ODU Technical Report No. 2002-02

Michael Wagner[1] · Jaroslaw Meller[2] · Ron Elber[3]

Large-Scale Linear Programming Techniques for the Design of Protein Folding Potentials

February 5, 2002

Abstract. We present large-scale optimization techniques to model the energy function that underlies the folding process of proteins. Linear Programming is used to identify parameters in the energy function model, the objective being that the model predict the structure of known proteins correctly. Such trained functions can then be used either for *ab-initio* prediction or for recognition of unknown structures. In order to obtain good energy models we need to be able to solve dense Linear Programming Problems with tens (possibly hundreds) of millions of constraints in a few hundred parameters, which we achieve by tailoring and parallelizing the interior-point code `PCx`.

Key words. protein folding, interior-point algorithm, PCx, Linear Programming, linear feasibility, parallel processing

1. Introduction

The recent unveiling of the human genome marked the transition in the biological sciences towards the post-genomic era, in which the understanding of protein structure and function becomes a crucial extension of the sequencing efforts. Despite recent progress in high throughput techniques, the experimental determination of protein structure remains a bottleneck in structural genomics. This poses a challenge and an opportunity for computational approaches to complement and facilitate experimental methods.

The protein folding problem consists of predicting the three-dimensional structure of a protein from its amino acid sequence. The methodology and modeling aspects of protein folding have been vastly discussed in the literature (for excellent and up-to-date brief surveys of methods as well as their limitations, see [3] and [9]). In order to characterize the existing computational approaches to this problem one may distinguish two underlying principles.

The so-called *ab-initio* protein folding simulations attempt to reproduce the actual physical folding process using the thermodynamical hypothesis, first introduced by Anfinsen [2]. The unique three-dimensional structure of a protein

Department of Mathematics and Statistics, Old Dominion University, Norfolk, VA 23529–0077. e-mail: `mwagner@odu.edu`

Pediatric Informatics, Children's Hospital Medical Center, University of Cincinnati, Cincinnati, OH 45229–3039. e-mail: `jmeller@chmcc.org`

Department of Computer Science, Cornell University, Ithaca, NY 14853–7501. e-mail: `ron@cs.cornell.edu`

is postulated to correspond to a global minimum of the free energy function. Thus, the search for the native conformation entails the solution of a global optimization problem.

The *protein recognition* approach, in turn, relies on the fact that a large number of protein folds are already determined. Given an appropriate scoring function, which can be thought of as a simplified folding potential, these methods find the "best" template from the library of known folds. In other words, the search for the native conformation is restricted to the set of known structures, as opposed to an expensive search in the space of all possible conformations. The scoring functions for protein recognition can be based on amino acid sequence similarity or they may incorporate measures of sequence to structure fitness. The latter approach, known as *threading*, allows to find distant homologs that share the same fold without detectable sequence similarity [5].

In both *ab-initio* folding and protein recognition we are faced with the problem of finding (designing) an appropriate expression for the free energy or scoring function, respectively. While optimization tools are certainly crucial for finding the native conformation, they also play an important role in the modeling stage. This paper introduces new, tailored optimization tools for the design and evaluation of folding potentials with superior prediction and recognition capabilities.

The energy functions we consider here depend linearly on parameters. As discussed in [14], the linear dependence of the potential functions on their parameters is not a major restriction. Any nonlinear function can be expanded or at least approximated as a linear combination of basis functions. The challenge is to find a set of basis functions of small cardinality that captures most of the intrinsic complexity of the true energy function and thus make for a reasonable model. The tools we present here allow us to *evaluate* the power of different modeling approaches (basis functions), so that over time we expect these to become increasingly more sophisticated and capture most of the intrinsic complexity of the true energy function.

The requirement of perfect recognition of known structures results in a linear feasibility problem (pioneered by Mairov and Crippen [11]), which we solve using Linear Programming techniques. We show that our large-scale tools, which allow for the solution of systems with hundreds of millions of constraints, result in significant improvements in the quality of potentials. We also demonstrate how solving these very large Linear Programming problems in conjunction with the recently proposed "Maximum Feasibility" heuristic [15] may be used to evaluate different functional forms. Our ultimate goal is an "optimal" energy model which balances complexity and accuracy, while avoiding the dangers of over- and underfitting.

The structure of this paper is as follows: In Sections 2 and 3 we present the parameter identification problem and a Linear Programming solution to it, respectively. Section 4 describes preliminary computational results and their biological interpretation for several commonly used models. We conclude with an assessment of the power and usefulness of our tools and by pointing to future research directions.

2. Potential Function Modeling for Protein Folding

2.1. Designing the Functional Form of the Potential Model

Proteins are linear polymers composed of a sequence of amino acid residues that are connected by peptide bonds (creating the protein "backbone"). There are 20 different amino acids that are characterized by chemically unique side chains (containing from one to up to about 20 atoms) that hang off the backbone chain. Protein molecules consist of several tens to several thousands of amino acids and thus between a few hundred to tens of thousands of atoms.

Protein structure is often represented in terms of simplified, reduced models that speed up computation. For example, the commonly used contact model represents each amino acid by just one point in $\mathbb{R}^3$, which defines the approximate location of an amino acid. The overall shape of the protein is characterized in terms of contacts between closely packed amino acid residues. Such contact models allow us to capture the packing of hydrophobic residues that are buried in the core of the protein and contribute to the stability of the structure.

In the present work we consider energy functions that employ reduced, contact models of protein structure. We will use the terms structure, fold, or conformation to mean the three-dimensional structure of the protein as defined by a set of coordinates of the geometric centers of the amino acids side chains. Also, the terms side chains (centers), amino acids, and residues are meant to be synonymous. Finally, following our earlier discussion, we will use the terms energy function, scoring function and potential function interchangeably.

We will denote our models of the potential function by E, and we will write it as a function of a sequence of amino acids s and a three-dimensional structure (a triplet of coordinates) x. The energy models considered here may be expressed in terms of functions $\varphi_i(s,x)$:

$$E_y(s,x) = \langle \Phi(s,x), y \rangle,$$

where y is a vector of parameters that are to be determined, $\Phi = (\varphi_1, \varphi_2, \ldots, \varphi_n)$, and $\langle \cdot, \cdot \rangle$ denotes an inner product. The set of functions $\{\varphi_i\}$ may be thought of as a set of *basis functions*. The "right" choice of basis functions is critical for the quality of the model, and the tools presented here allow one to explore different possibilities.

For example, in the *pairwise contact potential* two amino acids of type α and β ($\alpha, \beta \in \{1, 2, \ldots, 20\}$), respectively, are said to be in contact if the distance of their geometric centers is less than a certain threshold (here we use the distance of 6.4 Å[13]). The energy as a function of a given sequence of amino acids s and a given three-dimensional structure x can thus be expressed in the following way:

$$E_y(s,x) = \sum_{\alpha \le \beta} y_{\alpha,\beta} N_{\alpha,\beta}(s,x), \tag{1}$$

where $N_{\alpha,\beta}(s,x)$ represents the number of α-β contacts when sequence s is folded into structure x, and $y_{\alpha,\beta}$ are (unknown) weight parameters which represent the contribution such a contact makes toward the overall energy of the molecule.

In Section 4.2 we will refer to two other models, discussed in detail in [13]. We call them *threading onion models* (THOM) since they characterize structural environment ("profile") of an amino acid in terms of its contact shells. THOM1 models define the type of a residue using the first contact shell only and it is meant to capture the solvent exposure of amino acid residues. The nature of a given contact is disregarded and one simply counts the number of times a side chain of type α has a given number of neighbors (contacts):

$$E_y(s,x) = \sum_{\alpha} \sum_{m=1}^{k} y_{m,\alpha} N_{m,\alpha}(s,x). \tag{2}$$

$N_{m,\alpha}(s,x)$ represents the number of times a side chain of type α has m contacts, k is the maximum number of contacts and $y_{m,\alpha}$ are parameters to be determined. THOM2 models, which include the second contact shell (neighbors of neighbors), are meant to mimic pairwise interactions while preserving the efficiency of profile models (see [13] for details).

2.2. Optimization of the Parameters of the Potential Function

One traditional and widely used approach to finding values for the parameter vector y has been to derive them from statistical information about native folds that are already determined. For example, for the contact potential (1),

$$y_i = -C \ln(p_{\alpha,\beta}/(p_\alpha p_\beta)), \tag{3}$$

where C is a constant that defines the energy units, whereas p_α and p_β are the respective frequencies with which the amino acids appear in the chain, and $p_{\alpha,\beta}$ is the frequency of contacts of that type [16].

These statistical, knowledge-based potentials learn form the native structures ("good" examples) only. In order to increase their power to distinguish misfolded states (the "bad" examples) from native states, more sophisticated protocols incorporate data from decoy folds. To achieve this, we demand that the models mimic the postulate that the native state have the lowest energy. If we denote the native structure of a given sequence s by x_s^*, then the perfect potential function should satisfy:

$$E_y(s,x) > E_y(s,x_s^*) \qquad \forall s, \forall x \neq x_s^*,$$

or, using the expansion in terms of basis functions,

$$\Delta E_y = \langle \Phi(s,x) - \Phi(s,x_s^*), y \rangle > 0 \qquad \forall s, \forall x \neq x_s^*. \tag{4}$$

A slight but meaningful generalization arises when introducing the notion of a distance between structures in order to distinguish between "close to native" (but misfolded) versus radically different structures. By demanding that the energy gap for the latter be larger than for the former we achieve hierarchical ordering of misfolded states (known as “funnel” in the protein folding literature). In this case we have reason to demand that

$$\langle \Phi(s,x) - \Phi(s,x_s^*), y\rangle \geq b_{x,x^*} \qquad \forall s, \forall x \neq x_s^* \tag{5}$$

for appropriately chosen numbers $b_{x,x^*} > 0$ that in general should be proportional to the distance between the native and misfolded structure.

One approach to designing potentials that improves upon statistical potentials is z-score optimization [10]. Here the quality of a parameter vector y is measured using the distribution of energy gaps ΔE_y defined in (4). In particular, the goal is to maximize the dimensionless ratio of the first and second moments of the distribution (the “z-score”):

$$z(y) = \frac{\mu(\Delta E_y)}{\sigma(\Delta E_y)}. \tag{6}$$

μ and σ denote the mean and standard deviation of the energy gap distribution, respectively. The quantity z is, of course, nonlinear in its arguments. While z-score optimization may lead to remarkable improvements in the quality of the trained potentials, it is heuristic in nature and it does not rule out negative energy gaps.

Our goal is to attempt to adapt the models by choosing the parameters y such that (4) holds explicitly. In other words, we would like the models of the energy function to *perfectly recognize* native structures. To this end, we sample misfolded conformations to form a finite system of linear inequalities. The prediction and recognition capability of the resulting model will depend greatly on the number and type of misfolded structures that are included in the consideration. We employ a simple procedure to generate decoy structures called *gapless threading*, in which sequences are (imaginatively) folded into structures that are known not to be their native states [13].

In general the number of parameters in the models that are of relevance to us is on the order of a few hundreds. We aim to allow for the solution of problems with hundreds of millions of constraints, resulting from extensive sampling of misfolded structures. Given sufficient diversity of sampled types of proteins and a large set of inequalities (one per decoy), one may hope that an appropriate set of basis functions $\{\varphi_i\}$ would capture the essential features of the energy function so that the model E recognizes the structures in the database correctly. In Section 4.2 we will use training sets of decoy to find parameters y and then verify the prediction capability of the resulting model on a different test set of decoys and structures.

There are a number of techniques to solve linear systems of inequalities (see, e.g., [20] for alternatives in the protein folding context); we focus on Linear Programming here. Linear Programming is equivalent to solving linear inequality systems, and the modern algorithms we use allow for the efficient solution of problems with the dimensions we are interested in.

3. Linear Programming Solutions

The requirement that the parameters y define a model that satisfies the inequalities (4) for a set of decoy structures can be written as a system of strict linear inequalities

$$A^T y > 0, \tag{7}$$

where $A^T \in \mathbb{R}^{m\times n}$. Typically n is on the order of a few hundreds (one column per basis function φ_i) and m is on the order of tens of millions or more (one row per generated decoy fold). We note that if a solution to (7) exists, then it can be scaled to satisfy the system

$$A^T y \geq \rho \mathbb{1}, \tag{8}$$

which is a problem more amenable to computations. $\rho > 0$ is an arbitrary constant and $\mathbb{1}$ is the vector of ones, which is chosen merely for convenience. Our specific choices for ρ will be discussed in Section 4.2. In the more general case (5), which we will refer to from now on, we get a system

$$A^T y \geq b, \tag{9}$$

where $b > 0$ is the vector of desired energy gaps.

3.1. Modeling Techniques and Choice of Algorithm

There are a number of ways to cast (9) as Linear Programming problems. Since any feasible y can be scaled with a positive constant one might think of imposing a constraint on the norm of y in order to bound the feasible region (see, e.g., [18] for an example of this approach). However it is not a priori clear that the resulting system is feasible since we have just introduced an (arbitrarily scaled) right hand side. Hence for now we refrain from introducing this scaling of y explicitly and instead rely on the quality of the software used to produce a well-scaled parameter vector whenever possible.

Our first approach lies in adding a trivial objective function to get

$$(P) \qquad \begin{array}{l} \min \ 0^T y \\ \text{s.t. } A^T y \geq b. \end{array} \tag{10}$$

It is instructive to look at the corresponding dual problem:

$$(D) \qquad \begin{array}{rl} \max & b^T z \\ \text{s.t.} & Az = 0 \\ & z \geq 0. \end{array} \tag{11}$$

We see immediately that the dual problem is always feasible, and in fact that either the origin is the only feasible and hence optimal solution or the dual problem is unbounded (which implies an infeasible primal constraint system). Both of these cases are obviously of interest to us, we discuss their relevance to us in the following section.

There are two prevalent types of software for Linear Programming: those codes which are based on the simplex method and those based on the more recent interior-point methods. Although we don't want to rule out that a sophisticated implementation of the simplex method (with column generation techniques) might be successful in this case, we note that simplex-based methods are not easily parallelized and is likely to run into difficulties due to the degeneracy of the problems.

Instead we focus on using *interior-point* algorithms to solve (P). The interested reader is referred to [21] for an excellent in-depth introduction to these methods, we constrain ourselves to pointing out some of the features that are important in this context. Interior-point methods are Newton-like iterative methods that solve a sequence of perturbed KKT systems. Most importantly, they enjoy polynomial-time convergence properties and have been implemented in very efficient software that is competitive with implementations of the celebrated simplex method. Usually (i.e., for reasonably sized problems) the major computational effort required in each iteration lies in forming a matrix of the form AD^2A^T and then solving a linear system with this matrix using a modified Cholesky factorization. (A is the matrix of the linear equality constraints given to the solver and D^2 is an iteration dependent diagonal matrix.)

Interior methods have another feature which is beneficial in the context of our application (besides being amenable to parallel computation). Ideally we would like the energy gaps ΔE_y from (4) to be as large as possible. This would mean that the native structures have significantly less energy that any misfolded shapes, something that is generally conjectured to be the case for the true energy function also. This corresponds to having a solution that is, in some sense, "centered", i.e., where the distance to the boundary of the polyhedron is maximized. Interior-point algorithms are known to converge to the *analytic center* of the primal-dual optimal face, which, while not identical with the geometric center, nevertheless bodes well for the solution being away from the boundary of the polyhedron. Since the system $A^Ty \geq b$ is unbounded (and thus also the optimal face of (10)), the notion of an analytic center is not well-defined in this context. Nevertheless, and even though there is no theoretical guarantee that the algorithm will produce nicely scaled and centered solutions, our experience has never produced examples where this is not the case.

A more sophisticated LP-modeling approach which we mention here avoids the aforementioned unboundedness of the optimal face by minimizing the norm of the parameter sought. Additionally, it deals explicitly with infeasibility by introducing slack variables z and minimizing their norm:

$$(P') \qquad \begin{array}{ll} \min & \|y\|_1 + \gamma\|z\|_1 \\ \text{s.t.} & A^Ty + z \geq b \end{array} \tag{12}$$

Here γ is a tradeoff parameter that must be chosen in advance. The dual problem can be written in the following way:

$$(D') \qquad \begin{array}{ll} \max & b^Tx \\ \text{s.t.} & -\mathbb{1} \leq Ax \leq \mathbb{1} \\ & 0 \leq x \leq \gamma\mathbb{1} \end{array} \tag{13}$$

The advantage of this formulation is that both problems are guaranteed to be feasible and their respective optimal faces are guaranteed to be bounded, which implies that their analytic center is well-defined.

This formulation is reminiscent of Support Vector Machines (see for example [8]), except that the 1-norm is used for the minimization of $\|y\|$. Support Vector Machines are quadratic programming problems with vast applications in data mining and data classification. Our particular case can be interpreted as finding a separating hyperplane between the energy gaps and the origin, so that one of the two data classes effectively just consists of a single vector (the zero vector). We conjecture that an efficient implementation of a massive support vector machine (such as the on presented in [8] will be a viable alternative to our linear programming approach.

Turning now to our specific application: we note that if problem (P) or (P') were to be fed to any of the interior solvers we are aware of, then slack variables would be introduced to transform the (primal) constraints into equalities. As a consequence the resulting system AD^2A^T would have millions of rows and columns (for the problems of size we are interested in) and be completely dense, making any computation with it unrealistic. However, the respective dual problems are already in the standard form which solvers use internally and the system to form and solve in this case has row and column dimension of a few hundred (and would thus be comparatively trivial!). We conclude that if we can hold the constraint matrix in a distributed computing environment and allow for matrix-matrix and matrix-vector multiplications, we can use standard interior-point algorithms to solve these problems. We also note that the dimensionality of (D') is only marginally larger than that of (D), the computational effort required to solve either one will essentially be the same.

3.2. Dealing with Infeasibility and Insufficient Memory

In the previous section we alluded to the case where the systems of inequalities (7) (or (9)) admits no solution. If this is the case then this simply means that the model characterized by the set of basis functions $\{\varphi_i\}$ in question is not sufficiently sophisticated to correctly recognize all the proteins in the database (with the chosen desired energy gaps b). From conceptual point of view, this outcome is certainly a valuable information and an important conclusion when a given model is to be evaluated. For example, [19] and [17] show this way that the simple contact potential is in fact not generally good enough to recognize all structures that are already known.

However, the issue is more subtle than a simple decision whether linear inequality system is feasible or not. Not including enough native and misfolded structures in the training set can result in "underfitting" of the parameters for a given model, which is likely to result in poor performance on a larger test set and in real applications. On the other hand, with more extensive sampling the chances of introducing inconsistent constraints increase, which might lead one to resort to smaller training sets to avoid infeasibilities. Again, the resulting potential may again be significantly underfitted. A striking example of this type is discussed in Section 4.2.

In [15] we discuss a case in which adding membrane proteins to a database of soluble proteins, which are characterized by different folding principles, makes the problem infeasible. In order to find a potential which recognizes this augmented set of proteins correctly the number of parameters and basis functions needs to be increased by an order of magnitude compared to the potential for the problem without the membrane proteins.

This motivates the need to deal with infeasible (or near-feasible) problems in an efficient way in order to still obtain meaningful models, e.g., by attempting to correctly recognize a maximum number of proteins. One idea to approximately achieve this is to choose a maximal subset of satisfiable constraints. Unfortunately this is known to be an NP-complete problem [6], which means that a fast algorithm for its solution is unlikely to exist. In [15] we introduced a *Maximum Feasibility* (MaxF) heuristic that aims at finding a "maximally feasible" parameter y, i.e., a parameter that satisfies the largest number of constraints possible. We summarize it here as Algorithm 1.

Set $k = 0$, start with an initial approximate solution y_0.
loop
  Form A_k^T and b_k by finding all rows of A^T such that $A_k^T y_k \geq b_k$ holds.
  if no new rows are added **then**
    STOP.
  end if
  Compute a centered solution by running an interior-point algorithm.
  Let y_{k+1} be the solution obtained. Set $k = k + 1$
end loop

Algorithm 1: The `MaxF` heuristic.

We stress that this is only a heuristic and one whose performance will depend critically on the choice of a good starting point. Nevertheless, and as we show in Section 4.2, we have found it to be very useful in our application. Starting, e.g., from a statistical potential (3), which can always and easily be computed, the interior-point solutions to the subproblems in the heuristic each result in further improvement of the quality of the solution. Another plausible initial solution can be obtained by carefully selecting a subset of proteins for which we want to impose perfect recognition, and which is sufficiently diverse to capture the underlying, dominating physical characteristics of the folding process.

Note that in order to use the MaxF heuristic we need to be able to load all currently satisfied inequalities into memory. For approximate solutions of a good quality most of the constraints should be satisfied, which again motivates the need for parallel solvers that can handle very large problems.

If the number of generated inequalities that are of interest causes the problem to be too large to fit into the available memory, then we have little choice but to resort to an iterative scheme in order to try to find a feasible solution (or prove infeasibility). In particular this was often the case when we were constrained to a single-processor environment [17] [13]. We choose a subsystem that is small enough to fit into memory, try to find a feasible point and check whether the solution satisfies the rest of the constraints. If some of the inequalities are violated, they are used to replace some of the constraints of the original subproblem and the procedure is repeated. It may be necessary to intervene manually to get this process to converge in reasonable time.

If the number of degrees of freedom is small compared to the number of inequalities that can be solved in one shot, then this approach has proven to be fairly successful if the problem was feasible. It is not difficult to see that, regardless of the constraint selection procedure, this procedure is not guaranteed to terminate if the original system is infeasible to start with. Even though our applications do not seem to pose great difficulties in finding infeasible subsets of inequalities in case the whole system is infeasible, we really would like to avoid having to resort to these iterative heuristics, and being able to solve large problems in one shot becomes crucial.

Table 1 summarizes our discussion. There are essentially two ways of avoiding the undesirable case of needing to deal with an infeasible system which is too large to fit into memory. Firstly, by implementing a parallel code for a distributed memory environment, we are able to solve larger problems. Secondly, with increasing sophistication, the models tested are more likely to be able to recognize increasingly larger numbers of protein structures and are hence less likely to produce infeasible systems. We conclude that the challenge is addressed to both computational scientists *and* biochemists to increase the quality of the models and the scalability of the software.

	problem fits into memory	problem is too large
feasible problem	"easy"	· heuristic iterative scheme · works if subproblems are large enough
infeasible problem	· get proof of infeasibility · use MaxF heuristic	· heuristic might cycle · **want to avoid this case**

Table 1. Strategies of dealing with infeasible or large problems

3.3. `pPCx`*: A Tailored Parallel Dense Implementation*

The problem given by (10) with tens of millions of inequalities cannot be solved by conventional and readily available software. Given the dimensions and the fact that the constraint matrix A is in general almost completely dense we need to be able to resort to a distributed memory environment in order to have a chance of solving these problems without having to use the heuristic iterative schemes outlined previously. As mentioned before, the dual problem (11) is more amenable for solvers since it is already in the commonly used standard form. Hence we will always let the solvers work on (11), the variables of interest to us will be the dual variables. The formulation (13) has not been implemented yet, this will happen in a future version.

$$A = \begin{array}{|c|c|c|c|c|}\hline \quad p_1 \quad & \quad p_2 \quad & \quad p_3 \quad & \quad \bullet \quad \bullet \quad \bullet \quad & \quad p_k \quad \\ \hline \end{array}$$

Fig. 1. Distribution of the constraint matrix

Our approach was to tailor the interior-point software PCx [7] to fit our needs. PCx is a publicly available, state-of-the-art serial implementation of a primal-dual predictor-corrector interior-point algorithm which enjoys widespread popularity in the optimization community. We replaced the sparse serial data structures and basic linear algebra routines by parallel dense counterparts. In particular, the constraint matrix A as well as all long vectors (vectors of length m) like the variables z are held in distributed form only. The distribution is done in the obvious way, with each of the k processors holding m/k columns of the matrix A (see Figure 1). This way, we expect the formation of the matrix AD^2A^T to speed up linearly with the number of processors. We can easily avoid having to store both the matrix A and its transpose by forming AD^2A^T as a sum of outer products:

$$AD^2A^T = \sum_{i=1}^{m} d_i^2 a_i a_i^T , \tag{14}$$

where a_i denotes the ith column of A and d_i the ith diagonal entry of D. Note that the matrix AD^2A^T is small and that the effort to solve the associated system is comparatively negligible. Hence we expect the bottleneck in this case to lie in forming the matrix. As a consequence of the expected linear speedup for the formation of the matrix AD^2A^T we expect the overall code to scale well with increasing problem size and number of nodes.

The computations also require several matrix vector products in each iterations, both involving the matrix A and its transpose A^T. Since the short vectors of length n are kept in serial (i.e., each processor owns a copy of each short vector), forming Az requires nontrivial communication among the processors which does not to scale as well.

In order to leverage off of existing parallel linear algebra packages we chose to use the data structures provided by the package PLAPACK [1]. However, the overhead associated with the PLAPACK routines (e.g., matrix-vector multiplications) is so significant that we chose to re-implement all necessary BLAS routines in order to speed up the code (an earlier implementation using the PLAPACK routines turned out to be impractically slow). The solution of the linear system is done using a modified version of the parallel Cholesky solver provided by PLAPACK (see [7] for details on that modification).

At present we can store approximately 220,000 inequalities in 210 parameters per GB of RAM. Note that, for convenience, we store the matrix entries in double precision format and do not yet exploit the fact that these entries typically are small integers. This will change in a future version of `pPCx`.

The data is generated using a package called `loopp` developed by Meller and Elber [12] which performs the threading of the sequences into structures in order to generate the decoys. This process is currently done in serial for simplicity, parallelizing this should be straightforward and will be done in the near future. Each process then reads the local portion of the matrix A from files, the data is put in the appropriate data structures and the core optimization code is called. Preprocessing of the constraint matrix is turned off since it would require accessing and comparing entire columns and rows of the constraint matrix and thus require significant communication overhead. Our experience shows that the models investigated do not require preprocessing in the sense that the code does not fail because of linear dependencies.

We ran our code on the Microsoft Windows 2000 based Velocity Cluster at the Cornell Theory Center. This machine consists of 64 nodes with 4 Pentium III-based processors per node running at 500 MHz and with 4 GB of main memory and 50 GB of disk space per node. For optimal performance we ran the code on at most 2 processors per node. The largest problem we solved so far consisted of approximately 60 million inequalities with 180 parameters. Since the implementation is entirely written in C with MPI extensions it is entirely portable to other platforms. Specifically one could imagine running on a large network of (possibly heterogeneous) workstations as long as the communication between them is not too much of a bottleneck.

4. Results

4.1. Parallel Performance

Our main interest is to find a feasible solution to the parameter identification problem (9), which corresponds to finding a dual feasible solution for the problem that is given to the Linear Programming solver. We modified the termination criteria in `pPCx` slightly to reflect this somewhat special case. Our experience with problems of different sizes is that typically between 5 and 20 iterations are necessary to find an optimal solution, and up to 60 if the problem is infeasible. The number of iterations obviously depends on the particular choice of right-hand side in (10). In particular, the number of iterations will depend on the choice of the constant ρ in (8). For the experiments presented here we chose $\rho = .01$ since this seems to represent a reasonable balance between computation time and feasibility of the resulting parameter vector.

The solution times vary from a few minutes (for problems with only a few hundred thousand constraints) to about 2.5 hours for a feasible problem with ca. 60 million constraints, solved on 128 processors.

Table 2 shows the performance on a problem with 30,211,442 constraints and 200 parameters. The somewhat unorthodox choice of numbers of processors is solely due to memory requirements, the matrix does not fit onto just 32 processors. The problem is infeasible, which for the purposes of this evaluation is

	34 processors	64 processors
`InitTime`	176.30 (1.7 % of total)	97.62 (1.7 % of total)
`LoopTime`	10158.95 (98.1 % of total)	5664.12 (98.2 % of total)
`FormADATTime`	8040.23 (79.1 % of loop)	4424.49 (78.1 % of loop)
`PredictorTime`	677.52 (6.7 % of loop)	397.58 (7.0 % of loop)
`CorrectorTime`	693.08 (6.8 % of loop)	403.68 (7.1 % of loop)
`Factorization`	42.50 (0.4 % of loop)	43.43 (0.8 % of loop)
`TotalTime`	10351.72	5770.57

Table 2. Scalability on a problem with 30 million constraints

irrelevant. The Linear Programming solver took 57 iterations to terminate. Note that these solution times refer to the Linear Programming part only and do not include the data generation performed by the threading software `loopp`.

The `TotalTime` figure is the sum of `InitTime` (setup time plus time to find an initial point) and `LoopTime` (the main loop in the interior-point algorithm). `LoopTime`, on the other hand, is the sum of `FormADATTime` (the time it takes to form the Schur complement matrix), `PredictorTime` and `CorrectorTime` (time to compute the components of the search direction) and `Factorization`. As expected, the computation is largely dominated by forming the Schur complement matrix (14), and thus the speedup for a problem of this size is linear, as expected. The other parts of the computations don't speed up as well, due to more communication overhead associated with the matrix-vector products of the form $y = Az$. The factorization of the 200×200 matrix is done using the PLAPACK code and does not speed up, probably because the matrix involved is too small. At this stage we are not concerned about that since the computation time spent on the factorization is negligible.

For a more comprehensive demonstration of the code scalability we present results of an experiment done with an arbitrarily chosen subset of $2,219,755$ inequalities from the original constraint set of 30M. Figure 2 shows the overall speedup as well as the speedups of the various components of the algorithms. `pPCx` took 17 iterations to find a solution.

We see an overall speedup factor of roughly 1.8, i.e., doubling the number of processor results in a reduction of about 4/9 computation time. It is not surprising that this is somewhat less impressive than for the larger problem presented earlier, since the percentage of computation spent in forming AD^2A^T is smaller. The speedup factor for `ComputeADAT` is closer to 2.

4.2. Applications to the Design of Folding Potentials

We applied `pPCx`, in conjunction with `loopp` threading program [12], to evaluate and design several folding potentials. To keep the scope of this paper contained we present a few representative computational results that are meant to illustrate the power and value of the algorithms discussed in the previous sections. The first set of experiments we present here consist of applying the MaxF heuristic

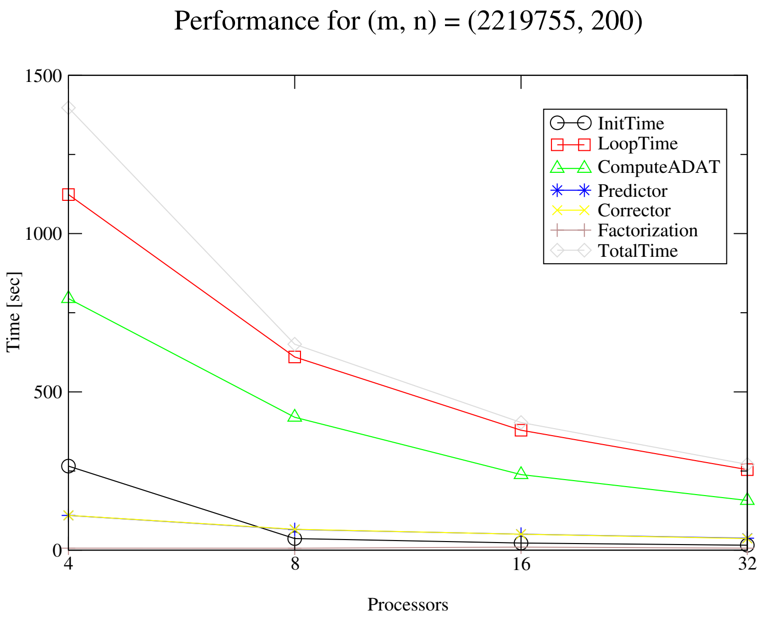

Fig. 2. Scalability on a problem with 2.2 million constraints

discussed in Section 3.2 to potentials introduced in Section 2 to see whether we can obtain improvements in their performance. We seek solutions that recognize possibly large sets of native structures when presented with a population of misfolded structures. We first discuss profile models (THOM1 and THOM2) and then a pairwise interaction model.

To test the recognition capabilities of a particular model of a potential energy function we train its parameters on a set of inequalities, derived form the training set of proteins using gapless threading. We first attempt to solve the whole training problem. If it proves to be infeasible, we perform a number of iterations of the MaxF procedure, starting from a certain initial guess, which is either a statistical potential derived using the `loopp` program or it is taken from the literature. The number of inequalities that are still not satisfied at convergence is used as a measure of the difficulty of a given training set and the quality of the model. The performance of the trained potentials is further evaluated on a control set of inequalities, derived from a disjoint set of proteins.

We used three different sets of proteins, developed before to train and test folding potentials. These sets were drawn from the Protein Data Bank PDB [4], which currently contains (as of January 2002) about 17,000 structures, with about 5000 distinct homology modeling targets that form a non-redundant subset of the PDB on the family level. The first set, which will be referred to as the TE set, was developed by Tobi et. al. [18] and it includes 594 structures chosen according to diversity of protein folds but also some homologous proteins (up to 60 percent sequence identity). Therefore it poses a significant challenge to the energy function. The total number of inequalities that were obtained from the TE set using gapless threading was 30,211,442. The second and third sets, referred to as S1082 and S2844, consist of 1082 and 2844 proteins, respectively. These were chosen to be relatively dense and non-redundant subsets of the databank. The S1082 set is used as control, whereas S2844 set is used to retrain pairwise model on a much larger set of proteins. The TE and S1082 sets are disjoint, although the S1082 set includes many structures that are relatively similar to representatives in the TE set [13]. All training and testing sets are available from the web [12].

As demonstrated before, the TE set is infeasible for the THOM1 model (2), and the parameters published in [13] were optimized on a smaller (feasible) set of proteins. Here, we take this potential as the initial solution y_0, and apply the MaxF procedure. The improvement is shown in Table 3. Column 2 lists the number of proteins that are not recognized correctly, column 3 the number of inequalities that are violated and, the last column contains the z-score as defined in (6). These are in fact three different measures of the quality of approximate solutions. Iteration 0 corresponds to the initial potential. The results on the training set are included in the left panel, whereas the results on the S1082 set, which is used as a control are included in the right panel.

	—TE training set—			—S1082 control set—		
MaxF iteration	not recog.	viol. ineqs.	z-score	not recog.	viol. ineqs.	z-score
0	120	162,274	1.58	415	539,664	1.42
1	59	1,217	1.87	360	249,854	1.66
2	54	905	1.93	358	250,850	1.66

Table 3. Results using MaxF procedure on a THOM1 potential

The results show a qualitative improvement on the training set. The initial solution violates more than 160 thousand constraints out of approximately 30 million in the TE problem. After just two iterations (with bulk of the improvement in the first iteration - note that we did not attempt to achieve full convergence) an approximate solution that violates only 905 constraints was obtained. The increase in z-score, from 1.58 to 1.93, indicates also the desired change in the overall shape of the distribution of energy gaps. The performance of the potential on the control set improved significantly as well, although to a smaller extent, still violating approximately 250 thousand constraints (out of 95 million included in the control set). As reported previously [13], various folding potentials from the literature reach only a limited accuracy on the S1082 set, which appears to be a demanding test, mostly because of many short proteins included in this set.

The TE set also proved to be infeasible for THOM2 model with two contact shells if fewer than 300 parameters were used [13]. Here, we consider a simplified THOM2 model with only 180 parameters. The reduction in number of parameters results from a different coarse graining of structural environments. Namely, we group together the second and the third, as well as the fourth and the fifth classes of primary sites that were used before, reducing the number of different types of primary sites from five to three (see [13] for detailed definition of contact types in THOM2 energy model). Using a statistical potential derived from the TE set as a starting point and applying MaxF we obtain a significantly improved reduced THOM2 potential with just 180 parameters, which approximately solves the TE problem that required as many as 300 parameters to solve it exactly.

As evidenced in Table 4, just one iteration of MaxF reduces the number of violated inequalities from approximately 265 thousand to 267. The increase in z-score from 1.22 to 1.87 is also impressive. A significant improvement in terms of the number of violated inequalities (from 1.6 million to 217 thousand) and overall shape of the distribution of energy gaps (z-score increasing from 1.03 to 1.53) is also observed on the control set. However, only minor improvement is observed in terms of the proteins that are not recognized exactly (from 364 to 335). The previously published THOM2 potential with 300 parameters when applied to the S1082 set misses only 205 proteins, although on the other hand, it violates more constraints (240 thousand) with a lower z-score of 1.35.

	—TE training set—			—S1082 control set—		
MaxF iteration	not recog.	viol. ineqs.	z-score	not recog.	viol. ineqs.	z-score
0	102	265,284	1.22	364	1,600,612	1.03
1	34	267	1.87	335	216,623	1.52
2	34	233	1.87	335	216,955	1.53

Table 4. Results using MaxF procedure on a THOM2 potential

Of course, a simultaneous increase in the number of recognized proteins and satisfied inequalities cannot be guaranteed, and in fact discrepancies between these two quality measures have been observed for a number of potentials from the literature on the S1082 set [13]. Since in practice additional filters, such as statistical significance estimates, are applied to a number of low energy matches, the solution with a smaller number of violated constraints may be advantageous.

The next potential we discuss here is a pairwise model (1). The TE set was used by Tobi et. al. [18] for parameter optimization, and the problem proved to be feasible. The solution had to be obtained iteratively by solving subproblems which fit into the memory of a single processor (see the scheme described in Section 3.2, and also Table 1). An additional objective function was used to skew the solution towards maximizing the z-score and thus to improve the quality of the energy gap distribution [18]. We attempted to improve this potential further. First, we solved the TE problem in one shot. The resulting solution does not show improvement over the Tobi et. al. potential (the z-score on the training set was 1.73, compared to 1.75). Secondly, and in order to further assess the effects of the training set and to sample more extensively from structural variations in protein families, we used the S2844 set for training. To keep the size of the training set manageable we only derived decoys for pairs of sequences and structures that are similar in length (the sequence must be not shorter than 80% of the structure it is aligned to, in order to generate a decoy and a constraint in effect). This results in an infeasible problem of approximately 16 million constraints. The well trained Tobi et. al. potential violates 64 thousand inequalities, missing as many as 600 proteins. When applying MaxF (with the Tobi et. al. potential as starting point) only a marginal improvement is observed: 71 additional proteins and approximately 2000 additional inequalities are satisfied after 5 iterations; the z-score of 1.83 remained unchanged.

Although MaxF's failure to improve does not constitute a definitive answer (and may simply have occurred due to the specific structure of the problem at hand), we conjecture that the observed results are an indication of the limits of the capacity of pairwise models. In light of the above, it is suggestive that the infeasibility reached before on various sets of native and misfolded structures with pairwise models [19] [17] was not due to some rare constraints, but rather due to the low resolution of the model. While non-redundant, the S2844 set includes many structural variations of certain folds with a distance of at least 3 Ångstrom RMS between the superimposed side chain centers [13]. This threshold of dissimilarity is apparently below the resolution of pairwise folding potentials. What we believe is interesting in this context is that the results from the application of optimization techniques can be interpreted biologically and will eventually lead to improved models and parameterizations of the free energy functional that underlies the process of protein folding.

5. Discussion and Conclusions

We described our efforts to provide practical large-scale Linear Programming based tools for the design and evaluation of potential functions that underlie the folding process of proteins. The interplay of biological and physical insights on one hand and optimization and modeling techniques, large-scale computing and heuristics on the other hand is used to facilitate the design of accurate and efficient potentials. The results presented here support the claim that biologically relevant results may be obtained using the new techniques. A systematic application of these techniques is expected to yield a significant improvement in the quality of potentials. We also expect to gain new insights to guide the selection of decoys to be included in the training process. The choice of a training set is a critical component of any successful learning procedure that extrapolates from examples.

With the present incarnation of `pPCx` we are able to solve problems with a few hundred parameters and tens of millions of constraints in a one-shot approach in a matter of minutes. Development of `pPCx` is ongoing. An extension of the current code will include an implementation of the alternative formulation of the potential modeling problem, defined in Section 3.1. The introduction of slack variables is expected to provide a more satisfactory solution for the infeasible case, while avoiding an increase in the size of the dual problem that we solve. It remains to be seen whether column generation or sampling techniques can be reliably used to speed up computation time. The software itself is application independent, we submit that any problem with similar dimensions (primal dimension several orders of magnitude smaller than dual dimension) and a dense constraint matrix can be efficiently solved using our code. In the future we plan to incorporate the parallel machinery developed for `pPCx` into a Support Vector Machine implementation that would handle large classification problems arising in genomics.

Acknowledgements. The authors gratefully acknowledge support from the Cornell Theory Center in the form of computing time on the Velocity Cluster. JM acknowledges support from the Children's Hospital Research Foundation, Cincinnati. RE acknowledges the support of NSF grant on 'Multiscale Hierarchical Analysis of Protein Structure and Dynamics'.

References

1. P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, and Y.-J. J. Wu. PLAPACK: parallel linear algebra package. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997)*, page 8 pp. (electronic), Philadelphia, PA, 1997. SIAM.
2. C. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.
3. J. R. Banavar and A. Maritan. Computational approach to the protein-folding problem. *Proteins: Structure, Function, and Genetics*, 42:433–435, 2001.
4. H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H.Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000. See also `http://www.rcsb.org/pdb/`.
5. J. U. Bowie, R. Luthy, and D. Eisenberg. A method to identify protein sequences that fold into a known three-dimensional structure. *Science*, 253:164–170, 1991.
6. N. Chakravarti. Some results concerning post-infeasibility analysis. *Eur. J. Oper. Res.*, 73:139–143, 1994.
7. J. Czyzyk, S. Mehrotra, M. Wagner, and S. J. Wright. PCx: an interior-point code for linear programming. *Optim. Methods Softw.*, 11/12(1-4):397–430, 1999.
8. M. C. Ferris and T. S. Munson. Interior point methods for massive support vector machines. Technical Report 00-05, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.
9. R. A. Friesner and J. R. Gunn. Computational studies of protein folding. *Annu. Rev. Biomol. Struct.*, 25:315–42, 1996.
10. R. A. Goldstein, Z. A. Luthey-Schulten, and P. G. Wolynes. The statistical mechanical basis of sequence alignment algorithms for protein structure prediction. In R. Elber, editor, *Recent Developments in Theoretical Studies of Proteins*, chapter 6. World Scientific, Singapore, 1996.
11. V. Mairov and G. Crippen. Contact potential that recognizes the correct folding of globular proteins. *Journal of Molecular Biology*, 227:876–888, 1992.
12. J. Meller and R. Elber. *LOOPP: Learning, Observing and Ouputting Protein Patterns.* Department of Computer Science, Cornell University, available at `http://www.tc.cornell.edu/CBIO/loopp`.
13. J. Meller and R. Elber. Linear programming optimization and a double statistical filter for protein threading protocols. *To appear in Proteins*, 2001.
14. J. Meller and R. Elber. Protein recognition by sequence-to-structure fitness: Bridging efficiency and capacity of threading models. In R. Friesner, editor, *Computational Methods for Protein Folding: A Special Volume of Advances in Chemical Physics*, volume 120. John Wiley & Sons, 2002 (in press).
15. J. Meller, M. Wagner, and R. Elber. Maximum feasibility guideline in the design and analysis of protein folding potentials. To appear in J. Comp. Chem.
16. M. J. Sippl and S. Weitckus. Detection of native-like models for amino acid sequences of unknown three-dimensional structure in a database of known protein conformations. *Proteins*, 13:258–271, 1992.

17. D. Tobi and R. Elber. Distance-dependent pair potential for protein folding: Results from linear optimization. *Proteins: Structure, Function and Genetics*, 41:40–46, 2000.
18. D. Tobi, G. Shafran, N. Linial, and R. Elber. On the design and analysis of protein folding potentials. *Proteins: Structure, Function and Genetics*, 40:71–85, 2000.
19. M. Vendruscolo and E. Domany. Pairwise contact potentials are unsuitable for protein folding. *Journal of Chemical Physics*, 109:11101–11108, 1998.
20. M. Vendruscolo, L. A. Mirny, E. I. Shakhnovich, and E. Domany. Comparison of two optimization methods to derive energy parameters for protein folding: perceptron and z-score. *Proteins: Structure, Function, and Genetics*, 41:192–201, 2000.
21. S. J. Wright. *Primal-dual interior-point methods.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.