BIOINFORMATICS ORIGINAL PAPER Vol. 24 no. 20 2008, pages 2339–2343
doi:10.1093/bioinformatics/btn450

Structural bioinformatics

Optimal design of thermally stable proteins

Ryan M. Bannen[1,†], Vanitha Suresh[2,†], George N. Phillips Jr[2], Stephen J. Wright[1] and Julie C. Mitchell[2,3,*]

[1]Department of Biochemistry, University of Wisconsin-Madison, 433 Babcock Drive, [2]Department of Computer Science, University of Wisconsin-Madison, 1210 W. Dayton and [3]Department of Mathematics, University of Wisconsin-Madison, 480 Lincoln Drive, Madison, WI 53706, USA

Received on May 16, 2008; revised on July 29, 2008; accepted on August 19, 2008

Advance Access publication August 22, 2008

Associate Editor: Burkhard Rost

ABSTRACT

Motivation: For many biotechnological purposes, it is desirable to redesign proteins to be more structurally and functionally stable at higher temperatures. For example, chemical reactions are intrinsically faster at higher temperatures, so using enzymes that are stable at higher temperatures would lead to more efficient industrial processes. We describe an innovative and computationally efficient method called Improved Configurational Entropy (ICE), which can be used to redesign a protein to be more thermally stable (i.e. stable at high temperatures). This can be accomplished by systematically modifying the amino acid sequence via local structural entropy (LSE) minimization. The minimization problem is modeled as a shortest path problem in an acyclic graph with nonnegative weights and is solved efficiently using Dijkstra's method.

Contact: mitchell@biochem.wisc.edu

1 INTRODUCTION

For many industrial and laboratory settings, it is advantageous to make a protein more structurally and functionally stable at higher temperatures. A more stable protein in an industrial setting allows for higher process temperatures which can increase reaction rates, increase reactant solubility and reduce the risk of microbial contamination (Eijsink *et al*., 2004; Turner *et al*., 2007). In a laboratory setting, increased stability can make the protein easier to store and handle.

Several approaches have been developed for making more thermally stable proteins. The first approach is rational or structure-based design that involves making particular amino acid mutations to specifically improve traits in the protein's structure. These mutations can be made to improve Van der Waals' interactions, hydrogen bonds, salt bridges, interactions with ions and disulfide bridges as well as several other terms (Eijsink *et al*., 2004). Structure-based design has been shown to successfully increase the thermal stability of proteins in a number of cases (Eijsink *et al*., 2004; Korkegian *et al*., 2005; Shah *et al*., 2007; Vieille and Zeikus, 2001). However, this method is limited by the fact that the 3D structure of the protein must be known before the mutations can be chosen. While the structures for many proteins are available in the Protein Data Bank (PDB) (Berman *et al*., 2000), the 3D coordinates for other proteins remain unknown. Another drawback is the lack of a clear methodology for deciding what properties to modify and which parts of the amino acid sequence to mutate. As such, current approaches are heavily directed by the user and cannot be automated.

A second approach is directed evolution, which attempts to speed up natural evolution in a laboratory setting. A number of random mutations are made to the initial protein sequence and these mutations are evaluated for improvements of a specific trait. The mutants that perform the best are then used for additional rounds of mutations until the researcher is satisfied with the results. As with structure-based design, this approach has been successful in improving the stability of proteins (Counago *et al*., 2006; Eijsink *et al*., 2005; Schoemaker *et al*., 2003; Vieille and Zeikus, 2001), but has its own limitations. Instead of requiring the 3D structure of the target protein, directed evolution requires a significant amount of laboratory resources for performing the multiple rounds of mutations and selections. Therefore, this approach can be both expensive and time-consuming.

A third approach is the consensus method, which is a computational approach to improving thermal stability. It uses data from sequence databases to find commonality between proteins within the same family (Chan *et al*., 2004; Lehmann and Wyss, 2001). A multiple sequence alignment is first performed using the target sequence and a large number of homologous sequences. If a majority of the homologous sequences all have the same amino acid in a particular position and if it is different from the amino acid in the target sequence, the consensus method states that the amino acid in the target sequence should be mutated to the amino acid shared by the majority of the homologous sequences. Although, it does not require knowledge of the 3D structure of the protein or the laboratory resources required for directed evolution, it does require a large number of homologous sequences to be known for the target protein, as well as arbitrary constraints for picking which amino acids to mutate when there is not a clear 'majority' amino acid at a given position in the sequence alignment.

We have recently devised a novel approach called Improved Configurational Entropy (ICE) (Bae *et al*., 2008) to design more thermally stable proteins based on measures of local structural entropy (LSE). LSE is a value that was first derived through a careful analysis of the PDB. (Chan *et al*., 2004). In that study, Chan *et al*.

*To whom correspondence should be addressed.

†The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.

examined structures deposited in the PDB to see how often particular amino acid tetramers appeared in protein secondary structures. If a tetramer appeared in several kinds of secondary structures, it was given a higher LSE value than a tetramer that was always in the same secondary structure. The study also showed correlations between LSE and previously published differences of thermal stability for thermophilic proteins and their mesophilic homologues. We incorporated LSE as part of ICE and used it to choose amino acids from closely homologous proteins that minimized the total structural entropy for the target sequence. In our initial study, we implemented ICE by using an exhaustive brute-force approach (Bae *et al*., 2008). Our target protein was the mesophilic adenylate kinase (AK) from *Bacillius subtilis* (AKmeso) and we were successful in making three variant proteins (AKlse1, AKlse2 and AKlse3) that were shown to have higher thermal stability when compared to the wild-type AK. One of the interesting aspects of this study was that even though the homologous sequence was from *Bacillus globisporus* (AKpsychro), a psychrophilic, less stable AK and not a protein from a more thermophilic species, we still saw a considerable increase in thermal stability by selectively applying our selection algorithm.

While the brute-force implementation of ICE was successful in finding the LSE local minimum for the specific case of AK, it was also computationally expensive and was limited to considering only two sequences. Since we have shown that the method works for a target protein and a less-stable homolog, we aim to expand the scope of the study by increasing the computational efficiency of the underlying LSE minimization problem so that it can be applied to any protein with any number of homologous sequences.

2 METHODS

In order to solve the LSE minimization problem efficiently, we model it as a shortest path network optimization problem. The key to constructing a network representation is to decompose a protein sequence of length n into an ordered sequence of nodes in a network. The label for each node corresponds to a subsequence of four consecutive amino acids from the protein sequence (a tetramer), with directional arcs connecting the nodes. Thus, in this representation each sequence of length n is represented as a path of length $n-3$. We also prefix each node label with its 'stage', which is the position of its first amino acid in the protein sequence. For example, the protein sequence MERLTG is represented as the following sequence of nodes and arcs:

$$\text{Source} \rightarrow (1,\text{M,E,R,L}) \rightarrow (2,\text{E,R,L,T}) \rightarrow (3,\text{R,L,T,G}) \rightarrow \text{Sink}$$

Note the overlap in the node labels—the last three amino acids in each node label match the first three amino acids in the label corresponding to the next node. We can define an entropy measure for the full sequence by summing the entropy values for each of the three arcs in the above representation. The values for each arc are the entropy values derived by Chan *et al*. that correspond to the tetramer at the destination node (Chan *et al*., 2004). Note that our method takes advantage of a property of the LSE scoring function, namely that the structural entropy of each tetramer can be calculated independently of other tetramers. This property is crucial to the application of a shortest path network approach.

Given a set of sequences that give different possibilities for the amino acids at some positions of the sequence, we can construct a network in such a way that each possible path through the network corresponds to a possible hybrid of the sequences in this set. Formally, node i in this network has the label $(n_i, a_{i1}, a_{i2}, a_{i3}, a_{i4})$, where n_i is the stage number corresponding to this tetramer and $(a_{i1}, a_{i2}, a_{i3}, a_{i4})$ is the tetramer itself. There is a directed arc from node i to node j if the following four conditions hold: $n_i + 1 = n_j$, $a_{i2} = a_{j1}$, $a_{i3} = a_{j2}$, and $a_{i4} = a_{j3}$. (Node i is said to be a *predecessor* of node j.) The arc connecting nodes i and j is assigned a weight equal to the entropy value for the tetramer $(a_{j1}, a_{j2}, a_{j3}, a_{j4})$ corresponding to node j. There are two additional nodes in the network: a source node from which arcs emanate to all the first-stage nodes, with weights equal to the entropies for the tetramers at those nodes; and a sink node that is the destination for arcs emanating from all nodes at stage $n-3$, with all weights zero.

Now consider the two protein sequence example in Figure 1. The figure illustrates a network corresponding to a small protein sequence fragment

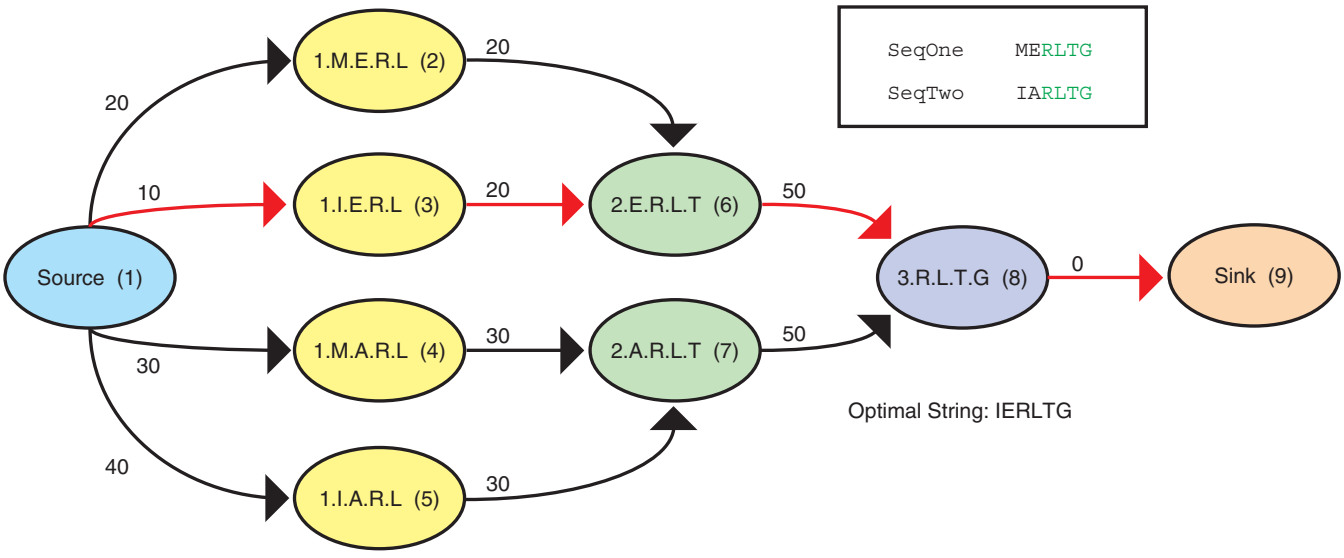

Fig. 1. An example of the graph that is assembled by ICE for a sequence alignment of two short protein sequences shown in the upper right corner of the figure. Nonconserved residues in the alignment are colored black. Each node in the graph represents a possible tetramer of amino acids that could be incorporated in the optimized solution. The numbers by each of the edges correspond to the LSE costs for choosing a particular tetramer. For this case, these LSE costs have been altered to clearly show the costs of choosing a particular path. The shortest path through the graph can be seen in red.

Table 1. The *D* (distance)-values for every iteration of Dijkstra's algorithm when applied to the example given in Figure 1

Iteration	S	w	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]	D[8]	D[9]
Initial	{1}	–	20	10	30	40	Infinity	Infinity	Infinity	Infinity
1	{1,3}	3	20	10	30	40	30	Infinity	Infinity	Infinity
2	{1,3,2}	2	20	10	30	40	30	Infinity	Infinity	Infinity
3	{1,3,2,4}	4	20	10	30	40	30	60	Infinity	Infinity
4	{1,3,2,4,6}	6	20	10	30	40	30	60	80	Infinity
5	{1,3,2,4,6,5}	5	20	10	30	40	30	60	80	Infinity
6	{1,3,2,4,6,5,7}	7	20	10	30	40	30	60	80	Infinity
7	{1,3,2,4,6,5,7,8}	8	20	10	30	40	30	60	80	80
8	{1,3,2,4,6,5,7,8,9}	9	20	10	30	40	30	60	80	80

with two variable positions. The two homologous six amino acid sequences give rise to a network with three stages, in addition to the source and sink nodes. The LSE value of the tetramer in each position is indicated on the incoming arcs. Note that the numeric node labels within parentheses in Figure 1 are aliases for the original node labels. For example, 2 is the numeric node alias for the original node label '1.M.E.R.L'. These labels will be used interchangeably in the ensuing sections for the sake of simplicity.

Any connected path through the network from source to sink corresponds to a protein sequence of the same length as the original sequence. The amino acid at each position is chosen from amino acids observed at that position in the multiple sequence alignment. The entropy corresponding to this path is obtained by summing the arc weights along the path. In this way, the problem of finding the optimal sequence can be reduced to finding the shortest path through this network from source to sink. Fortunately, a standard and highly efficient algorithm from network optimization—Dijkstra's algorithm—can be used to solve problems of this type. Dijkstra's algorithm is a 'greedy' algorithm that determines the cost of the shortest path from the source node to *every* node in the graph, including the sink node.

As with any dynamic programming approach, Dijkstra's algorithm starts by solving a simple subproblem, then expands this subproblem incrementally, modifying the solution at each step, until the solution of original problem is obtained. In the case of Dijkstra's algorithm, each subproblem consists of a subset of nodes whose shortest path to the source node is known. (The trivial subset for the first subproblem is the source node itself.) At each step, this subset is expanded by a single node, by adding the closest node to this source from outside the set of nodes with known distances. The algorithm terminates when the subset encompasses the sink node. The solution is recovered by backtracking along the arcs from sink node to the source node.

Since the algorithm is simple to describe, we now outline details of Dijkstra's approach and demonstrate its application to the given example. Given a directed graph $G=(V, E)$ where V is the set of nodes, E is the set of arcs, and $C[i,j]$ is the cost of the edge going from node i to node j, the algorithm maintains a set of node S for which the shortest distance from the source is known.

(1) We initialize S to contain only the source node.

(2) All nodes in the set $(V-S)$ that have a predecessor in S are associated with a 'special path' starting at the source node that passes only through nodes in S, whose total length is as short as possible. The special path for every node is stored in an array D, while the predecessor node (in S) on this special path is stored in an array P.

(3) At each step, we add to S a single node from the set $(V-S)$, whose distance from the source (D-value) is as short as possible.

(4) The algorithm terminates once all the nodes have been moved to S.

(5) The shortest path for any given node can be determined by tracing the predecessors in P.

Table 2. The *P* (predecessor)-numbers for the nodes given in Figure 1

Node	2	3	4	5	6	7	8	9
P[node]	1	1	1	1	3	4	6	8

For example, P[6] = 3 indicates that node 3 is a predecessor to node 6 on the shortest path from node 1 to node 6. The shortest path can be determined by tracing the predecessor array backwards from the sink (node 9) to the source (node 1). The shortest path for the example in Figure 1 is {9,8,6,3,1}.

Further details on Dijkstra's algorithm have previously been published (Aho *et al*., 1983).

Details of how the algorithm operates on the example are provided in Figure 1. Note that we use the parenthesized numeric node labels in the ensuing explanation below.

Initially, the set $S=\{1\}$ and the D-values for the neighboring nodes 2, 3, 4 and 5 are $D[2]=20$, $D[3]=10$, $D[4]=30$, and $D[5]=40$. Since nodes 6, 7, 8 and 9 cannot be directly reached from node 1, their distances are set to infinity.

At every iteration of the algorithm, w represents the node that is selected to enter the set S. For example, in the first iteration, we select the node $w=3$ to enter S, since it has the smallest D-value. Proceeding to the second iteration, the distances to the neighbors of node 3 in the set $(V-S)$ are updated. Since node 6 is the only such neighbor of node 3, we get $D[6]=\min(\text{infinity}, D[3]+C[3,6])=\min(\text{infinity}, 10+20)=30$. The other distances do not change because there is no way to reach them as part of a path containing node 3. The $P-$number for node 6 i.e. $P[6]=3$ indicates that node 3 is the predecessor to node 6 on the shortest path from node 1 to node 6. The sequence of D-numbers at the end of each iteration is indicated in Table 1 and the final P-numbers are given the predecessor array table (Table 2). By tracing the predecessor array backwards from the sink (node 9), one can clearly see that the nodes on the shortest path tracing backwards from sink (node 9) to source (node 1) are {9,8,6,3,1}. The shortest path is shown in Figure 1.

The optimal string is retrieved from the shortest path by stripping out the node stage numbers and merging the tetramers along the path, eliminating overlaps. In the given example, we thus obtain the string IERLTG.

Our implementation of Dijkstra's algorithm in C++ uses the adjacency list representation to store the vertices in $V-S$. The overall running time for an efficient implementation of Dijkstra's algorithm on a general graph is $O(e \log v)$ where v is the number of nodes and e is the number of arcs. For our graph, we calculate the value of v in terms of the number of candidate amino acids p_i at each stage i. The number of nodes at stage i equals the total number of possible tetramers starting at position i, which is the product $(p_i, p_{i+1}, p_{i+2}, p_{i+3})$. By defining $p_{n+1}=p_{n+2}=p_{n+3}=1$, we obtain the total number of nodes v by summing $(p_i, p_{i+1}, p_{i+2}, p_{i+3})$ over $i=1,2,\ldots,n$. Note that this quantity is bounded by a constant multiple of sequence length n. We obtain a bound on the number of arcs e by noting that the number of

outgoing arcs from any node at stage i is exactly p_{i+3}. Hence, the total number of arcs e is at most a factor $max_{i=1,2,\ldots,n}$, p_i greater than v, so that e is also bounded by a constant multiple of n. The general complexity analysis thus implies that Dijkstra's algorithm would have a running time of $O(n \log n)$ on our graph. In practice, we will usually have only an $O(n)$ running time. This is because, since our graph has a linear structure with edge weights that are not too diverse, the two main operations at each iteration—choosing the minimum D-value from among the nodes in $V - S$ to enter the set S, and updating the D-values of the neighbors of the latest node added to S—can usually be performed in time that is bounded independently of n.

3 RESULTS

3.1 Performance of ICE using the brute-force approach

The brute-force implementation of ICE described in the initial study (Bae, 2008) had two steps: the creation of all the possible sequences given the allowable substitutions, and the average LSE calculation for the generated sequences. The latter step was the rate-limiting step, as it required tetramers to be constantly looked up on a table that was 160 000 lines long. With this previous study, the most computationally expensive calculation was finding the lowest average LSE value for the variant AKlse3. This sequence had 19 possible mutations and using a Sun Fire X4100 Linux workstation (AMD Opteron Processor, 2.2 GHz, 4.0 Gigs RAM), this algorithm needed 3 h and 40 min to calculate the LSE scores for all the possible mutants. Due to the way this algorithm scales, it was unreasonable to perform it using more than 28 allowable substitutions at a time (2^{28} possible sequences).

Two of the other variants in our previous study had 53 allowable substitutions between the target protein and the homologous sequence. In order for this algorithm to work in a reasonable timeframe, we needed to split the multiple sequence alignment into four manageable segments, the largest of which had 17 mutations. The segments were split into regions where there were at least four conserved amino acids in a row. Because LSE is calculated by examining tetramers, each amino acid influences the LSE values for its three neighboring residues. By splitting the sequence at regions where there were four conserved residues, it ensured that the mutations made in one segment would not influence the mutations in the following segment. We were fortunate that our sequence alignment allowed for the four segments to be isolated and run separately. Had we been unable to split the problem into four segments, running this algorithm with 2^{53} possibilities would have taken an estimated computation time on the order of millions of years.

3.2 Performance of ICE using the shortest path approach

The shortest path approach for ICE described in this article addresses many of the previous approach's limitations and dramatically improves the computational cost. Because of the way the nodes are organized, there is no immediate upper limit on the number of allowable substitutions, and there is no practical limit for how many sequences can be incorporated into the algorithm. We used this algorithm to solve the problem with the 19 mutations described above using the same Sun Fire X4100 workstation. Instead of taking 3 h and 40 min, this algorithm arrived at the correct answer in less than a second.

Other tests with the shortest path approach further revealed its computational efficiency. With each test, the length of the sequences, the percent sequence identity, and the number of sequences included are all variables but they can all be compared by counting the total number of nodes in the graph for each of the cases. For example, using the shortest path approach with two sequences that are 67% identical and 4800 amino acids long will produce a graph roughly the same size as two sequences that are 400 amino acids long and share 0% sequence identity (∼6500 nodes). The shortest path algorithm calculated the global minimum in these two cases in less than a second. In a substantially larger case with ten sequences of 200 amino acids that share 0% sequence identity, the graph is on the order of 2 million nodes and the global minimum was found in 26 h using a single processor from the aforementioned Sun Fire X4100 Linux workstation. This shows that the algorithm is computationally efficient even in extreme situations.

3.3 Discussion and extensions

The application of a shortest path algorithm allows for a wider application of ICE to improve the thermal stability of proteins. Instead of being restricted to a brute-force algorithm that was limited to a small number of mutations, the shortest path algorithm described here can be used on proteins much larger than the 217 amino acids in AK.

Currently the input sequences for the ICE algorithm cannot have any 'gaps' when the sequences are aligned. If there is a gap, the sequences on either side of the gap need to be run separately and it is up to the user to split the sequences accordingly. In the future, this limitation can easily be circumvented by allowing for automated segmentation of the sequence alignment. Although, the algorithm's current implementation is extremely efficient for our current protein targets, the complexity could be further improved by using Fibonacci heaps as the algorithm's data structure (Fredman and Tarjan, 1987).

Finally, this same basic procedure can be used to optimize properties of protein structures based on other localized measures that can be defined according to a sliding window of blocked residues. In the present, we have used structural entropy applied to blocks of four residues, but other block sizes and optimality measures are amenable to solution using the procedure we have presented. For example, localized measures of aggregation propensity (Bracken *et al*., 2001; Esteras-Chopo *et al*., 2005; Galzitskaya *et al*., 2006; Lopez De La Paz and Serrano, 2004; Romero *et al*., 2004; Thompson *et al*., 2006; Zbilut *et al*., 2004) might be used to design proteins less likely to form amyloid fibrils.

Funding: DOE BACTER (DE-FG2-04ER25627 to V.S. and R.M.B); The Center for Eukaryotic Structural Genomics NIH/NIGMS (U54 GM074901-01 and P50 GM064598 to G.N.P.).

Conflict of Interest: none declared.

REFERENCES

Aho,A.V. *et al*. (1983) *Data Structures and Algorithms*. Addison Wesley, USA.

Bae,E. *et al*. (2008) Bioinformatic method for protein thermal stabilization by structural entropy optimization. *Proc. Natl Acad. Sci. USA*, **105**, 9594–9597.

Berman,H.M. *et al*. (2000) The Protein Data Bank. *Nucleic Acids Res.*, **28**, 235–242.

Bracken,C. *et al*. (2001) Disorder and flexibility in protein structure and function. *Pac. Symp. Biocomput.*, **6**, 64–66.

Chan,C.H. *et al*. (2004) Relationship between local structural entropy and protein thermostability. *Proteins*, **57**, 684–691.

Counago,R. *et al*. (2006) In vivo molecular evolution reveals biophysical origins of organismal fitness. *Mol. Cell*, **22**, 441–449.

Eijsink,V.G. *et al*. (2004) Rational engineering of enzyme stability. *J. Biotechnol.*, **113**, 105–120.

Eijsink,V.G. *et al*. (2005) Directed evolution of enzyme stability. *Biomol. Eng.*, **22**, 21–30.

Esteras-Chopo,A. *et al*. (2005) The amyloid stretch hypothesis: recruiting proteins toward the dark side. *Proc. Natl Acad. Sci. USA*, **102**, 16672–16677.

Fredman,M.L. and Tarjan,R.E. (1987) Fibonacci heaps and their uses in improved network optimization algorithms. *ACM*, **34**, 596–615.

Galzitskaya,O.V. *et al*. (2006) Prediction of amyloidogenic and disordered regions in protein chains. *PLoS Comput. Biol.*, **2**, e177.

Korkegian,A. *et al*. (2005) Computational thermostabilization of an enzyme. *Science*, **308**, 857–860.

Lehmann,M. and Wyss,M. (2001) Engineering proteins for thermostability: the use of sequence alignments versus rational design and directed evolution. *Curr. Opin. Biotechnol.*, **12**, 371–375.

Lopez De La Paz,M. and Serrano,L. (2004) Sequence determinants of amyloid fibril formation. *Proc. Natl Acad. Sci. USA*, **101**, 87–92.

Romero,P. *et al*. (2004) Natively disordered proteins: functions and predictions. *Appl. Bioinform.*, **3**, 105–113.

Schoemaker,H.E. *et al*. (2003) Dispelling the myths–biocatalysis in industrial synthesis. *Science*, **299**, 1694–1697.

Shah,P.S. *et al*. (2007) Full-sequence computational design and solution structure of a thermostable protein variant. *J. Mol. Biol.*, **372**, 1–6.

Thompson,M.J. *et al*. (2006) The 3D profile method for identifying fibril-forming segments of proteins. *Proc. Natl Acad. Sci. USA*, **103**, 4074–4078.

Turner,P. *et al*. (2007) Potential and utilization of thermophiles and thermostable enzymes in biorefining. *Microb. Cell Fact.*, **6**, 9.

Vieille,C. and Zeikus,G.J. (2001) Hyperthermophilic enzymes: sources, uses, and molecular mechanisms for thermostability. *Microbiol. Mol. Biol. Rev.*, **65**, 1–43.

Zbilut,J.P. *et al*. (2004) Singular hydrophobicity pattern and net charge: a mesoscopic principle for protein aggregation/folding. *Physica A*, **343**, 348–358.