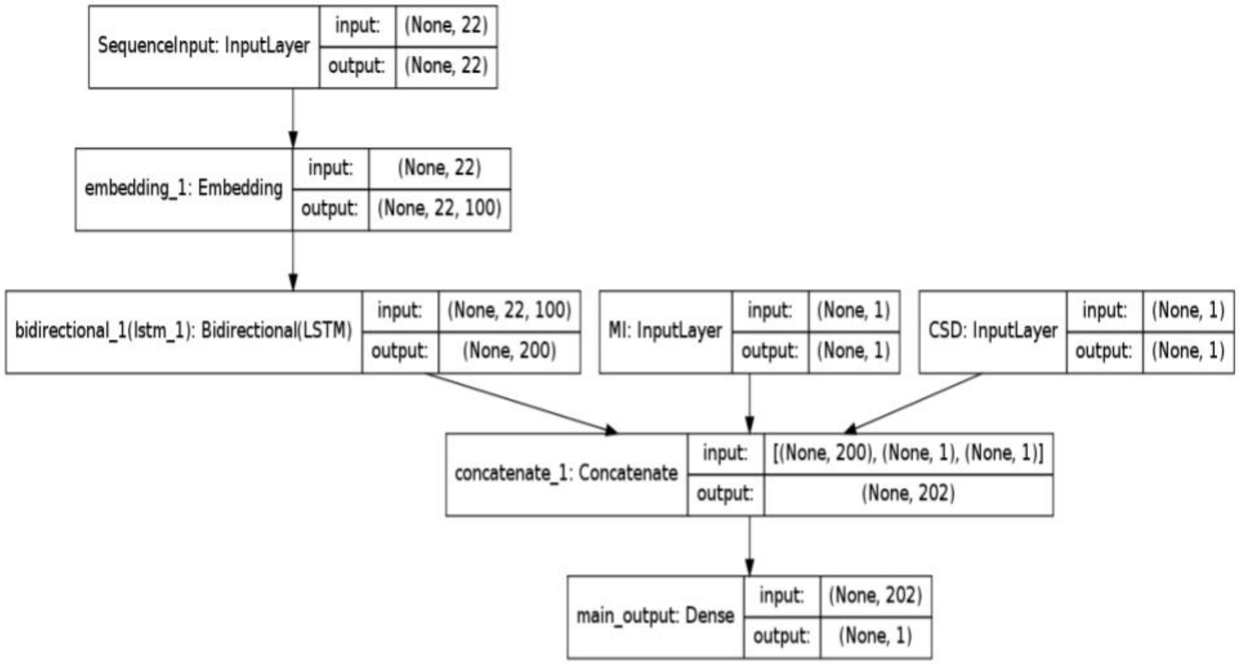

Figure 3.3: A bidirectional LSTM with 100 hidden units

distance of all 14407 disulfide bonds were used. To create the negative examples, from all chains with greater than equal to 2 disulfide bonds, sequence context, mutual information and cysteine separation distance of 14407 mismatched pairs were used. Here a mismatched pair consists of cysteines from two different disulfide bonds in the protein chain.

3.3 Results

3.3.1 Sequence context of disulfide bonds

In this study, patterns of evolution of the local sequence context of disulfide bonds were examined for insights into the evolutionary basis of the different sequence motifs of disulfide and non-disulfide cysteines. In Fig. 3.4 we observe that if a disulfide is substituted by another amino acid or deleted, the local sequence context conservation shifts to $< 50\%$ conservation, while in the case where a disulfide is deleted, a drastic shift to $< 10\%$ conservation is observed. From this we can surmise that while there are subtle changes in the local sequence context due to substitutions, a deletion leads to drastic changes.

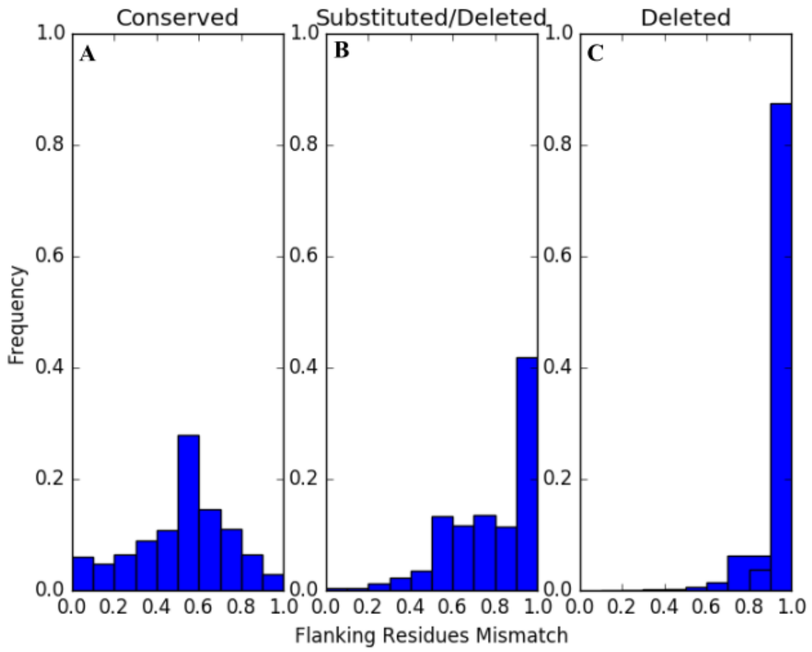

Figure 3.4: Fraction of ±3 residues flanking a disulfide bond in homologous sequences when the disulfide bond is conserved, when it is substituted/deleted, deleted.

3.3.2 Neural network results

The results of machine learning experiments are summarized in Table 3.1. The results report the mean accuracy after 5-fold cross validation. Comparing the results of the model without the ProtVec representation and its corresponding model with ProtVec (Unidirectional + MI), a 10% increase in accuracy is observed. This justifies the use of ProtVec as well as provides a method to incorporate the physicochemical properties of amino acid triplets into the model. The ProtVec representation provides larger accuracy gain than mutual information (MI) and cysteine separation distance (CSD) and so is a more important feature determining disulfide bond formation. The order of importance from our results is ProtVec > MI > CSD. All the best performing models for each configuration+feature type had 150 hidden units (the largest size tested). Since LSTM models are able to learn longer sequences than GRU, they consistently outperform GRUs. Finally, the mean accuracies are similar to the state of the art accuracies previously reported for disulfide bonding pattern prediction problem.

Table 3.1: Machine learning accuracy results

Configuration+Features	**LSTM (hidden units)**	**GRU (hidden units)**
Unidirectional+WithoutProtVec+MI	64.19%(150)	64.62%(150)
Unidirectiontal+MI	76.05%(150)	75.29%(150)
Unidirectional+CSD	74.7%(150)	72.91%(150)
Unidirectional+MI+CSD	78.94%(150)	76.97%(150)
Bidirectional+CSD	75.84%(150)	74.67%(150)
Bidirectional+MI+CSD	79.68%(150)	78.31%(150)

3.4 Discussion

This study serves as a first attempt to deeply examine the nature and extent of sequence effects in disulfide bonds. Based on the data and results presented here, the following conclusions and avenues for further studies can be drawn-

- Long range sequence interactions determine disulfide bonding (LSTM vs GRU performance Table 3.1). Further extending the local sequence context in training models should be explored. Since, LSTMs and GRUs treat the input as sequential input, this would not require a larger number of parameters to be learned.

- Since the best performing models had the largest number of hidden units tested (150 Table 3.1) and the sequence context for substituted disulfide bonds show minor decrease in sequence similarity (Fig. 3.2), likely, a lot of complex sequence motifs exist for disulfide bond formation.

- A physicochemical representation of protein sequences is vastly more important than mutual information and cysteine separation distance. While mutual information serves as a proxy for contact among residues, cysteine separation distance is important because longer loops created by disulfide bonds lead to increased stability. However, these are at best, indirect clues. Further efforts should focus on creating better representations of protein sequences.

Chapter 4

Patterns of Salt Bridge Evolution for Rational Design of Thermostable Enzymes

[1]

4.1 Introduction

Electrostatic interactions between oppositely charged amino acids- the anionic carboxyl group of glutamate (E) or aspartate (D) and the cationic ammonium group of arginine (R) or lysine (K) at a distance of < 4Å are called salt bridges (Musafia, et al. 1995; Donald, et al. 2011). Salt bridges are commonly found in proteins, and contribute to their structural and functional conformation. They also contribute to catalysis, stability, protein degradation and recognition (Takahashi 1996; Xu, et al. 1997; Elcock and McCammon 1998; Albeck, et al. 2000; Kumar and Nussinov 2002a, Nussinov and Kumar 2002b; Bosshard, et al. 2004). However, the contribution of salt bridges to stability of proteins remains unresolved (Elcock and McCammon 1998; Donald, et al. 2011). Salt bridges (electrostatic interactions) can be easily formed, however, the contribution of salt bridges to stability of proteins varies (Nussinov and Kumar 2000; Gribenko and Makhatadze 2007). Several of these reports have shown an unfavorable effect of salt bridges on stability, whereas others have demonstrated favorable contributions. Consequently, the rational design of salt bridges is still challenging because of lacking of comprehensive understanding of their contribution to stability of proteins.

Previous studies showed that biological properties of proteins depend on cumulatively cooperative interactions of amino acids (Gribenko and Makhatadze 2007). Multiple studies have explored the geometric conformational constraints of salt bridges, sequence separation, secondary structure

[1]This chapter has been adapted from- Evolutionary Stability of Salt Bridges Hints its Contribution to Stability of Protein. Xiaofeng Ban, Pratik Lahiri, Abhishek S. Dhoble, Zhengbiao Gu, Caiming Li, Li Cheng, Yan Hong, Zhaofeng Li, Bhalerao Kaustubh. Xiaofeng Ban contributed the experimental results on GBE reported here.

distribution and flexibility (Nussinov and Kumar 2002a; Nussinov and Kumar 2002b; Donald, et al. 2011). These results have been incorporated into empirical rules for design of salt bridges (Lee et al. 2014). However, none of the studies or methods have included homology and phylogeny. We believe that incorporating evolutionary information will enable us to identify salt bridges that are important for stability.

A commonly used measurement of importance of an amino acid within a protein chain is its conservation. In addition to conservation of a position in a multiple sequence alignment of homologous sequences, another method of measuring conservation is the evolutionary trace (ET) algorithm (Mihalek, et al. 2006; Lua, et al. 2016). The ET is a phylogenetic approach that serves as an indicator for the evolutionary pressure on an amino acid and reliably measures the conservation of amino acid residues in proteins (Lua and Lichtarge 2010; Amin, et al. 2013; Lua, et al. 2016; Mihalek, et al. 2004). It is a powerful tool that has been shown to identify residues crucial to protein stability as well as active sites. It is based on the principle that amino acid mutations occurring at positions that are closer to the root of the phylogenetic tree correspond to functional changes, while variations closer to the leaf nodes of the tree correspond to negligible functional changes. (Ng and Henikoff 2001; Madabushi, et al. 2004; Ward, et al. 2009; Katsonis, et al. 2014).

This evolutionary trace method enables us to explore the contribution of salt bridges to proteins from an evolutionary perspective. Here we study the conservation of the salt bridge itself as well as the conservation of the local sequence context and its influence on protein stability. Additionally, from an engineering perspective we ask,
1) Is it possible to identify the importance of any particular salt bridge? and
2) Can we engineer salt bridges to predictively alter the biological properties of a protein?

Based on our analysis, we hypothesize that salt bridges influence the evolutionary stability of their neighborhood. Specifically, alterations at one residue in a salt bridge are correlated with variations in the other residue and also affect the evolutionary stability of local sequence around the salt bridge. As a corollary, we expect that salt bridges would increase the evolutionary stability of the neighborhood when at least one half of a salt bridge is conserved, which is beneficial for the stability of whole proteins. To validate the hypothesis, we present experimental results of engineering salt bridges in existing structure to positively affect the thermostability of an enzyme.

4.2 Material and Methods

4.2.1 Development of the salt bridge data set

In order to establish a database of salt bridges, the protein database was selected with resolution no less than 3.0Å and an R-factor of no more than 0.25 (Kumar and Nussinov 2002b; Donald, et al. 2011). For each protein, a set of homologous sequences was retrieved from the NCBI Entrez non-redundant protein sequence database with the E-value < 0.05 (Maglott, et al. 2011). In addition, if two sequences shared more than 99.5% sequence identity, one was arbitrarily selected.

Salt bridges were defined as an interaction between two oppositely charged groups, such that the distance of heavy atom in each pair is the less than 4.0Å . Here, Arg (R) or Lys (K) provide the positively charged cationic ammonium ($RNH3^+$) while, Asp (D) or Glu (E) provide the negatively charged anionic carboxylate ($RCOO^-$). His (H) has an ambiguous protonation state at pH 7.0, and so we excluded it from our dataset. Thus, we considered, Arg-Glu, Arg-Asp, Lys-Glu and Lys-Asp salt bridges for the creation of the database. Local salt bridges were defined as salt bridges where the sequence separation between the oppositely charged residues was less than or equal to five (Donald, et al. 2011).

4.2.2 Secondary structure elements

We used secondary structure elements as defined by DSSP, which is a program to assign secondary structures to segments of protein sequences based on the sequence information alone (Kabsch and Sander 1983). The DSSP program defines eight categories for secondary structures: H = -helix; B = residue in isolated -bridge; E = extended strand, participates in ladder; G = 3-helix (310 helix); I = 5 helix (-helix); T = hydrogen bonded turn; S = bend and R = coil or other random coil (Kabsch and Sander 1983; Sarakatsannis and Duan 2005; Gvritishvili, et al. 2008).

4.2.3 Evolutionary stability

The evolutionary stability analysis for our dataset was performed by identifying the substitutions at salt bridge positions in homologous sequences for all protein chains. Homologous sequences were obtained from BLAST Tool in NCBI database-
https://blast.ncbi.nlm.nih.gov/BlastAlign.cgi. The sequence alignments were carried out by ClustalW2 (gap open penalty 10, gap extension penalty 0.05)(Li 2003; Margelevicius and Venclovas 2005). The frequency of substitutions for all amino acids for each type of salt bridge (Arg-Glu, Arg-Asp, Lys-Glu, Lys-Asp) was calculated.

4.2.4 Regional evolutionary stability

Regional evolutionary stability was calculated by counting the frequency at which the three residues upstream and downstream of salt bridge residues in the primary sequence were conserved in the multiple sequence alignments of homologous sequences.

4.3 Results

4.3.1 Distribution characteristics of salt bridges

Our databases of salt bridges successfully recaptures the previously reported distribution patterns and characteristics of sequence separation, secondary structure preferences, C-C and N-O distances. Here we present results for sequence separation and secondary structure preferences for all salt bridges as well as local salt bridges.

The sequence separation between the two halves of the salt bridges were calculated and are illustrated in Fig. 4.1(A,B). The results show that there is a sharp concentration in the distribution

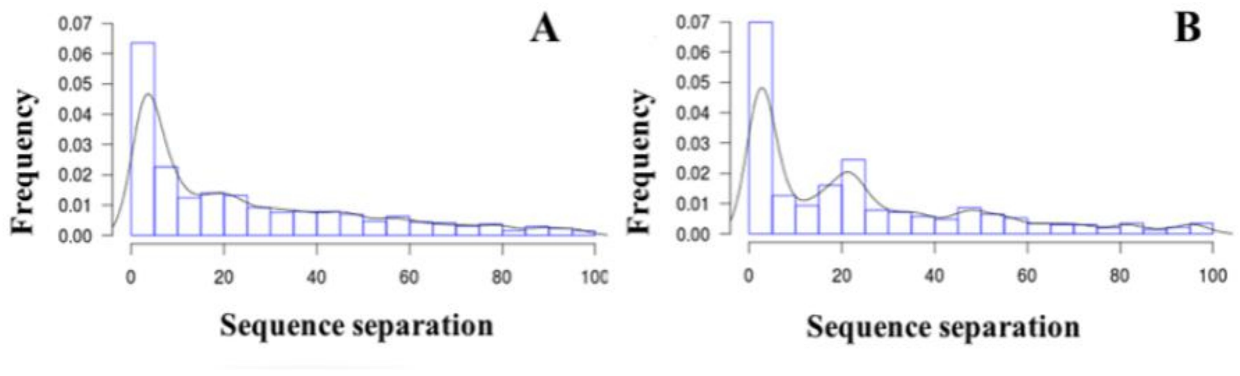

Figure 4.1: Distribution of salt bridges in primary structure. A: Glu based salt bridges, B: Asp based salt bridges.

Table 4.1: The percentage of local salt bridges against total salt bridges

Sequence separation	**Arg-Glu**	**Arg-Asp**	**Lys-Glu**	**Lys-Asp**
<=5	30.23	33.11	37.34	37.61

of sequence separation at a short sequence separation, showing a strong propensity for local salt salt bridges. The frequency of these local salt bridges, whose sequence separation <5, among all salt bridges is presented in Table 4.1. We can see that approximately one third of all salt bridges were local salt bridges. On further examination, we see that even at a small sequence separation, the distribution of salt bridges have particular biases. As seen in Fig. 4.2(A-D), the maximum frequency of local Arg-Glu salt bridge was at a separation of five residues, while a majority of local Arg-Asp salt bridges occur at a sequence separation of three or four residues. In both types of Lys based salt bridges, most of local salt bridges are found at a sequence separation of four.

A survey of the secondary structure elements of salt bridges, which has been used for the rational design of salt bridges, was investigated in this study and has been presented in Table 4.2. The results demonstrate that the residues in salt bridges tended to appear at same secondary structures.

Since local salt bridges consist of residues very near each other, their secondary structure statistics can be interpreted in the context of sequence separation. The results of this analysis are presented in Fig. 4.3. The results show that local salt bridges dominantly appeared at sequence separation of three, four and five in helix structure. It is evident that the content of secondary structures in

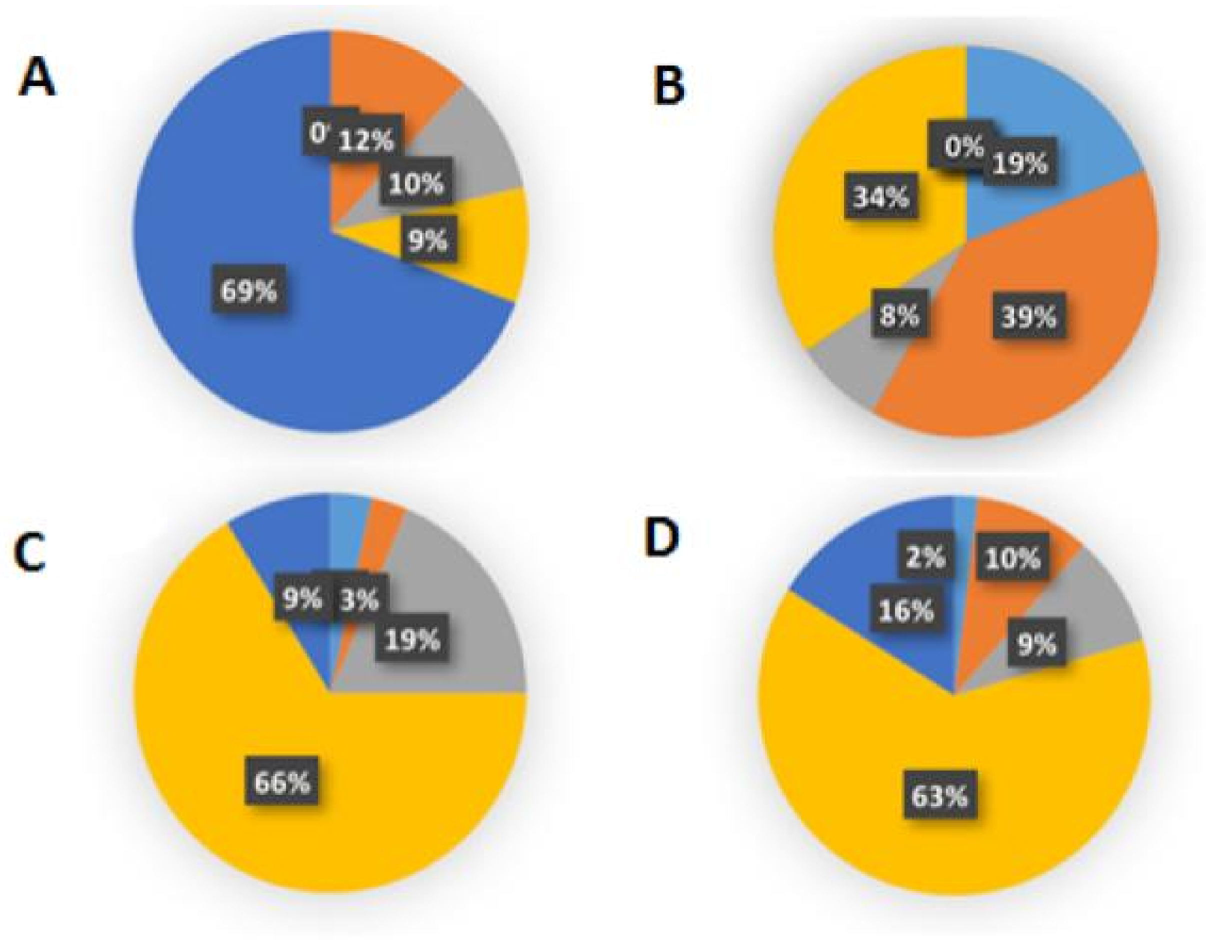

Figure 4.2: A: local Arg-Glu salt bridge, B: local Arg-Asp salt bridge, C: local Lys-Glu salt bridge, D: local Lys-Asp salt bridge. The sequence differences, 1: (light blue), 2: (orange), 3: (gray), 4: (yellow), 5: (dark blue), between two amino acids in protein sequence.

Table 4.2: Percentage of salt bridges with both residues in the same secondary structure

Both residues in same secondary structures	**Arg-Glu**	**Arg-Asp**	**Lys-Glu**	**Lys-Asp**
<=5	44.0	30.6	35.20	24.1

Lys based salt bridges is distinct from Arg based salt bridges. However, the two residues in Lys based salt bridges still tend to occur in same types of secondary structures. Our results also show that Arg and Lys tend to locate in helix structure in local salt bridges, in which Glu tended to locate in helix or coil structures. However, Asp had not shown any preference toward either helix or coil structures.

This survey of distribution characteristics of sequence separation, secondary structure preferences, and previously reported geometrical constraints for C-C and N-O distances of salt bridges lead us in the direction of design local salt bridges.

4.3.2 Substitutions at salt bridge positions

In this analysis, we divided salt bridges into two groups based on the identity of the positive residue: Arg based salt bridge, and Lys based salt bridges. In Fig. 4.4 we present plots of the rates of substitutions of salt bridges with other residues in homologous sequences. The blue chart shows the percentage of substitutions to other amino acids occurring at the positions of positively charged residue in salt bridges, while the grey chart presents the rates of possible substitutions at the positions of negatively charged residue in the same type of salt bridges. As shown in Fig. 4.4, the amino acids are most frequently conserved, indicating significant evolutionary pressure to conserve these interactions. One exception is the Lys-Glu salt bridge in which the Glu is substituted to Asp at a small, but significant frequency (10%).

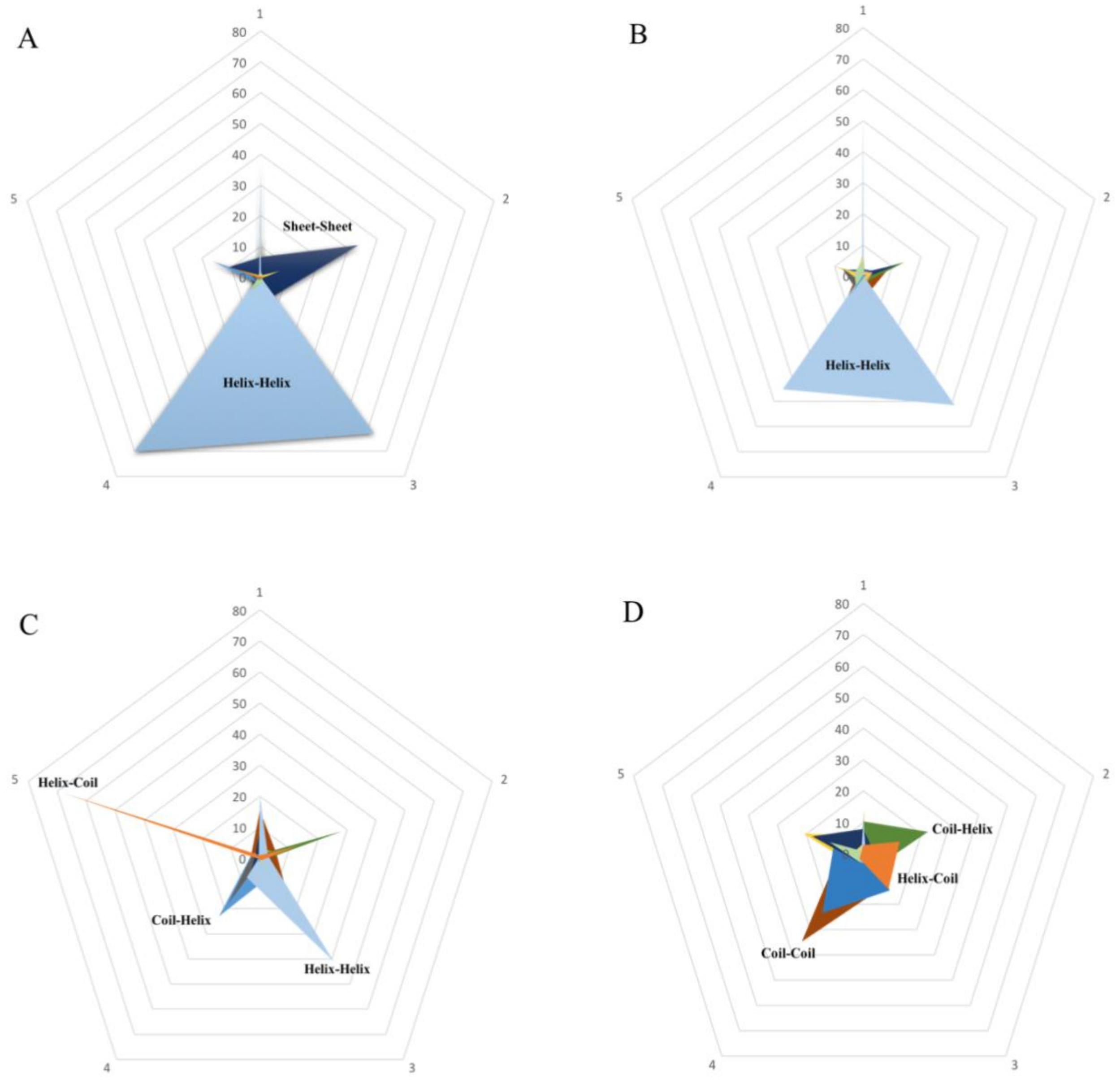

Figure 4.3: Radar Chart of of secondary structure elements in local salt bridges at sequence separation 1-5.A: Arg-Glu, B: Lys-Glu, C: Arg-Asp, D: Lys-Asp.

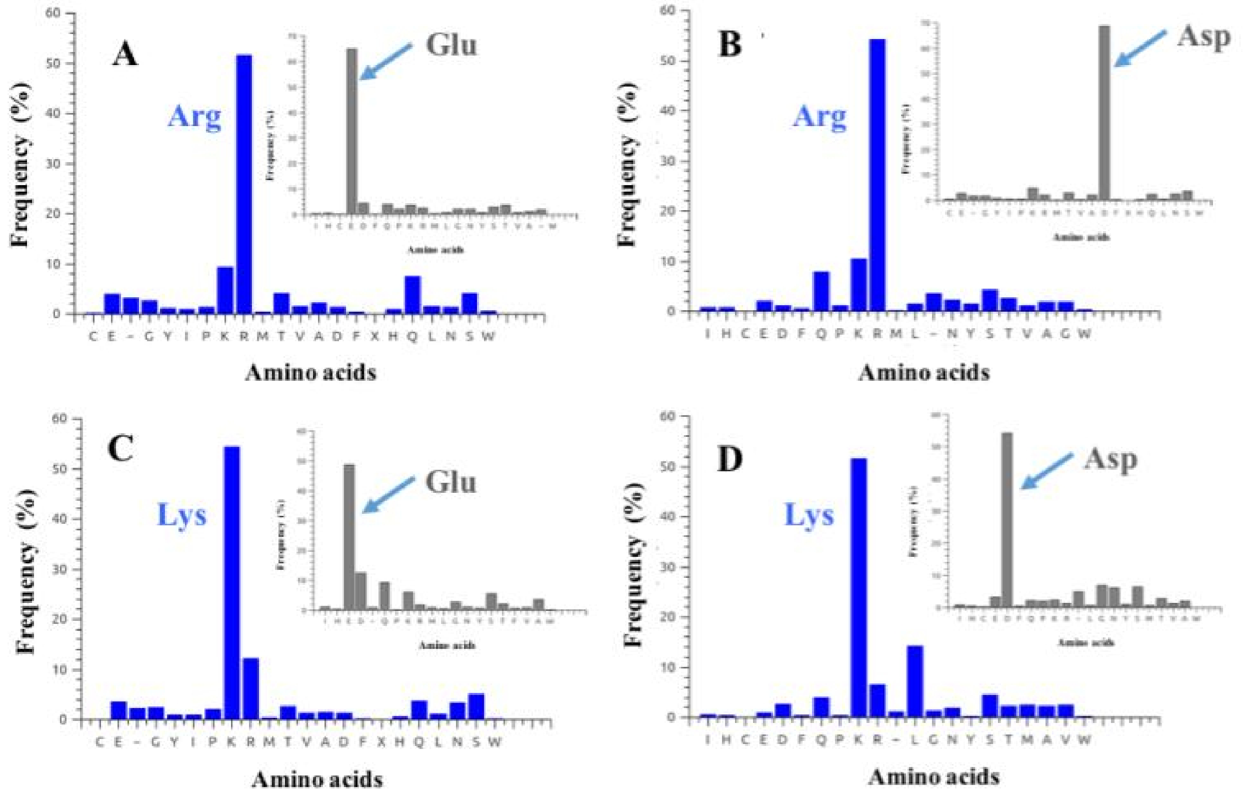

Figure 4.4: Frequency of substitutions at given positions of salt bridges. (A) Arg-Glu; (B) Arg-Asp; (C) Lys-Glu; (D) Lys-Asp. The frequency of possible substitutions at, Arg position (A-blue), Glu position (A-gray), in Arg-Glu salt bridge; the frequency of possible substitutions at, Arg position (B-blue), Asp position (B-gray) in Arg-Asp salt bridge; the frequency of possible substitutions at, Lys position (C-blue), Glu position (C-gray), in Lys-Glu salt bridge; the frequency of possible substitutions at, Lys position (D-blue), Asp position (D-gray), in Lys-Asp salt bridge.

4.3.3 Pairwise evolution of salt bridges

We next analyzed the pairwise substitutions of salt bridges when one of the positions is conserved. We categorized the substitutions at the second position into three categories-

1) conserved

2) substitution to a charged amino acid of the same polarity

3) substituted to a neutrally charged residue.

The results of this analysis are presented in Fig. 4.5. We see that there is significant evolutionary pressure to conserve a salt bridge (category 1, 2). However, 20% of homologs fall in the category 3, implying that the loss of a charged residue at one of the salt bridge positions does not always lead to loss of the corresponding oppositely charged residue. This indicates that charged amino acids are interchangeable to some extent in salt bridges and there is some flexibility in the design of salt bridges. Earlier reports examining the energy of salt bridges through mutation studies have shown that flipping the salt bridge, substituting one charged residue with another of the same polarity can change the energy/thermostability of the bond (Makhatadze et al. 2003; Lee et al. 2014). Thus, salt bridges could part be a mechanism to make fine adjustments to thermostability of proteins.

4.3.4 Effects of salt bridge on local evolutionary stability

Considering that there is a significant co-evolution in salt bridge interactions, we investigated the conservation of residues in the local sequence context of salt bridge positions. Since around one-third of all salt bridges are at a small sequence separation (Table 4.1), we restricted the inquiry to 3 residues upstream and downstream of the each of the residues constituting the bridge. Fig. 4.6, summarizes the results of this analysis. We see that even when at least one of the salt bridge residues is conserved, the local sequence around the conserved residue as well as the local sequence around the other salt bridge residue are 70% conserved each, while this score drops to 40% when that residue in the salt bridge is not conserved. We can infer that there is correlation between salt bridge conservation and local sequence conservation. Previous results have already shown important differences in the preferred conformation for different types of salt bridges (Donald, et al. 2011). Hence the results in Fig. 4.6 indicate the existence of sequence motifs that contextualize different types of salt bridges.

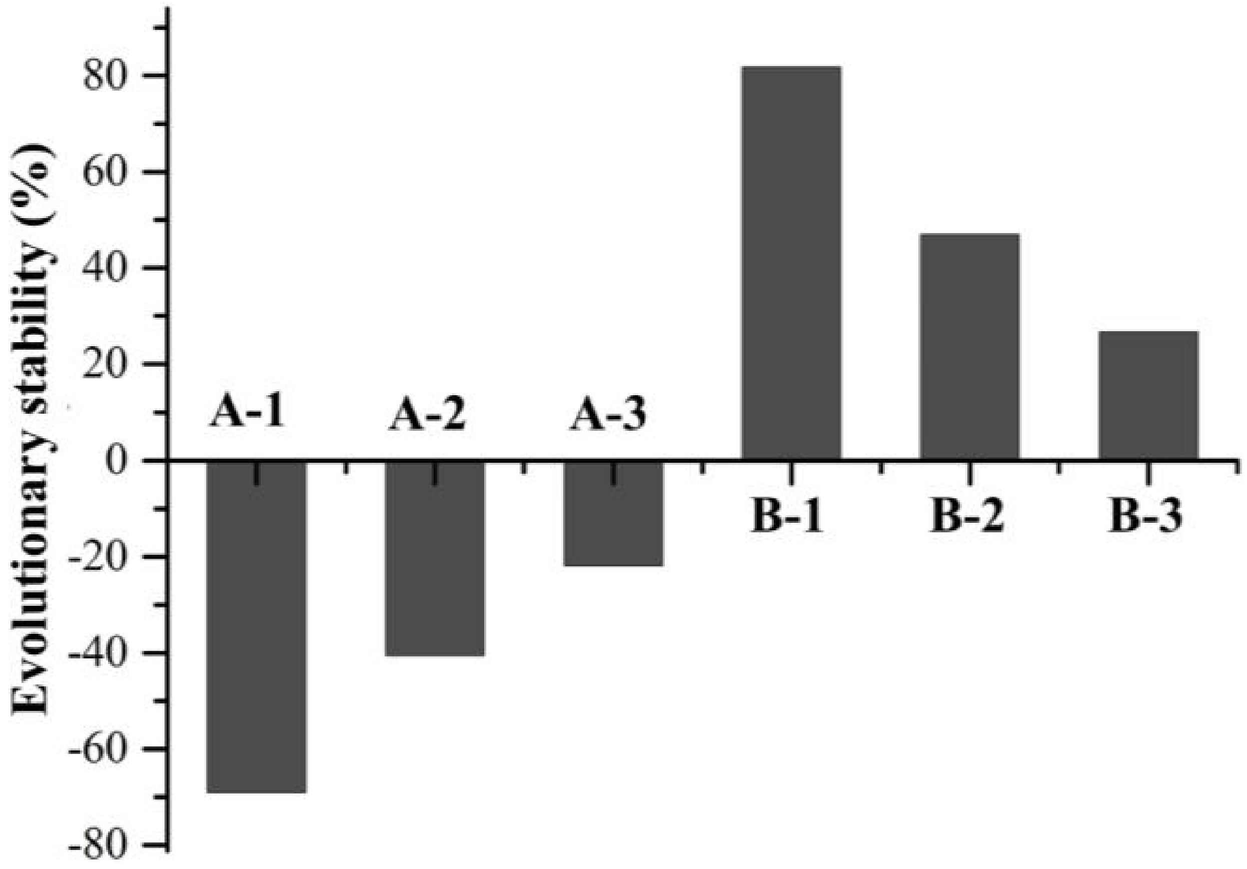

Figure 4.5: Effects of directed substitutions at salt bridges on evolutionary stability. The vertical axis shows percentage of identical residues at divergent branches. The percentage of identically negative residues (A-1), positive residues (B-1), of salt bridges against total BLAST results when the other part of salt bridges is conservative; The percentage of identically negative residues (A-2), positive residues (B-2), of salt bridges against total BLAST results when the other part of salt bridges evolve to other similar residues; The percentage of identically negative residues (A-3), positive residues (B-3), of salt bridges against total BLAST results when the other part of salt bridges evolve to other neutral residues.

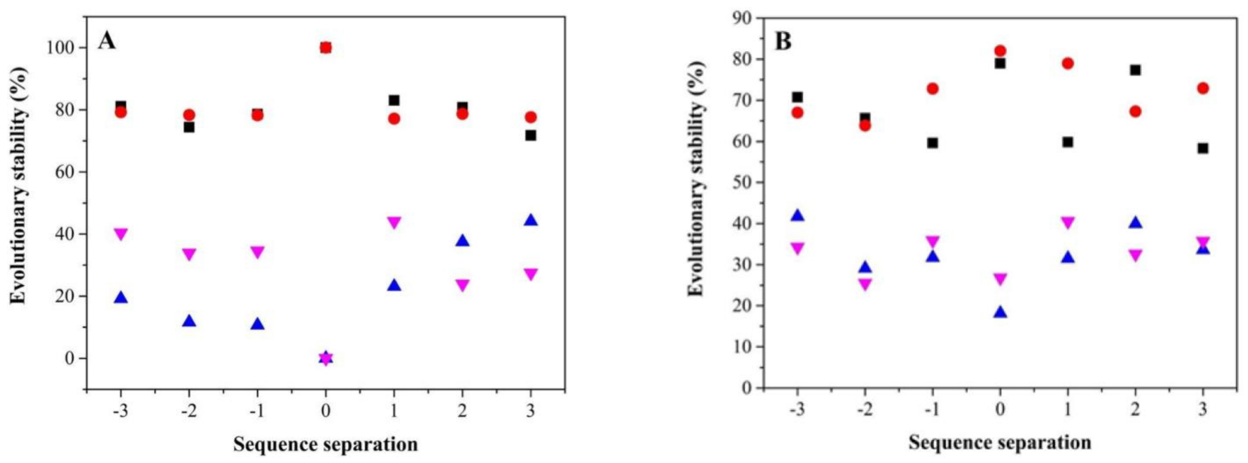

Figure 4.6: Regional evolutionary stability. (A) (□) Arg based salt bridges; (○), Lys based salt bridges, when one part of salt bridges was conservative. (△), Arg based salt bridges; (∇), Lys based salt bridges, when one part of salt bridges was unconservative. (B)Other part of (□), Arg based salt bridges; (○), Lys based salt bridge.

4.3.5 Relationship of evolutionary stability between parts of salt bridges

We further investigated the effect of the presence of salt bridges on their local sequence context via the ET scores which not only measures the frequency of variations, but, also how close these variations occur to the root of the phylogenetic tree. In our study, the results showed that strong correlations of ET scores were observed between local sequence contexts of the positive parts and negative parts of salt bridges (Fig. 4.7). These results show show evidence for the coordinated evolution of the local sequence context around salt bridge residues and hints that alternation of one residue in the salt bridge context may result in variations in the other corresponding part of salt bridges.

4.3.6 Rational design of salt bridges

Salt bridges are capable of increasing the evolutionary stability of proteins around their local sequence context, suggesting that the electrostatic interactions improve the contribution of amino acids around salt bridges to the stability of proteins. Additionally, local sequence context around one residue of salt bridges could affect the other part of salt bridge in terms of evolutionary

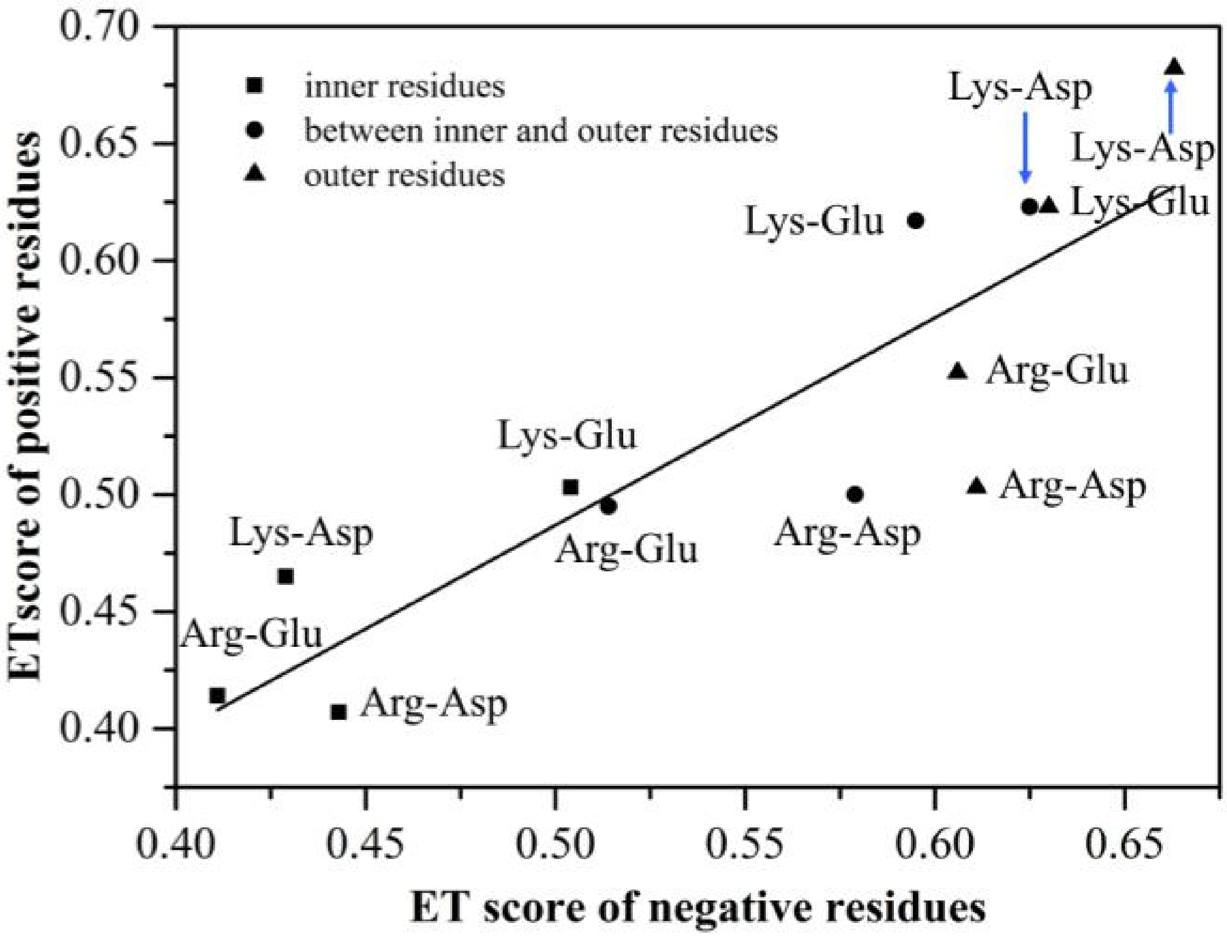

Figure 4.7: Relationship of ET scores between residues in salt bridges.

stability. According to the analysis of evolutionary characteristics of salt bridges, salt bridges that may significantly improve stability are likely to have conserved local sequence contexts, or at least one of the residues of the salt bridges does. The local sequence context has a direct effect on the conformation of salt bridge residues. Thus, a conserved local sequence context indicates the need for precise geometry- either for stability or function. Incorporating such evolutionary information is likely vital for rational design of salt bridges.

Based on above analysis, salt bridges should be designed following the two rules. First, in accordance with geometrical distribution of salt bridges, the distance between the C-C carbons should be < 15 Å ; Second, both the positions should have the same secondary structure; Third, the local sequence context around at least one part of salt bridges should be conserved $> 70\%$. These design heuristics were tested experimentally by a collaborator. To experimentally validate this procedure, the 1,4–α–glucan branching enzyme (GBE; EC 2.4.1.18) from G. thermoglucosidans STB02 (GenBank accession no. KJ660983) was used. GBE is a glycoside-transferase belonging to glycosyl hydrolase family 13 which reacts with α–(1,4) and/or α–(1,6) glucosidic linkages and subsequently synthesizes α–1,6-glucosidic bonds. Based on the structure of G. thermoglucosidans STB02 GBE, 8 mutants at 5 positions of GBE were constructed by directed-site mutagenesis. The characteristics of evolution and structural information of mutant GBE were listed in Table 4.4. Our collaborators report, that all mutants except at the one at I266, which did not have a conserved local sequence context improved thermostability.

4.4 Discussion

The aim of this research is to take advantage of evolutionary information to help select potential sites for mutation to charged residues such that the newly formed salt bridges will have significantly positive effects on stability of proteins.

Our study shows that charged amino acids appear more frequently than neutral amino acids at salt bridge positions along the phylogenetic tree, suggesting that salt bridges are more conserved

Table 4.3: The conservative analysis of possible salt bridges

Secondary Structure	**Res 1$^{\alpha}$**	**Local Sequence Context$^{\beta}$**	**Res 2$^{\alpha}$**	**Cons/Non-Cons**	**C_{α} dist (Å)$^{\gamma}$**
	H224	nc	R170	c	6.8
α–helix	Q231	nc	D227	c	6.2
	V37	c	K32	nc	4.3
β–sheet	I571	c	R569	c	5.7
Random coil	I266	nc	R269	nc	8.0

$^{\alpha}$: Residues of salt bridges.
$^{\beta}$: nc, non-conservative; c, conservatives $>$ 70%.
$^{\gamma}$: Distance between residue 1 and residue 2.

than non-electrostatic interactions. Since natural selection is supposed to eliminate redundant parts of structures in the process of biological evolution (Shionyu-Mitsuyama, et al. 2005), the importance of salt bridges raises another question: how do salt bridges affect the evolutionary stability of proteins?

The strong correlation of ET scores between two parts of salt bridges demonstrates that the conservation of residues in salt bridges are tightly correlated (Fig. 4.7). Therefore, the influences of salt bridges on evolutionary stability of corresponding amino acid regions was carried out by counting the ratio of identical residues within a sequence separation of 3 from salt bridge residues against all possible substitutions at these positions along the phylogenetic tree. The results reveal that salt bridges are able to increase the evolutionary stability of related amino acid regions. Contrastingly, breakdown of these electrostatic interactions results in decline of regional evolutionary stability. The results hint that the formation of salt bridges not only offers additional electrostatic bonds but also contributes to the stability of the regions around them.

Based on survey of bioinformatics and evolutionary stability of salt bridges, 8 mutations, H244D, H244E, Q231R, Q231K, V37E, V37D, I571D and I266E, were constructed according to structural information of G. thermoglucosidans STB02 GBE by our collaborators. Except for I266E, the rest of 7 mutations, which consist of at least one residue with a conserved local sequence context, prolong half-time of GBE compared with that of wild type GBE.

However, several factors are related with thermostability of proteins, which cannot be summarized in one rule. The investigation of evolutionary stability of salt bridges also offers us a new way of understanding the contribution of salt bridges. In addition, the evolutionary survey provides us a new approach to rationally design of salt bridges with significant contribution to stability of proteins.

Appendix A

Plugin Experimental Results

Table A.1: Results of MD Simulations on mutations suggested by Plugin

PDB ID	**Plugin Potential Mutations**	**Stable Plugin Mutations**
	AA153C;SA219C	
	SA99C;VA242C	
	AA54C;AA77C	
	GA81C;GA82C	
	SA105C;YA114C	
	TA169C;GA233C	
	AA77C;LA90C	
	AA116C;LA148C	
	SA56C;FA179C	
	SA105C;EA117C	
	LA57C;RA58C	
	TA55C;DA71C	
	DA34C;TA37C	
	KA170C;GA233C	
	GA63C;KA170C	
	HA17C;AA23C	
	YA114C;FA147C	
	YA128C;KA133C	
	KA154C;LA157C	
	NA232C;EA236C	
	QA28C;QA249C	
	FA147C;PA215C	
	PA186C;EA214C	TA169C;GA233C
2CBA	HA96C;WA245C	DA34C;TA37C

* ———Continued On Next Page——— *

PDB ID	Plugin Potential Mutations	Stable Plugin Mutations
1SNO	AA69C;AA94C RA105C;SA128C RA35C;RA87C IA18C;TA22C AA17C;KA63C HA46C;GA50C TA13C;MA26C DA77C;NA118C NA119C;EA122C HA46C;KA49C KA9C;EA73C EA43C;HA46C KA78C;PA117C MA98C;EA101C PA42C;EA52C	A43C;HA46C
2LZM	SA38C;AA41C SA36C;AA41C TA34C;SA36C GA28C;AA63C TA151C;AA160C RA148C;AA160C DA20C;GA23C KA83C;AA112C TA59C;DA61C KA19C;GA23C; MA120C;AA129C SA136C;WA138C DA20C;YA24C RA125C;EA128C IA58C;EA62C DA20C;EA22C	None

* ———Continued On Next Page——— *

PDB ID	Plugin Potential Mutations	Stable Plugin Mutations
1XNB	GA62C;SA176C GA64C;GA173C GA23C;GA24C GA13C;GA14C GA12C;GA13C GA102C;GA103C AA55C;TA138C VA98C;AA152C TA33C;GA34C SA100C;NA148C; RA112C;AA115C IA15C;SA31C GA92C;YA108C YA65C;TA67C VA57C;NA181C TA10C;FA36C RA73C;NA163C DA83C;RA136C PA60C;GA86C NA17C;NA29C WA71C;AA165C; WA58C;TA138C YA108C;WA129C	None
1RNB	GA52C;AA74C GA52C;GA53C GA9C;DA12C AA46C;KA49C LA20C;YA24C KA66C;RA69C DA101C;QA104C IA25C;EA29C IA4C;PA21C	GA9C;GA53C
9RAT	TA87C;AA96C AA4C;VA118C AA20C;YA25C SA50C;DA53C KA104C;SA123C HA48C;SA80C DA83C;TA100C AA109C;HA119C	TA87C;AA96C;

* ———Continued On Next Page——— *

PDB ID	Plugin Potential Mutations	Stable Plugin Mutations
	SA168C;SA268C	
	SA52C;AA59C	
	AA252C;SA318C	
	AA50C;AA136C	
	AA1C;AA12C	
	GA60C;AA136C	
	GA209C;AA223C	
	GA119C;AA299C	
	GA61C;GA62C	
	GA60C;GA61C	
	GA159C;GA160C	
	VA73C;AA135C	
	TA193C;TA196C	
	SA101C;NA131C	
	AA103C;RA129C	
	AA111C;LA123C	
	SA52C;QA55C	
	NA260C;AA304C	
	SA232C;HA340C	
	NA88C;GA355C	
	AA79C;IA141C	
	EA39C;SA86C	
	SA232C;MA346C	
	IA51C;GA62C	
	DA84C;AA357C	
	SA115C;PA296C	
	FA70C;AA135C	
	AA120C;FA197C	
	QA55C;AA59C	
	LA126C;LA286C	
	DA64C;TA133C	
	GA220C;QA240C	
	QA145C;GA220C	
	FA306C;TA329C	
	PA144C;GA220C	
	GA211C;PA227C	
	PA121C;RA270C	
	LA226C;EA236C	
	LA207C;PA225C	
	FA155C;FA161C	SA168C;SA268C
	NA218C;EA221C	AA1C;AA12C
	EA261C;FA264C	LA226C;EA236C
1MNP	PA189C;PA194C	NA218C;EA221C

* ———Continued On Next Page——— *

PDB ID	Plugin Potential Mutations	Stable Plugin Mutations
	SA164C;GA456C	
	GA383C;SA386C	
	GA339C;SA365C	
	TA290C;SA405C	
	VA37C;SA56C	
	GA23C;AA32C	
	GA23C;AA24C	
	AA300C;GA383C	
	GA96C;GA103C	
	GA101C;GA103C	
	SA99C;YA458C	
	TA72C;AA471C	
	AA32C;VA111C	
	TA53C;GA127C	
	TA173C;GA183C	
	LA319C;AA421C	
	DA176C;SA185C	
	NA93C;AA115C	
	SA95C;EA106C	
	SA399C;EA409C	
	EA400C;SA411C	
	YA329C;RA428C	
	SA284C;PA307C	
	DA403C;GA407C	
	DA309C;GA314C	
	NA427C;VA432C	
	FA351C;AA358C	
	KA61C;AA134C	
	DA126C;TA188C	
	GA35C;HA80C	
	DA245C;YA306C	
	PA272C;VA346C	
	PA258C;VA433C	
	WA212C;AA454C	
	YA232C;EA280C	
	WA52C;DA55C	
	EA198C;WA437C	
3GLY	WA28C;PA41C	TA72C;AA471C

* ———Continued On Next Page——— *

PDB ID	**Plugin Potential Mutations**	**Stable Plugin Mutations**
	CA17C;VA28C	
	SA40C;VA57C	
	AA51C;GA77C	
	GA83C;GA84C	
	GA23C;GA24C	
	IA75C;TA78C	
	NA70C;KA73C	
1CHH	KA5C;DA93C	None
	RI62C;RI65C	
2CI2	KI43C;AI46C	None
	GA34C;GA35C	
	GA33C;GA34C	
	SA141C;MA157C	
	DA36C;TA72C	
	FA112C;RA148C	
	DA36C;DA74C	
	EA159C;QA162C	
4TNC	PA53C;EA57C	DA36C;TA72C

Appendix B

DbD2 Experimental Results

Table B.1: Results of MD Simulations on mutations suggested by DbD2

PDB ID	**DbD2 Potential Mutations**	**Stable DbD2 Mutations**
2CBA	HA17C;AA23C; QA28C;QA249C; PA30C;EA106C; SA48C;GA81C; AA54C;AA77C SA56C;FA179C LA57C;RA58C IA59C;FA176C GA63C;KA170C AA77C;LA90C YA88C;HA122C HA96C;WA245C WA97C;AA116C GA98C;QA103C SA99C;VA242C SA105C;EA117C AA116C;LA148C WA123C;AA134C YA128C;KA133C YA128C;AA134C AA134C;GA140C GA145C;VA211C FA147C;PA215C GA151C;SA217C AA153C;SA219C KA154C;LA157C PA186C;EA214C LA229C;MA241C NA232C;EA236C	None

* ———Continued On Next Page——— *

PDB ID	DbD2 Potential Mutations	Stable DbD2 Mutations
1SNO	KA9C;EA73C IA15C;KA24C AA17C;KA63C IA18C;TA22C RA35C;RA87C VA39C;AA109C TA41C;AA58C PA42C;EA52C EA43C;HA46C HA46C;KA49C EA52C;GA55C AA58C;LA108C AA69C;AA94C KA78C;PA117C MA98C;EA101C LA103C;AA109C GA107C;AA132C GA107C;NA138C NA119C;EA122C	GA63C;KA170C;
2LZM	DA20C;YA24C YA25C;AA42C GA28C;AA63C LA33C;EA45C TA34C;SA36C SA36C;AA41C SA38C;AA41C IA58C;EA62C NA81C;LA84C KA83C;AA112C MA120C;AA129C AA130C;RA154C SA136C;WA138C RA148C;AA160C TA151C;AA160C	None

* ———Continued On Next Page——— *

PDB ID	**DbD2 Potential Mutations**	**Stable DbD2 Mutations**
	TA10C;FA36C	
	GA14C;NA32C	
	GA21C;NA25C	
	GA24C;VA184C	
	GA39C;VA168C	
	SA46C;RA49C	
	AA55C;TA138C	
	VA57C;NA181C	
	LA66C;SA84C	
	LA68C;VA81C	
	YA69C;AA170C	
	GA70C;QA167C	
	WA71C;AA165C	
	RA73C;NA163C	
	DA83C;RA136C	
	GA92C;YA108C	
	GA92C;TA110C	
	KA95C;IA107C	
	VA98C;AA152C	
	SA100C;NA148C	
	YA105C;SA130C	
	RA112C;AA115C	
1XNB	WA153C;MA158C	None
	IA4C;GA9C	
	IA4C;PA21C	
	LA20C;YA24C	
	YA24C;GA52C	
	IA25C;EA29C	
	AA30C;WA35C	
	GA34C;VA45C	
	AA46C;KA49C	
	FA56C;WA71C	
	KA66C;RA69C	
	IA76C;DA86C	
1RNB	DA101C;QA104C	None

* ———Continued On Next Page——— *

PDB ID	DbD2 Potential Mutations	Stable DbD2 Mutations
9RAT	AA20C;YA25C AA20C;TA82C CA26C;CA84C CA26C;TA99C CA40C;CA95C HA48C;SA80C SA50C;DA53C CA58C;CA110C CA65C;QA69C CA65C;CA72C CA84C;YA97C TA87C;AA96C KA104C;SA123C VA108C;PA117C	TA87C;AA96C

* ———Continued On Next Page——— *

PDB ID	DbD2 Potential Mutations	Stable DbD2 Mutations
	AA1C;AA12C	
	CA3C;CA15C	
	CA14C;CA289C	
	CA33C;CA117C	
	EA35C;GA350C	
	AA37C;AA113C	
	EA39C;SA86C	
	LA43C;SA86C	
	AA48C;AA63C	
	AA50C;GA60C	
	AA50C;AA136C	
	IA51C;GA62C	
	SA52C;QA55C	
	SA52C;AA59C	
	QA55C;AA59C	
	GA61C;DA64C	
	DA64C;TA133C	
	GA65C;AA103C	
	PA71C;PA356C	
	VA73C;AA135C	
	AA79C;IA141C	
	DA84C;AA357C	
	VA87C;AA357C	
	SA101C;NA131C	
	AA103C;RA129C	
	AA111C;LA123C	
	CA117C;AA120C	
	CA117C;FA197C	
	GA119C;FA197C	
	GA119C;AA299C	
	PA121C;RA270C	
	LA126C;LA286C	
	FA155C;FA161C	
	SA168C;SA268C	
	SA168C;AA272C	

* ———Continued On Next Page——— *

PDB ID	DbD2 Potential Mutations	Stable DbD2 Mutations
	DA182C;GA214C	
	IA185C;GA214C	
	AA188C;MA237C	
	PA189C;PA194C	
	DA191C;QA200C	
	DA191C;VA201C	
	DA191C;VA201C	
	TA193C;TA196C	
	LA207C;PA225C	
	GA211C;PA227C	
	PA213C;GA235C	
	NA218C;EA221C	
	LA226C;EA236C	
	SA232C;MA346C	
	AA252C;SA318C	
	CA253C;CA319C	
	NA260C;AA304C	
	EA261C;FA264C	AA1C;AA12C
	FA306C;TA329C	NA218C;EA221C
1MNP	CA341C;CA348C	LA226C;EA236C

* ———Continued On Next Page——— *

PDB ID	DbD2 Potential Mutations	Stable DbD2 Mutations
	VA13C;GA396C	
	AA14C;YA419C	
	GA23C;IA36C	
	WA28C;PA41C	
	AA32C;VA111C	
	VA37C;SA56C	
	SA40C;YA48C	
	WA52C;DA55C	
	KA61C;AA134C	
	VA64C;GA137C	
	LA66C;DA71C	
	TA72C;AA471C	
	NA93C;AA115C	
	SA95C;EA106C	
	GA96C;SA100C	
	SA99C;YA458C	
	GA121C;YA175C	
	PA128C;DA162C	
	AA129C;AA195C	
	GA139C;VA155C	
	LA142C;AA151C	
	LA163C;GA199C	
	AA167C;WA212C	
	DA176C;SA185C	
	AA190C;GA251C	
	HA193C;IA220C	
	LA196C;AA217C	
	EA198C;WA437C	
	SA200C;CA213C	
	AA203C;CA210C	
	CA210C;CA213C	
	WA212C;AA454C	
	CA222C;CA449C	
	YA232C;EA280C	
	IA233C;TA248C	
	DA245C;YA306C	
	DA257C;AA260C	
	DA257C;EA439C	
	PA258C;VA433C	
	CA262C;CA270C	
	PA269C;DA330C	
	CA270C;AA331C	
	PA272C;VA346C	
	LA275C;SA347C	
	SA284C;PA307C	

* ———Continued On Next Page——— *

PDB ID	DbD2 Potential Mutations	Stable DbD2 Mutations
3GLY	TA290C;SA405C LA295C;EA299C AA302C;PA316C DA309C;GA314C WA317C;CA320C LA319C;AA421C AA325C;FA384C LA328C;AA381C YA329C;RA428C FA351C;AA358C HA391C;GA407C LA398C;SA418C SA399C;EA409C EA400C;SA411C NA427C;VA432C	TA72C;AA471C;
1CHH	DbD2 does not accept the PDB file	None
2CI2	WI24C;LI27C KI43C;AI46C RI62C;RI65C	None
4TNC	DA36C;TA72C QA51C;QA85C PA53C;EA57C FA112C;RA148C EA159C;QA162C	DA36C;TA72C;

Appendix C

PyMol Plugin

```
#
# -- Pymol script to analyze arbitrary protein for potential ssbonds
#

from pymol import cmd, stored
import os.path
import subprocess as sp
from scipy.spatial.distance import cdist
import re
import numpy as np
import time

def getPDBFile(pdb):
    '''
    DESCRIPTION

    This function looks to see if given PDB is already on disk.
    It will load the file in Pymol from disk, alternatively
    it will download it from RCSB.

    It will also remove hets and display the molecule as a cartoon

    PARAMS
    string pdbid


    OUTPUT
    Produces a selection object containing all the amino acid residues.
    Reinitializes and creates     OUTPUT on PyMol graphic device.
    '''
    cmd.reinitialize()

    file = './%s.pdb' % pdb
    if os.path.exists(file):
        cmd.load(file)
    else:
        cmd.fetch(pdb)

    cmd.remove('het')
    cmd.hide(representation = "lines", selection = "all")
    cmd.show(representation = "cartoon", selection = "all")
    cmd.cartoon('loop', 'all')

    cmd.select('residues', '(byres name ca)')



def findSurfaceResidues(objSel="(all)", cutoff=2.5,
                        doShow=False, verbose=False):
    """
    findSurfaceResidues
        finds those residues on the surface of a protein
        that have at least 'cutoff' exposed A**2 surface area.
```

```
PARAMS

    objSel (string)

        the object or selection in which to find

        exposed residues

        DEFAULT: (all)


    cutoff (float)

        your cutoff of what is exposed or not.

        DEFAULT: 2.5 Ang**2


    asSel (boolean)

        make a selection out of the residues found


RETURNS

    (list: (chain, resv ) )

        A Python list of residue numbers corresponding

        to those residues w/more exposure than the cutoff.


Code taken from FindSurfaceResidues example on PymolWiki

"""

tmpObj="__tmp"

cmd.create( tmpObj, objSel + " and polymer");

if verbose!=False:

    print "WARNING: I'm setting dot_solvent. \

            You may not care for this."

cmd.set("dot_solvent");

cmd.get_area(selection=tmpObj, load_b=1)
```

```
# threshold on what one considers an "exposed" atom (in A**2):
cmd.remove( tmpObj + " and b < " + str(cutoff) )

stored.tmp_dict = {}
cmd.iterate(tmpObj, "stored.tmp_dict[(chain,resv)]=1")
exposed = stored.tmp_dict.keys()
exposed.sort()

selName = "exposed_atoms"
if verbose!=False:
    print "Exposed residues are selected in: " + selName

cmd.select(selName, objSel + " in " + tmpObj )

selNameRes = "surface_resi"
cmd.select(selNameRes, "byres " + selName )

if doShow!=False:
    cmd.show_as("spheres", objSel + " and poly")
    cmd.color("white", objSel)
    cmd.color("red", selName)



cmd.select('disulfides', 'byres (cys/sg and bound_to cys/sg)')
cmd.select('residues', '(byres name ca & surface_resi)
                    and !disulfides')
cmd.select('carbons', '(residues and name cb) or
                    (gly/ca and residues)')
    cmd.show('sticks', '(disulfides and cys/ca+cb+sg)')
    cmd.show('spheres', '(disulfides and cys/ca+cb+sg)')
    cmd.set ("sphere_scale", 0.25, selection='(disulfides
                    and cys/ca+cb+sg)')

    cmd.delete(tmpObj)
    cmd.delete('exposed_atoms')
    cmd.delete('surface_residues')




def getDistanceMatrix():
    '''
    DESCRIPTION

    This function obtains the distance matrix of every beta carbon
    to every other beta carbon in the list. It does not care about
    whether this is a single or multichain protein.

    Note: For glycine

    PARAMS
    a selection object from pymol consisting of beta carbons.

    OUTPUT
    numpy matrix containing distance between beta carbons (falls back
    to alpha carbon distance for GLY)
    '''
    stored.betaCarbons = []

    cmd.iterate_state(1, pymol.selector.process('carbons'),
                "stored.betaCarbons.append([x,y,z])")

    Y = cdist(stored.betaCarbons, stored.betaCarbons)

    return(Y)



def getProbability(table, index):
    '''
    DESCRIPTION

    This function takes a index-value table and finds the
        highest value smaller than the given index.
        For this value it returns the corresponding probability

    PARAMS
    array table containing a list of two-tuples.
    float index

    OUTPUT
    float probability
    '''

    smallers = [y for (x,y) in table if x<= index]
    try:
            return smallers[-1]
        except:
            return 0.0


def getCandidateResidues(Y):
    '''
    DESCRIPTION

    Produces a list of candidate residues available for
    modification to CYS, along with an evolutionary score
    computed from a statistical distribution

    PARAMS
    distance matrix

    OUTPUT
    list of tuples containing ranked candidates
    '''

    escore = {  'ALA':  0.42982450,
                'ARG':  0.02808730,
                'ASN': -0.30177649,
                'ASP': -0.45340845,
                'CYS':  3.15118885,
                'GLN': -0.59681526,
                'GLU': -0.76325665,
                'GLY':  0.36711727,
                'HIS': -0.65986515,
                'ILE': -0.37885249,

                'LEU': -0.08448341,

                'LYS': -0.57031967,

                'MET': -0.77845348,

                'PHE': -0.52710314,

                'PRO': -0.86851000,

                'SER':  0.78266876,

                'THR':  0.25801733,

                'TRP': -1.15163219,

                'TYR': -0.07394938,

                'VAL':  0.21469455,

                '.':    1.97682717}



    stored.names = []

    stored.resnum = []

    stored.reschain = []

    print stored.tmp_dict

    cmd.iterate('carbons', 'stored.names.append(resn)')


    cmd.iterate('carbons', 'stored.resnum.append(resi)')


    cmd.iterate('carbons', 'stored.reschain.append(chain)')

    scores = []

    for i in range(0, len(Y)):

        for j in range(i+1, len(Y)):


            dist = Y[i][j]

            if dist < 4.25 and dist > 3.5:
                id1 = cmd.identify('carbons', 0)[i]
                id2 = cmd.identify('carbons', 0)[j]

                score = escore[stored.names[i]] +
                            escore[stored.names[j]]
                scores.append((score, (stored.names[i],
                            stored.resnum[i], stored.reschain[i],
                            stored.names[j], stored.resnum[j],
                            stored.reschain[j])))


                cmd.select('cys1', 'byres id ' + str(id1))
                cmd.show('sticks', 'cys1 and ! name c+n')
                cmd.label('cys1 and name ca', '"%s-%s" % (resn, resi)')

                cmd.select('cys2', 'byres id ' + str(id2))
                cmd.show('sticks', 'cys2 and ! name c+n')
                cmd.label('cys2 and name ca', '"%s-%s" % (resn, resi)')

    scores.sort(reverse=True)

    cmd.delete('cys1')
    cmd.delete('cys2')

    return(scores)
```

```
def highlightCandidates():
    '''
    DESCRIPTION

    Highlight all possible pairs of residues that might be
        candidates for disulfide bonds.

    Currently will highlight everything within a fixed
        distance that have positive E-scores.
    '''
    cmd.show('spheres', 'residues')
    cmd.set ("sphere_scale", 0.25, selection='residues')
    cmd.color('yellow', 'residues')
    cmd.color('red', 'carbons')



def mutate(selection,mutframe):
    '''
    DESCRIPTION

    Function adapted from rotkit.py

    PARAMS
    string selection
    int mutframe

    OUTPUT
    rotamer probability
    '''
    cmd.wizard("mutagenesis")
    cmd.do("refresh_wizard")
    cmd.get_wizard().set_mode("CYS")
    cmd.get_wizard().do_select('('+selection+')')
    cmd.frame(mutframe)
    title = cmd.get_title('mutation', 1)
    print title
    prob_rot = None
    if title:
        prob_rot = float(title[:-1])/100.0
        cmd.get_wizard().apply()
        cmd.set_wizard()
    return prob_rot


def evaluateForFit(potentials):
    '''
    DESCRIPTION

    This function takes a list of residue pairs, mutates each one to
    one of the three rotamers of CYS, and evaluates these modifications
    for potential SSBond features.

    PARAMS

    A list of potentials of the form
        (escore, (cys1, cys1num, cys2, cys2num))
```

```
'''
with open('./bondlength.csv', 'rb') as tmp:
    lines = tmp.readlines()
lines = [line.strip().split(',') for line in lines]
length_table = [(float(line[0]), float(line[1])) for line in lines]

with open('./chi3density.csv', 'rb') as tmp:
    lines = tmp.readlines()
lines = [line.strip().split(',') for line in lines]
angle_table = [(float(line[0]) * 57.2958, float(line[1]))
                    for line in lines]
### 57.2958 = 180 / pi



all_candidates = []


for pair in potentials:
    score, (cys1, cysnum1, cys1chain,
                cys2, cysnum2, cys2chain) = pair


    cmd.select("Cys1", 'resi ' + str(cysnum1))
    cmd.select("Cys2", 'resi ' + str(cysnum2))
            #cys1chain = cmd.get_chains("Cys1")[0]
            #cys2chain = cmd.get_chains("Cys2")[0]
    candidates = {'Residues' : ' %s: %s (%s) -- %s:
            %s (%s) ' % (cys1chain, cys1, cysnum1,
                    cys2chain, cys2, cysnum2)}


    candidates['ETscore'] = score
```

```
print cys1chain, cys1, cysnum1, cys2chain, cys2, cysnum2
cmd.select("Cys1", 'resi ' + str(cysnum1))
cmd.select("Cys2", 'resi ' + str(cysnum2))
cmd.select('Pair', 'Cys1 or Cys2')
cmd.center('Pair')
cmd.orient('Pair')
cmd.zoom('Pair', buffer = 3.0)

cmd.refresh()

for i in range(0,3):
    cmd.select("Cys1", 'resi ' + str(cysnum1) +
               ' and chain '+cys1chain)
    prob_cys1 = mutate('(Cys1)', i+1)
    cmd.refresh()



    for j in range(0,3):

        print i,j
        cmd.select("Cys2", 'resi ' + str(cysnum2) +
                   ' and chain '+cys2chain)
        prob_cys2 = mutate('(Cys2)', j+1)
        cmd.refresh()
        if prob_cys1==None or prob_cys2==None:
            continue
        cmd.select("Cys1", 'resi ' + str(cysnum1) +
                   ' and chain '+cys1chain)
        cmd.select("Cys2", 'resi ' + str(cysnum2) +
            ' and chain '+cys2chain)

dst = cmd.distance('dst', 'Cys1 and name sg',
            'Cys2 and name sg')

dst_prob = getProbability(length_table, dst)
cmd.select('cys1cb', 'Cys1 and resn cys and name cb')
cmd.select('cys1sg', 'Cys1 and resn cys and name sg')
cmd.select('cys2sg', 'Cys2 and resn cys and name sg')
cmd.select('cys2cb', 'Cys2 and resn cys and name cb')

angle = cmd.get_dihedral('cys1cb', 'cys1sg',
            'cys2sg','cys2cb')

angle_prob = getProbability(angle_table, angle)
record = {'RotamerPair' : (i+1, j+1),
            'Length':dst,
            'Dihedral':angle,
            'RotamerProb': prob_cys1*prob_cys2,
            #'StructuralLikelihood':(dst_prob,
                           angle_prob)}
            'StructuralLikelihood': \
            np.log(angle_prob) + np.log(dst_prob)}



key = 'Pair' + str(i+1) + str(j+1)
candidates[key] = record
cmd.hide('dashes', 'dst')
```

```
                    cmd.hide('labels', 'dst')

            if len(candidates.keys())>1:
                all_candidates.append(candidates)

        return(all_candidates)




    def getDisulfideCandidates(pdb):
        '''
        DESCRIPTION

        Wrapper function to analyze and score different candidate
        disulfide bonds.
        '''
        amino_acids = {'CYS': 'C', 'ASP': 'D', 'SER': 'S',
                       'GLN': 'Q', 'LYS': 'K', 'ILE': 'I',
                       'PRO': 'P', 'THR': 'T', 'PHE': 'F',
                       'ASN': 'N', 'GLY': 'G', 'HIS': 'H',
                       'LEU': 'L', 'ARG': 'R', 'TRP': 'W',
                       'ALA': 'A', 'VAL':'V', 'GLU': 'E',
                       'TYR': 'Y', 'MET': 'M'}

        getPDBFile(pdb)
        findSurfaceResidues(cutoff=0.5)
        carbons_dist = getDistanceMatrix()
        potentials = getCandidateResidues(carbons_dist)
```

```
results = evaluateForFit(potentials)

mutations=[]

for candidate in results:

    (cys1chain, cys1, cys1num, cys2chain, cys2, cys2num) =\

            re.findall(r'(\S+): (\S+) \((\S+)\) -- (\S+):

                (\S+) \((\S+)\) ', candidate['Residues'])[0]

    mutation=amino_acids[cys1] + cys1chain + cys1num + \

                amino_acids['CYS'] + ';' + amino_acids[cys2]\

                        + cys2chain + cys2num

                        + amino_acids['CYS'] + ';\n'

    mutations.append(mutation)

    print '\n\n====='+candidate['Residues']+'====='


    print 'Evolutionary Trace score: ' + str(candidate['ETscore'])


    print 'Pair, Rotamer Probability, Structural Likelihood,\

                Bond Length, Dihedral'


    for k,v in candidate.iteritems():


        if k[0:4] == "Pair":

            print '%s, %.3f, %s, %.3f, %.3f' % \

               (str(v['RotamerPair']), v['RotamerProb'],

                   v['StructuralLikelihood'], v['Length'],

                   v['Dihedral'])


run_foldx(mutations,pdb)
```

```
def run_foldx(mutations,pdb):
    print mutations
    f=open("./foldx_runs/individual_list_"+pdb,"w")
    for mutation in mutations:
        f.write(mutation)
    f.close()
    f=open("./foldx_runs/config_"+pdb,"w")
    f.write("command=BuildModel\n")
    f.write("pdb="+pdb+".pdb\n")
    f.write("mutant-file=./foldx_runs/individual_list_"+pdb+"\n")
    f.write("numberOfRuns=3\n")
    f.write("output-dir=./foldx_runs\n")
    f.close()
    sp.call(["./foldx","-f","./foldx_runs/config_"+pdb])

import __main__
__main__.pymol_argv = [ 'pymol', '-xiqc'] # Quiet and no GUI

import sys, time, os
import pymol
os.environ['PYMOL_DATA']='/usr/share/pymol/data'
os.environ['CHEMPY_DATA']='/usr/share/pymol/data/chempy'
stdout = sys.stdout
stderr = sys.stderr
# In /usr/lib/python2.7/dist-packages/chempy/fragments/__init__.py
# change path = chempy.path + 'fragments/' to
# path = '/usr/share/pymol/data/chempy/fragments/'
pymol.finish_launching()
```

```
##
# Read User Input
spath = os.path.abspath(sys.argv[1])
sname = spath.split('/')[-1].split('.')[0]
# Load Structures

getDisulfideCandidates(sname)
# Get out!
pymol.cmd.quit()
```