Aus dem Institut für Molekulare Medizin der Albert-Ludwigs-Universität Freiburg i. Br. in Kollaboration mit dem Lehrstuhl für Bioinformatik der Universität Würzburg

A Combined Threading and Genetic Algorithm Based Approach to Predict Protein Structure

INAUGURAL - DISSERTATION

zur

Erlangung des Medizinischen Doktorgrades

der Medizinischen Fakultät

der Albert-Ludwigs-Universität

Freiburg i. Br.

Vorgelegt 2004

von Jürgen Christian Hans Hench

geboren in Basel

Jahr der Promotion: 2005

Table of Contents

Preface

This thesis deals with computer aided protein structure prediction. Since up to date molecular rendering has become an integral part of this research field we would like to invite the readers of this work to read this document in its digital version via a computer screen. Printing three dimensional protein models is not a good compromise and a lot of valuable information gets lost during this process.

We have not only included several three dimensional molecular models within this electronic document; we also provide quick links to almost all references which have been indexed by PubMed. This is why in the hardcopy several citations will appear underlined; these are all direct HTML hyperlinks to PubMed.

The molecular rendering is made possible through the Chime Plugin which is available for free from MDL (http://www.mdl.com) for several operating systems including MS Windows® and Apple Mac OS®. The addition of this plugin will turn your web browser into a 3D molecular viewer which may be useful to retrieve data sets from the Brookhaven Protein database.

Of course, the hardcopy of this work includes all models in a printed form as well. We have put certain effort on presenting the data nevertheless as clear as it is possible with a printed form.

I. Introduction

Many approaches have been made to predict the three dimensional conformation of the probably most diverse biomolecules, the proteins (Schultz and Schirmer, 1979; Creighton TE, 1993). Having a backbone mainly composed of twenty different building blocks one can easily imagine that their nature will be very complex. Experimental approaches to predict the structure of a protein are highly reliable but can not be done quickly. It takes a lot of time and work to achieve one single complete structure. This means that only proteins of major interest can be explored with experimental methods.
Proteins are a crucial component in almost any metabolic and signaling pathway of a cell. This makes proteins the major target structures for drug design and medical diagnostic techniques. It is important to gain knowledge on protein structures. Especially for experimental research groups any information on the structures in question may be helpful. Life sciences are still increasing in their speed of data gain. In particular, sequence information on proteins is very often available whereas structural information is rare. Tools are needed that support researchers with first sequence based predictions on structure in course or their experiments and for educating new scientists.
Nowadays the continuously increasing capacity of digital information processing devices makes it possible to support the experimental biochemistry by simulating natural circumstances. Even though this approach will never reveal more precise data than a real experiment, it is much cheaper, can be done quicker and may give useful hints to design better experiments.
The ideal and "holy goal of protein fold prediction software development is to write an algorithm that will convert a DNA or amino acid sequence directly into an atom coordinate file as it can be found in the Protein Database Brookhaven format. This format is widely used for storage of experimentally solved protein structures. Our project tries to work further towards this aim. It is an approach that wants to be helpful for other scientists who are interested in additional information on the proteins they actually deal with. Therefore one should consider this application as a simple and fast tool to achieve first predictions on protein structures for a given protein sequence. The following pictorial example shall point out this fact: Imagine a simple slide rule and a pocket calculator. With the slide rule you may calculate quite accurately for general purposes but you would rather use the calculator to be

more accurate. There are limitations in using the slide rules for numbers with lots of decimals. In contrast you may easily type them into the electronic calculator. But still everyone will be glad to have the slide rule instead of having nothing. With protein structure prediction algorithms one does not have the choice between the slide rule and the exact calculator. One can only choose between different heuristic prediction methods or one can use all of them in parallel. The results they give will differ in some respects and may be equivalent in others. The user will have to decide in which part of the result he wants to trust. Since the different protein structure algorithms are based on different principles, the matching parts of the resulting structures are likely to be correct. This is the context in which the work presented here should be seen and, of course, should be used.

There are quite a lot of protein structure prediction systems today providing results which are more accurate than the results of our application. In contrast to these systems our application is able to run on a standard desktop computer which is available in most laboratories. In order to address this problem we always have to make a compromise between accuracy and calculation speed. Finally, our application needs to process several data and settings automatically or the user would need to be experienced in protein structure modeling by simulation.

Nature has always been extraordinarily helpful to learn and copy things from. The Genetic Algorithm (GA), summarized by Goldberg in 1989 (see references), is an example on how man can tell a computer to figure out solutions for specific problems as nature did it when it developed different species as biological solutions for the challenges of the environment. The GA is a tool to find several solutions that fit a specific problem rather than to find one single solution for a specific problem. Spoken in a more biological manner this means that a specific location in environment is the problem and the species that are able to live in it are the solutions. The GA basically puts individual artificial solution trials to the test on how they survive in an artificial environment which represents the problem. The mechanisms are adapted from simple genetics. This explains why there will never be only one single result coming from a GA based application program. However, regarding protein structure predictions, all results that are produced will generally converge to the optimal structure as only the natively observed structure should best fulfill all selection criteria.

Previous application of the genetic algorithm has proven it to be a robust search strategy. It has been successfully adapted to protein fold prediction many times. However, predictions with a GA are time consuming, especially if they involve heavy and recurrent calculations. It is necessary to carefully optimize parameter weights by many simulations. In this thesis we

circumvent the empirical weight determination by combining two well established strategies: We have coupled the GA to threading. Threading identifies to which known protein structures the sequence to be predicted is most similar. It "threads" the sequence on the coordinates of all known crystal or NMR structures to test this. This method allows to quickly establish suitable, protein specific weights for the fitness parameters. The large speed gain of this procedure makes protein fold prediction on desktop computers possible.
We demonstrate the application in fold prediction for small proteins several of which are of medical importance, e. g. small toxins and interleukins. We have tried to extend our strategy for the prediction of helical transmembrane folds. Proteins containing this structural motif are of high medical and pharmacological interest such as the GTPase coupled 7-TM receptors. We show that the expanded concept of the GA is applicable to these structures with certain modifications. However, complete and detailed simulation of these macromolecules is still a challenge.
These facts make the GA favorable as a core algorithm for protein structure prediction. This work is based on previous applications that used the GA (Dandekar, Argos, Du; Saxena et al. 2001). In a GA based application there is no need to figure out a rigid solution path for the problem. This is exactly what one needs if one has no idea of how the final protein structure would look like but one still wants a computer to find a solution.

Motivation

We can summarize that protein fold prediction from sequence is a complex task. Currently there are no desktop versions for protein folding programs available. The GA is a powerful strategy and the combination with a threading procedure has made it even faster and more versatile.
As demonstrated in this thesis we achieve in this way a fast and efficient tool to derive first predictions on the structure of proteins including medical target structures.

II. Principles

II. 1. Representing a Protein

We address the question how we can represent a three dimensional protein structure in a linear or sequential manner so that it may easily be processed by a desktop computer.
There are two major ways to encode the conformation of a protein. The absolutely accurate method encodes every atom of the molecule in form of Cartesian coordinates (x, y, z). A variant of this kind of representation is an alpha-backbone which only contains the alpha carbon coordinates for each amino acid. Instead of describing only the alpha carbon atoms one could also represent each amino acid as a backbone (C-alpha, C' and N position) without its side chain. It is obvious that this kind of representation exceeds the computational capacity of a desktop machine.
An alternative to a full Cartesian coordinate system is a grid-free representation system. This system has succeeded in previous trials already (Dandekar & Argos, 1994, 1996). The description 'grid-free' should underline the general principle that is applied here. Ramachandran et al. (Ramachandran and Chandrasekaran, 1971) found standard conformations when they analyzed lots of crystal structures. The standard conformations e. g. for the alpha-helix can be described with two dihedral angles that encode the rotations around the alpha-carbon besides the other fixed angles in any amino acid. A protein can be modeled in a vector-like manner using several standard conformations as building blocks for the chain. A suitable set of seven standard conformations was introduced by Rooman, Kocher and Wodak (1991, Table II.1.1.). Our augmented set including the experimental data from Karplus (1996) consists of eight pairs of dihedral angles and provides a ninth conformation for cis-proline (Du and Dandekar, 1999, 2000). Except for the case of proline we provide eight possible conformation for any residue. For each proline that occurs the trans configuration may be changed to a cis and vice versa using the ninth cis-proline conformation.

Table II.1.1: *standard conformations*

conformation code	**conformation**	**phi (°)**	**psi (°)**	**omega (°)**
A	alpha-helix	-65	-40	180
C	3/10-helix [...]	-89	-1	180
B	beta-sheet	-117	142	180
P	extended #1	-69	140	180
E	extended #2	103	-176	180
G	extended #3	78	20	180
Y *)	extended #4	-109.4	165.6	180
N *)	extended #5	79.6	-67.1	180
O	*cis-proline*	-82	133	0

*) augmented set

If we compare both the coordinate based and the standard conformation modeling system, the latter clearly shows advantages for the use with desktop computers. Each residue is encoded by only one character or integer number. This conformation code addresses a pair of statically stored dihedral angles. The geometric information is finally used to calculate the positions for several atoms in any amino acid main backbone. These atoms include the nitrogen, alpha carbon, carboxyl carbon and, with the exception of glycine, the beta carbon. If we now compare the amount of data that must be processed per amino acid with both methods we come to one integer for the standard conformation system versus four or five float point values for the coordinate based method. These facts clearly make the standard conformation building blocks a favorable method for our current desktop application. We are now able to encode a whole protein structure with only a sequence of integers or characters. These linear strings may individually be converted to three dimensional arrays that contain the absolute Cartesian coordinates for every single atom represented.
Another problem we encounter is the modeling of linking loops for helical transmembrane proteins such as the well known adrenergic receptors (Stryer, Biochemistry). Building models of such complex molecules exceeds the capacity of our simplified grid-free building block representation. Additional conformations are required to model the short extramembranous loops which may be as short as two residues. We ask if it is possible to build a database for such loops without the use of structural information from models of transmembrane molecules. There are only very few of these molecules known in terms of crystal structure. The most popular example is the photoreceptor bacteriorhodopsin. The structure of this interesting molecule was determined experimentally by Nobel laureates Deisenhofer J, Huber

R and Michel H (Nobel lecture 1988). Nevertheless there is not enough information on these loops to build homology models for the linking loops for our purpose. The major difficulty would be to combine these nature derived structural elements with our standardized helical building blocks.

We introduce in this thesis a linear search algorithm that creates a loop database whose angles are derived from the allowed regions of the Ramachandran map. It is obvious that some of the known angles in linking loops are located even in forbidden or unstable regions of the Ramachandran map. However, it is not allowed to use any conformation of the complete and full map since we do not know which parts of the map are inaccessible even for transmembrane models.

Our search algorithm tries to link two paralleled alpha helices of about 45A length which is about the diameter of a cellular membrane. It uses steps of one degree (°) and scans the allowed regions of the RAMACHANDRAN map (Karplus PA, 1996). Conformations for one and two residues are evaluated. In case of two residues we combine one of our eight standard conformations with one pair of dihedral angles from the linear search. Every pair of angles that allows a more or less parallel alignment of the two test helices is stored within the loop database. The creation of the database may take several hours of computer processing on a desktop machine. Since the result is stored in a file automatically by our application, this task needs to be done only once. As an alternative it is possible to distribute such a database file with the software package.

When sequential data on secondary structure information are entered the user may specify the position of linking loops. This information is added to the secondary structure file. The program will now use the extended set of conformations from the database for these loop regions and is now able to represent these structural features.

II. 2. The Genetic Algorithm

During the thesis project we fist implemented the successful search strategy established previously from our group for protein folding, namely, the genetic algorithm. The performance of the new implementation and its new subroutines for rapid calculation was tested and the results are given here.
We can show that the GA is confirmed to be a robust and efficient search strategy. The second challenge, the identification of correct fitness parameters and their weights, will be explained later exploiting in addition the second strategy introduced in this thesis, the threading based approach. In order to achieve a fast conformational search for our structural prediction system we look for an algorithm that minimizes calculation time. Linear search algorithms would systematically cover an entire conformational space in terms of long time. However this principle is unsuitable for our purpose. We would like to bring protein structure modeling to the desktop computer with only a limited amount of calculations within a preferable time. In addition to a standard GA we apply a breeding procedure (see below) in our version.
We test here the efficiency of our new and fast implementation of the Genetic Algorithm (GA) which has been applied to many tough and complex multiple optimization problems earlier. Such problems involve the earliest trials of optimization of gas flow in jet engines (Ingo Rechenberg, 1969 and subsequently Prof. Paul Schwefel, University of Dortmund) and the flow through different gas pipelines (John Holland). John Holland, an investigator in genetic algorithms (1975) called them an "optimization learned from nature". Subsequently David Goldberg (1989) showed in his meticulous monograph the efficiency and good optimization of the GA in a number of problems in mathematics, engineering, biology and medicine, including hospital costs. Furthermore, Holland and Goldberg gave theoretical foundations for GA, in particular the identification of GA-hard problems and the schema theorem (details in Goldberg, 1989), explaining the optimization process of different solutions by systematic recombination.
A GA simulates an evolution process and uses it to find solutions that fit a given problem. In the case of grid-free protein modeling the problem is to find a correct three dimensional structure of a protein and the solution consists of a linear vector array representing all conformational elements. The easiest way to understand the GA is to compare it with nature as John Holland did it before. We summarize this comparison in table II.2.1..

Table II.2.1: *Comparison of nature and the GA*

Nature	**GA**
species / individuals	**chromosomes / individuals** = solution trials = secondary structure strings
environment, i. e. temperature, humidity, food etc.	**objective function**, judges the fitness of the individuals
mutations = random base shifts, deletions, insertions	**mutations** = random number shifts within the coding range
generation = period of time between sexual propagation, i. e. genetic recombination	1 **generation** = 1 program cycle that includes chromosomal recombination and judgment by the objective function
cross over: pairwise exchange of chromosomal material	**cross over:** pairwise exchange of data between two chromosomes
reproduction	**number of copies** of a single chromosome in the next generation

Since we try to make programs easier for a computer to process we choose a bytewise representation of the secondary structure. Nine different letters or numbers are used. An integer number from 0 to 8 represents one amino acid conformation each. An integer number in the computer's random access memory is nothing but a bit string consisting of 4 bytes = 32 bits (on the Power PC processor). One could call this a waste of memory, but the aim was to use the most standard variable type in the C programming language to make the program most portable. This setting virtually allows to encode as many different building blocks as the integer number can grow. We use a larger spectrum of numbers when we model database derived structural elements. The GA described by Goldberg worked on 0 & 1's only, but finally it used short integers to store either 0 or 1. Accessing the RAM bitwise is not a preferable method when using a higher compilable programming language such as C.

Table II.2.2.: *Standard conformations and their representations in computer memory*

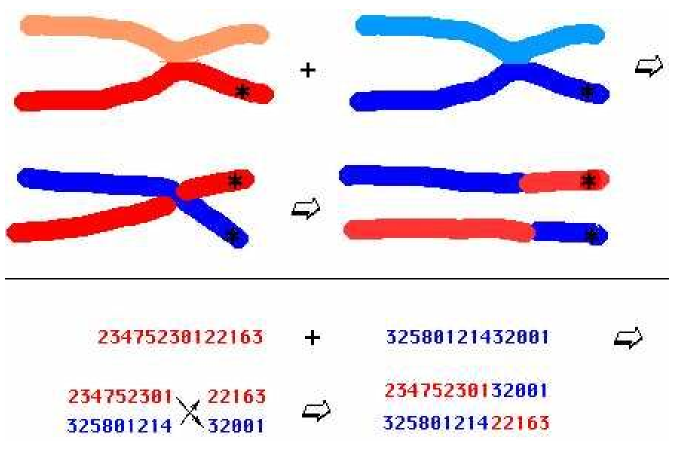
conformation code	**conformation**	**integer code**
A	alpha-helix	0
C	3/10-helix	1
B	beta-sheet	2
P	extended #1	3
E	extended #2	4
G	extended #3	5
Y	extended #4	6
N	extended #5	7
O	*cis-proline* (allowed only if the amino acid is a proline)	8

The synthetic evolution process starts from a chaos or, precisely spoken, with numbers that were generated by the computer's random number generator. A population of random integer arrays will result. Each individual within this population is a sequence of integer numbers and represents a protein structure. A skeptic may say that a GA is a random search. It is so only at start. Subsequently, selection leads to information storage in the individuals or chromosomes (table II.2.1.), and the search space is intelligently explored by always better adapted solution structures. The GA now uses a subroutine, called the objective function, to judge all individuals or chromosomes. Their fitness according to the problem is returned as a number of credit points. The objective function will be discussed later and we will now look at the GA core functions.

The GA decides how the chromosomes will reproduce. It determines, according to the fitness values, the number of copies that each chromosome will contribute to the new population. In contrast to nature the population size can neither increase nor decrease. Chromosomes with a low fitness value will be replaced with copies from fitter individuals. In analogy to nature a low fitness value should not set the probability of survival to zero. We realize this by the use of a roulette wheel selection. A chromosome with a high fitness value is given more winning numbers than a chromosome with a low fitness value. In most cases the fitter chromosomes will replace the worse but there can be exceptions since the roulette wheel is based on the random number generator.

The next step is mating. The chromosomes are crossed over at random positions and with random partners. Figure II.2.1. shows the analogy to nature.

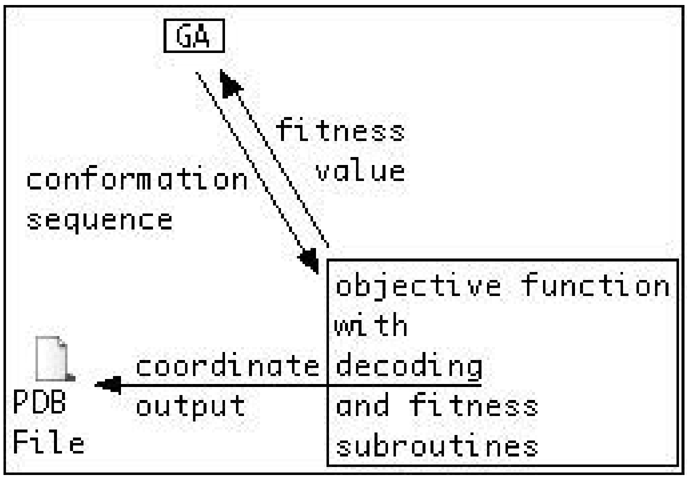
Figure II.2.1.: *Crossing over in nature and in the GA*



Generations and crossing over alone would already bring up results that are more than just random products. On the long run this can not be efficient since all solutions would depend on the first randomly generated population. As nature does we introduce mutations similar to those which occur in genomes. On a computer they are represented by random integer shifts within individuals of the population. Again, we use a random number generator to determine the occurrence of these shifts. The overall mutation rate per integer is determined by the user. In addition to these traditional functions of a GA we apply a synthetic breeding procedure. By doing so we try to save as much calculation time as possible. The breeding is realized by a backcrossing routine. The fittest offspring is conserved and will be brought back to the population by crossing over with randomly chosen individuals. The positions of backcrossing are determined randomly with a user defined probability.

It is important to understand the relevance of the GA in this application. The GA is the core search routine of the program. The fitness functions are the most complex part. The figure II.2.2. depicts the object hierarchy and shows that the GA is the master function. The box sizes represent the complexity of the routines mentioned.

Figure II.2.2.: *The GA in the program hierarchy and the comparison of complexity between the routines.*



II. 3. Threading Procedures

The GA is a robust optimization strategy. However, it can only perform well, if there is sufficient information to judge the quality of different solution trials and to develop the evolution in the correct direction further.

To this end, previous efforts from us and others optimized different fitness criteria and weights to judge protein structure and fitness by complex fitness functions and different weights.

This way is typical for a protein folding expert but involves a huge amount of time until optimal weights and fitness parameters are established and also long running times in individual simulation runs.

In this thesis, a new strategy is successfully implemented: Using a threading strategy we identify before the start of the GA simulation the optimal weights for the protein fitness function features according to the protein sequence for which the three dimensional prediction is desired. In other words: The correct environment (table II.2.1.) for the evolution of the protein structure to be predicted is chosen before the GA starts.

Compact and efficient implementation of the different fitness criteria (see next chapter) was established as well. In this way it was possible to get fast access to optimal weights for the simulation and run the simulation with low time effort for smaller proteins as a desktop application.

We examine to this end in this chapter the results obtained from a threading procedure that uses existing protein structure data for an automatized calibration process. We are in this way able to replace empirically determined maxima for most of our fitness functions. The program scans a set of PDB files from experimentally determined protein structures irrespective of their size and conformation. Since it is capable of predicting monomeric structures only, it reads the first chain from each file. Therefore the user should primarily provide monomeric structural information for a reliable calibration.
The coordinate data are transferred to a three-dimensional array which is judged by the fitness functions. Each function will return a value that corresponds to the structural characteristics of the known structure. These values are transferred to a file in order to create a set of expectable

values for native structures.

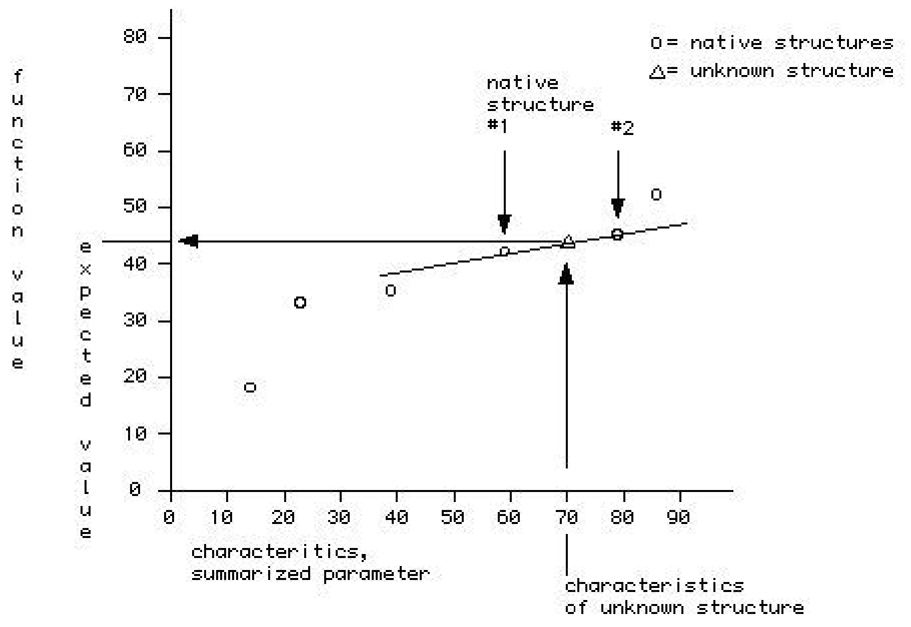
In addition to the function values the program saves additional general information for the native molecule. This information includes the PDB code of the molecule. This may be important to the user in case of errors during the calibration. An example could be a truncated PDB file that causes unexpected results. The calibration routines also need some general protein criteria to quantify the relation and similarities between the unknown protein structure and the data sets in the calibration database. These primarily include the length of the protein, the amino acid sequence and information on the secondary structure. Many PDB files in the Brookhaven database include detailed information on the secondary structure. The calibration routines will use the information supplied by the authors of the structure file only.

The calibration generates a single output file of a few kilobytes and allows quick processing of large amounts of data. One could call that process a very strong simplification of the protein database. This data set of rudimentary and general characteristics stores general building principles of the protein according to the fitness criteria. This means that the unknown structure will most probably not result in a model of a known protein using the sequence of the unknown one. It rather evolves into a specific solution for the new, unknown structure. An environment appropriate for the evolution of the sequence, in particular with suitable selection weights for the different fitness criteria, is chosen. Our threading procedure does not use the concept of homology modeling.

The algorithm is mainly intended to predict structures for small soluble proteins. Their length should be limited by only a few hundred residues. Having this in mind it does not make sense to include large molecules such as the proteasome in the database. For convenience large molecules will not cause any errors and unexpected results during the calibration. They simply won't be used for comparison because of their length. This leads to selection criteria suitable for native models during the calibration for an unknown protein.

The program reads the length and the other characteristics and summarizes the deviation of each native structure data set to the data set of the unknown protein. It chooses the two closest relatives to the unknown structure. These two are used to determine the expected values for all calibrated fitness functions. A linear correlation between the chain length and some parameters is assumed. In most cases both of the chosen native data sets will not largely differ in chain length, given a good interpolation result. Figure II.3.1. shows an example.

Figure II.3.1.: *simplified selection criterion for native structures from the database*



In this sample we only use six native structures for calibration. The structures #1 and #2 are the closest relatives to the unknown structure and have therefore been chosen to determine expected function value.

An expected value is estimated for each fitness function and is used to optimize this function. In contrast to prior versions (Dandekar and Argos, 1994; Saxena et al., 2001) the fitness value tells the GA the deviation from the expected optimum in percent. The GA finally optimizes the fitness functions to achieve values close to 100%.

III. Materials and Methods

This work was exclusively developed on a desktop computer. In this chapter the specifications of the system and the software that was used are listed.

Hardware

Most parts of the software were developed on a Macintosh computer with OS 9 running on a G3 PowerPC processor. However, the software has successfully been tested on IBM compatible PC's running MS Windows.

Programming Language and Compilers

The source code for the application was developed in ANSI C, mainly by means of the MPW shell source code editor from Apple Computers Inc., Version 3.5, 1999. The source code has been originally compiled with the SIOW library on the MrC C compiler included in the Macintosh Programmers Workshop software Package from apple. As the application designed as a shell-based program it has been successfully compiled with the GNU C Compiler (GCC) Version 2.95.2 on Microsoft® Windows®. Compilation was successful as well on OS 10.2 from Apple, here the version 3 of GCC was used that comes with the Developers Toolkit for OS X from Apple. The application is able to run native on the following computer environments:

- Mac OS classic versions on a PowerPC, i. e. mainly 8.x to 9.x
- Mac OS X / Darwin
- MS Windows including 95, 98, NT, XP

There is also a LINUX version of GCC, on which our application has not been tested yet. The 3D rendering environment does currently not exist for LINUX (see next paragraph).

Rendering and Display of Results

The PDB format is used for any kind of coordinate set, no matter if input or output. PDB files can be rendered to a 3D picture with a large number of software packages. Rasmol (Sayle and

Milner-White, 1995) was mainly used during development. In order to automatize the rendering and combine textual and graphical results the Chime plugin from MDL (MDL Information Systems, Inc. Chemscape Chime 2.0a; 1996-1998; based on Rasmol) is used in combination with the web browser Netscape Communicator (tested on version 4.5) or Mozilla (tested on version 1.0.1). The application creates all necessary HTML code during a simulation and the user only needs to open a master page. This page will then open subpages and embedded PDB files that display simulation characteristics and the resulting structures. The Chime Plugin is currently available for Mac OS classic and MS Windows only.

Brookhaven Protein Database

Our application uses large parts of the protein database (founded by Bernstein et al., 1977) for self calibration. The software package includes a list of all proteins that have been used for calibration. It may be extended or changed by the user at any time.

In addition, certain models have been used for reference and testing purposes. The PDB codes are the following: 1crn, 2gb1, 1dfn, 1erp, 1atx, 1pnh, 1ubq. Several of these small proteins are of biomedical relevance:

- 1crn - crambin: Plant seed storage protein, a storage protein
- 2gb1 - domain B1 of protein G, an immune system protein
- 1erp - ER-10: mating pheromone, a hormone
- 1ubq - ubiquitin: important among several other mechanisms for receptor stability and controlled protein degradation

Furthermore, two proteinous toxins were simulated in their structure:

- 1atx - sea anemone toxin from *Anemonia sulcata*: This toxin inhibits neuronal function by specifically blocking the sodium channel. It delays inactivation of the sodium channel and thereby stimulates the mammalian cardiac muscle contraction.
- 1pnh - scorpion toxin from *Androctonus mauretanicus*: This molecule has affinity for the ampamine sensitive potassium channel (also known as small conductance calcium-activated potassium channels) and binding to this channel inhibits neuronal function.

Superposition Routine and RMSD calculation

The application provides an RMSD calculator which superimposes two protein structure PDB files and then calculates the RMSD between them. The superposition and RMSD calculation routines were taken from Goodfellow JM, Moss DS (1992, refined by Walter D., 1996, EMBL Heildelberg, unpublished). Both were slightly modified to fit the application context.

IV. Results

IV. 1. Grid Free Building Block Protein Representation

We have explained in the introduction and principles section that protein folding is a complex phenomenon, on the other hand a simple representation or encoding is critical since we want to achieve a fast tool for desktop computers. We prefer to reach a good resolution as well as detailed representation of the molecule with a minimized amount of calculations. A lot of effort has been made on this topic and solutions such as protein representation on a fixed grid (Skolnick and Kolinski) have been developed. Other ways of representation are reviewed by Schultz and Schirmer (1979) e. g. to describe the protein structure in cylindrical or polar coordinates. Furthermore, a tree-like representation of peptide structures was introduced by Abagyan and Totrov (1994).

Building on previous experience (Dandekar & Argos, 1994, 1996; Saxena et al., 2001), we test here a grid free representation of protein building blocks found to be useful in our previous studies on protein folding and prediction, In this strategy, we answer the search for a simple but still versatile protein representation system with a building block mechanism, originally introduced by Rooman, Kocher and Wodak (1991). Each of those building blocks consists in a pair of two dihedral angles, phi and psi, and an additional omega angle to model a cis-proline. If we use this modeling procedure we are able to represent helical, stranded and looped conformations in protein folding simulations as it has been shown by our group before (Dandekar and Argos, 1994).

As we will see later, in this thesis we successfully introduce a new ingredient: Specific turn conformations are critical for modeling of TM proteins. They are introduced in a way which does not unduly increase the conformational search space as may be the effect from introducing general additional conformations (e.g. according to Karplus, 1996) as investigated earlier by Du and Dandekar (presented at Molecular Modeling Workshop Darmstadt, 1997). Our application generates a library with possible loop conformations that may potentially be integrated into transmembrane protein models. Recent work from Forrest et al. (2003) has shown that libraries for linking loops are an important tool for fold prediction in this class of molecules. In contrast to their work we generate our library based on systematic plotting the allowed regions of the Ramachandran map. This algorithm is much less calculation intensive than the method applied by Forrest et al. (2003) and therefore suitable for desktop computer purposes.

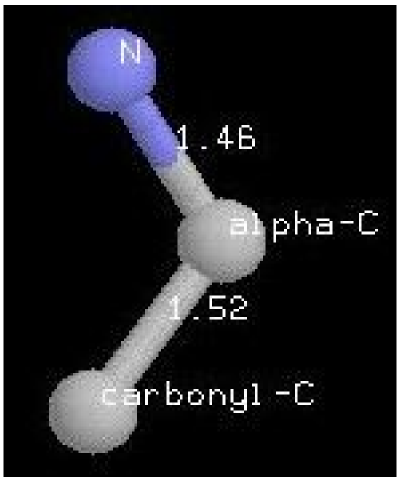
The building procedure for each of our models starts at the coordinates [x, y, z] = [0, 0, 0] with the N-terminal nitrogen of the protein. It continues with the alpha carbon and the carboxyl carbon and so on. Looking at any set of coordinates that was created in that manner it would look like the following extract in PDB format (table IV.1.1.)

Table IV.1.1.: *Beginning of a PDB encoded protein that starts with aspartic acid (ASP). It was generated with the grid free representation technique.*

```
ATOM      1  N   ASP     1       0.000   0.000   0.000
ATOM      2  CA  ASP     1       1.470   0.000   0.000
ATOM      3  C   ASP     1       1.980   1.443   0.000
```

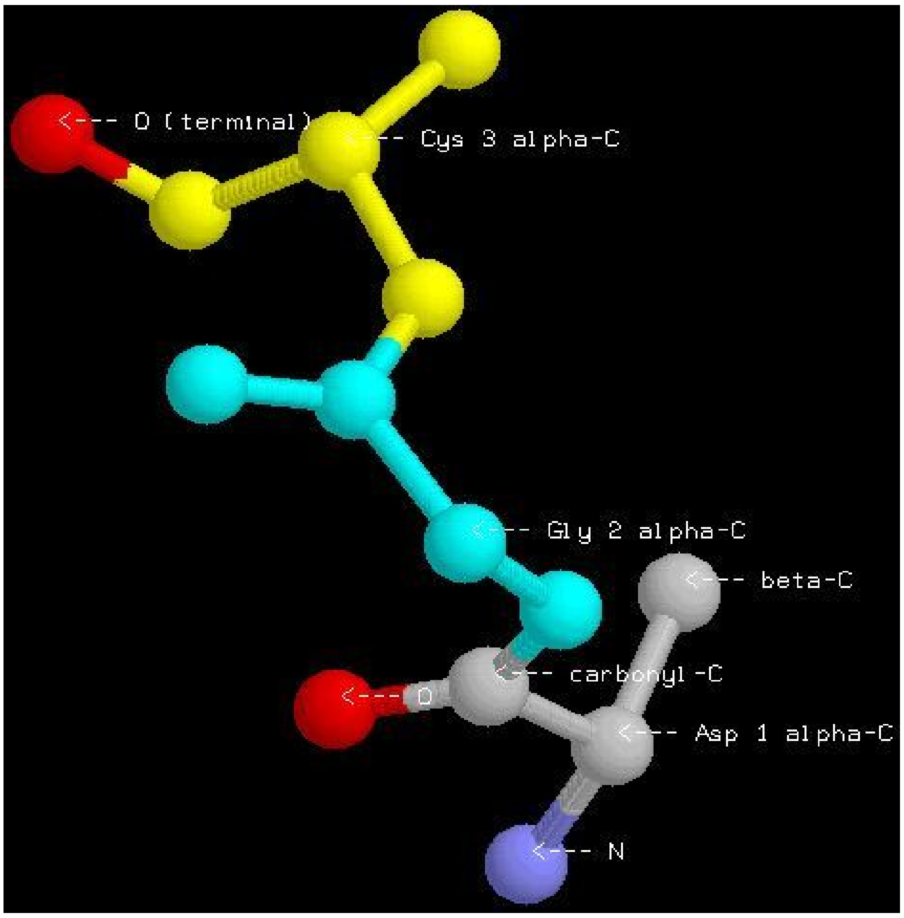
All distances are given in Angstrom (10Å = 1nm) and one can now easily recognize the distance between N and CA (alpha-carbon) Figure IV.1.1. shows a rendered model of the three atoms from table IV.1.1.. The PDB format allows rendering with any PDB viewer such as the Chime Plugin from MDL. (The online HTML version of this document contains a 3D model for Figure IV.1.1.. Click on the snapshot to open a new window with the 3D representation if you have the Chime Plugin installed.)

Figure IV.1.1.: *The N-terminus in three dimensional representation*



To make the representation more accurate we add the oxygens and beta-carbons to the model. Oxygens and nitrogens may serve as electron donor and acceptor pairs in order to scan for possible hydrogen bonding. The beta-carbon enables us of applying fitness functions which look for interactions between side chains. The entire side chains are not represented. Figure IV.1.2. shows a hypothetical tripeptide in AAA conformation, i. e. a very short helix (The HTML version allows visualization with the Chime Plugin through a click on the snapshot). The sequence of the peptide is ASP-GLY-CYS.

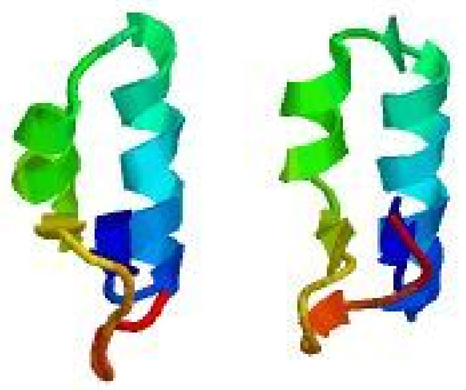
Figure IV.1.2.: *The model in three dimensional representation*



Please note that there is no beta carbon atom for glycine. The algorithm adds these atoms to all amino acids except for glycine which only has a hydrogen as side chain.

Representation of whole proteins is possible with this method as well and figure IV.1.3. shows a model of the plant seed protein Crambin (PDB code 1CRN) compared to the experimentally derived crystal structure. This model was generated by our structural prediction system.

Figure IV.1.3.: *left model: 1CRN original structure; right model: program output, 4.77 Å RMSD*

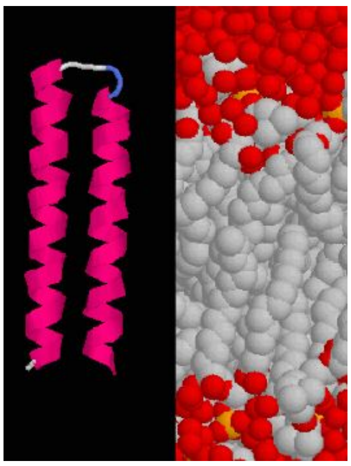


With the set of standard conformations we are easily able to represent structures of small soluble proteins.

However, to allow more accuracy for transmembrane proteins a specific library of TM-conformations is specifically introduced for protein modeling in this thesis.

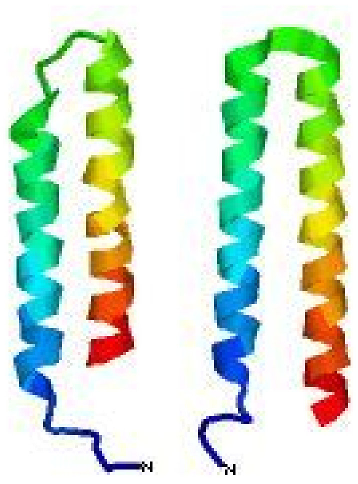
When we intend to model helical transmembrane proteins we apply a loop database which is based on a combination of standard conformations and an extra pair of angles derived from the RAMACHANDRAN map. Figure IV.1.4. shows one of the test models during database creation. The helices may span a phospholipid bilayer membrane, a model of which is depicted at the right side.

Figure IV.1.4.: *left side: two transmembrane helices and a linking loop; right side: phospholipid bilayer membrane surrounded by water.*



Indeed it is possible to represent a 7-TM receptor molecule by the loop database extended set of building blocks. In figure IV.1.5. we compare the N-terminus of a bacteriorhodopsin simulation to the N-terminus of the experimentally determined structure.

Figure IV.1.5.: *right: bacteriorhodopsin structure from the PDB (1QM8), residues 2 to 62 compared to left: generated model with application of loop database; RMSD = 6.496 Å*



We may conclude that it is possible to represent protein molecules with the building block system in a way that is accurate enough for the simulation of smaller chains. Extensions of the building blocks make the program more flexible so that conformations typical for TM proteins, such as linking loops, may be added.

The building block grid-free modeling system only requires a string of identifiers. This string is directly converted into a set of coordinates for every represented atom of the protein. The GA is able to optimize these strings by means of the fitness functions. These functions operate on the atomic coordinates and measure distances between atoms or points of the coordinate system. The orientation of the model inside the coordinate system is irrelevant for these calculations.

For an efficient computer implementation of this method we use several dynamic arrays to store the values. We need an integer string that represents the current conformation of a model in the synthetic evolution. These strings are stored in a two dimensional array. Coordinate output is temporarily stored in a three dimensional floating point array and the amino acid sequence as well as secondary structure data are stored in linear character arrays. If needed, the loop database is loaded into a linear floating point array. All those arrays are allocated dynamically leaving the upper limit open to the physically available size of memory. Finally we are able to write the coordinate values of our models to PDB files. This file format is standardized and allows the user to process the program output with any other application.

IV. 2. Capabilities of the GA

We have shown in the previous chapter that we have achieved in this work to represent protein molecules by simple integer strings and now tackle the problem of fast detection of suitable folds. There are large numbers of possible folds in the conformational search space and we are interested in finding the few correct ones. The way of complete evaluation is not feasible since we want our program to run on desktop computers. There are many models and hypotheses on mechanisms of protein folding which is clearly an important topic. Several genetic and sporadic diseases, such as cystic fibrosis or ALZHEIMER's dementia are caused by protein misfolding (Cohen and Kelly, 2003; Dobson 2003). The problem is that these models and theories are in general much too complex to implement in a desktop computer application. However, recent literature indicates that there are probably not more than 10'000 folds in the protein universe (Koonin, Wolf and Karev, 2002). Despite this fact, protein fold prediction and kinetics still remain a problem that requires multiple solution strategies. Work from Snow, Nguyen, Pande and Gruebele (2002) presents a *in silico* (i. e. computed) simulation of protein folding for several nanoseconds and compares it to the *in vitro* results. Similarly, Yang and Gruebele (2003) have chosen a small, quickly self folding protein for their simulations. In contrast, we do not to simulate the complete process of protein folding, our main output remains the final structure. Of course, it is more consistent with reality to simulate the folding mechanism instead of trying to predict a fixed structure, but the calculation speed aspect was not considered as important in the two above mentioned studies.

Many researchers have found that Genetic Algorithms are suitable tools to approach complex functions which require an undetermined and open time to arrive at a solution (NP complete problem) (Corne et al., 2003). Indeed the GA is able to evolve towards maxima for functions that have never been solved before. For example, identification of complex alignments (SAGA algorithm, Notredame et al., 1996), prediction of new protein structures (Dandekar and Leippe, 1997; Saxena et al. 2001), as well as other complex and new optimization problems in economy and engineering have been elucidated by GAs. Only by means of descriptive data it is possible to describe the function and then let the GA find the optima. The GA is an analog to nature and represents a fast method for systematic information accumulation. Like nature it is a robust tool that is likely to run stable even in case of difficult simulations.

The effectiveness in search that is achieved by a GA mainly depends on the quality of the objective function. In this thesis we implemented a new set of fitness criteria which are suitable for desktop computation in terms of memory usage and speed. We show that these simple criteria make our GA a robust core application which is capable of representing protein structures well. The function is able to separate and identify correct structures for short protein chains without any fitness parameter weight determination or adjustment by the user. The overall application speed makes usage on a desktop computer possible.
We present here the results of different simulations and the GA search strategy to show that successful modeling of protein structures with known crystal structure as golden standards was well achieved. Subsequently we will discuss the different fitness criteria.

In our trials the algorithm in table IV.2.1. has shown to be useful. The original concept was taken from Goldberg et al. (1989) and has been extended in this thesis by a synthetic breeding procedure. This backcrossing method takes place in steps eight and nine.

Table IV.2.1.: simplified flow scheme of the GA

STEP	
1	generate random population
2	cross over pairs of two individuals with each other at random positions (The population size is always an even number.)
3	call the objective function for each individual to get the fitness values
4	distribute roulette wheel winning numbers according to fitness
5	run roulette wheel
6	distribute propagation space in new population and fill it with copies from the winning individuals
7	mutate if random generator indicates it while copying individuals to new population
8	find out if there is a fitter individual than the individual that has been the fittest so far...
9	... if this is the case replace the so-far-fittest with the new fitter individual
10	backcross the fittest individual if triggered by the random generator
11	go to step 2 until the number of generations given by the user is reached. *(We recommend to run as long as a increasing gain in fitness may be achieved. Crambin with approximately 50 residues requires about 100 generations.)*
12	give out the fittest individual that has ever occurred.

A GA may be used in a standardized manner to optimize nearly any solution for a specific problem. However there are several parameters that must be adapted to the problem individually. If one misses this step the GA may be able to optimize in the course of the simulation but the performance in terms of the solution achieved will be low. Parameters which must be adjusted according to the GA include such elementary values as the number of generations, the population size and the mutation rate. For the breeding procedure we also give a backcrossing probability. The easiest way to illustrate the importance of these values is to run the GA on a small protein such as the mating pheromone (1ERP). Our program features a fitness log file that is presented together with the simulation results. These log files may be copied to spreadsheet applications with graph generation routines. The visualization is helpful in determining the performance of the GA with specific settings for the variables mentioned in this paragraph. In addition, these logs may be evaluated if the application is used for teaching purposes. The following table (IV.2.2.) and the figures IV.2.1. to IV.2.4. show four examples with different settings in mutation rate and backcrossing.

Table IV.2.2.: *Parameters for GA demonstration in figures IV.2.1 - IV.2.4.*

Test:	**1) no mutation and low backcrossing**	**2) no mutation and normal backcrossing**	**3) high mutation and normal backcrossing**	**4) normal mutation and backcrossing**
population size	500	500	500	500
number of generations	100	100	100	100
mutation probability per integer copy	0 %	0 %	10 %	0.1 %
backcross probability per generation	25 %	50 %	50 %	50 %
average fitness at start (= random)	17.40 %	17.75 %	18.14 %	18.76 %
average fitness reached	81.45 %	83.71 %	5.87 %	76.78 %
maximum fitness reached	81.45 %	83.71 %	78.36 %	86.70 %

Note: The maximum and average fitness values strongly depend on the settings within the objective function and it's subfunctions. They only represent the quality of the GA in this example and do not indicate the quality of the protein structure.

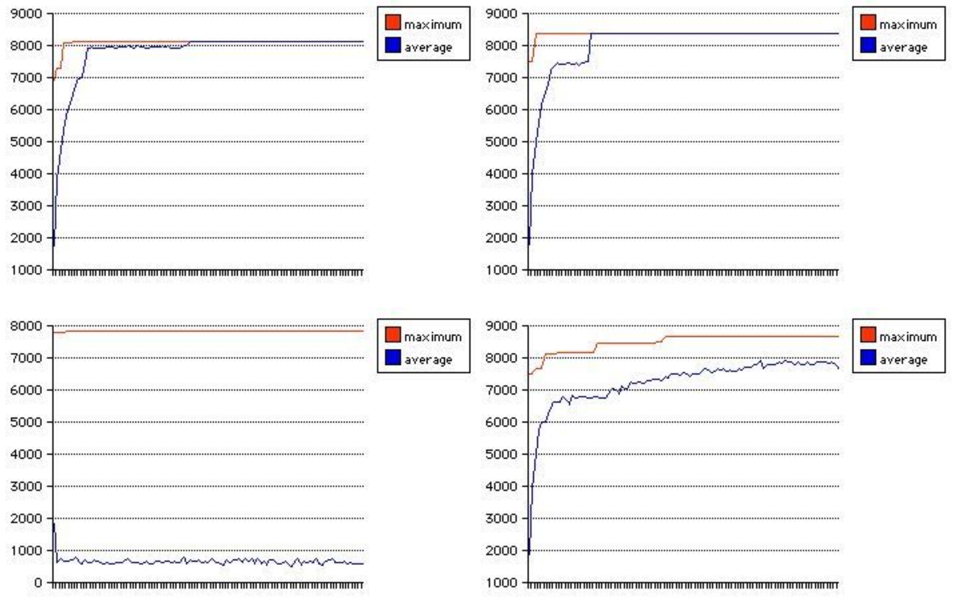
Figures IV.2.1 - IV.2.4. *from left to right /top to bottom:*

1) low mutation and low backcrossing

2) low mutation and normal backcrossing

3) high mutation and normal backcrossing

4) normal mutation and backcrossing



The maximum line represents the actual best solution, the x-axis indicates the generations. On the y-axis the actual fitness points are measured where 10'000 equals the maximum fitness that can be reached. The average graph represents the mean fitness value of the whole population which starts out of the chaos from a low level in each simulation.
Each simulation was run for 100 generation and 500 individuals.

Figure IV.2.4. represents a good setting for mutation rate and backcross probability. This is pointed out by the following features:

1. The maximum value continues to rise for a long time - much longer than in the three other simulations. It stops increasing at about 50 generations. This indicates that for further modeling with these settings it would most probably be enough to run the evolution for only 50 generations instead of 100.
2. The average graph never touches the maximum graph. This is due to mutations, indicating sufficient diversity of the solution trials. One can recognize this behavior with a high mutation rate in an extreme manner (Figure IV.2.3.) whereas one does not find it without mutations (Figures IV.2.1. and IV.2.2.)

3. The speed aspect becomes clear when we compare the graphs in figure IV.2.1. which has a low probability of backcrossing compared to IV.2.2.. The mutation parameter has been excluded (no mutations allowed) for this comparison: This leads to a fusion of the maximum and average graph, i. e. every individual in the population becomes identical from the point of fusion. Since there is no further information change due to the missing mutations, the fitness does neither increase nor decrease. The GA operations like crossing over become futile from this point. The only difference between the two graphs is the moment when the fusion occurs. In other words a higher rate of backcrossing makes this event happen earlier. It is clear that a graph fusion is equivalent to a booster for the maximum fitness in combination with a normal mutation rate. And it is important to achieve these boosters as fast and often as possible to save calculation time. Of course, excessive and inadequate use of backcrossing bears the risk of trapping the population in local minima.

From empirical observations we propose several suitable default values for the basic GA parameters in our application software. The user may apply these values but may freely decide to use other values as well. It is important to adjust these parameters according to the size of protein structure that should be predicted. The calculation time per model does not increase in a simple linear manner per residue added to the protein. Linearity is true for the standard GA operations only since any residue or integer in the representing bit string will be treated similarly. In reality the calculation time increases in a more exponential manner. The fitness functions measure distances all over the molecule and the number of necessary calculations dramatically grows with the chain length. We summarize that for short protein chains there is a variety of choices for population size and number of generations. The total calculation times will differ in terms of just seconds which is irrelevant to the user. As soon as the structures to predict become longer than two hundred residues, the calculation times will differ by several minutes which indeed is important for an effective workflow with a computer application. Our application always tries to predict the running time, the computation of which is more or less accurate depending on the computer system. This may help the user to change the values adequately before making the mistake of waiting a long time for a result.

In general we propose a population size not smaller than fifty individuals and not bigger than a few hundred. The mutation rates should be increased when the population size is decreased.

A good value is 0.01 (1%). The backcrossing probability has proven effective in time reduction for middle sized populations (50-100 individuals) with values around 0.1 (10%).

IV. 3. Threading and GA Combination

Fitness determination is critical for the selection procedure by the GA.

The crucial and most challenging factor in evolutionary computation is a suitable fitness function. Complex tasks such as protein structure prediction usually require extended knowledge and large numbers of empirical tests in order to achieve a reasonable selection. This selection is the heart of any genetic optimization procedure.

Our group has shown earlier that empirical testing is a possible way in determination of fitness function weights for the use in a GA based application. Now, since we intended to further develop the protein structure prediction for laboratory usage with desktop computers we have decided to automate the fitness weight determination as complete and, at the same time, as protein specific as possible.

If we look at our previous versions that use a GA we find that the user applied empirically determined fitness parameters and weights to all proteins desired to predict. We now examine a solution that is able to run without this input. We have combined the potentials of protein database threading with the GA to build a tool for automated predictions with protein specific weights.

In contrast to complex protein structure simulations that after extensive computations can be more accurate than ours (e. g. TOUCHSTONE, PROSPECTOR; Skolnik et. al e. g. 2002, 2003), another example is the ROSETTA algorithm (Bradley et al., 2003). However, none of these applications seems to be suitable for desktop computer use. Our application is ready for desktop computer use and still, it yields quick and topologically satisfactory results. Furthermore we describe the implementation of several extensions allowing more accurate and faster prediction under special circumstances. These include helical TM proteins with their linking loops and inclusion of experimental data as distance constraints. We obtain the potential for predictions that are similarly accurate as above if large scale simulations, including threading, are attempted.

We have introduced above the combination of a threading procedure coupled to our GA as a new way of solving the problem of fast and automated structure prediction. The program takes advantage of the continuously growing protein database. in contrast to homology search it does not copy parts of known structures in order to get template motifs for the molecule in question. Our threading procedure is a way of bypassing the necessity of empirical fitness function weight determination. It stays adapted to current protein structure knowledge by re-calibration and thus provides an efficient method for structure prediction without time consuming user input.

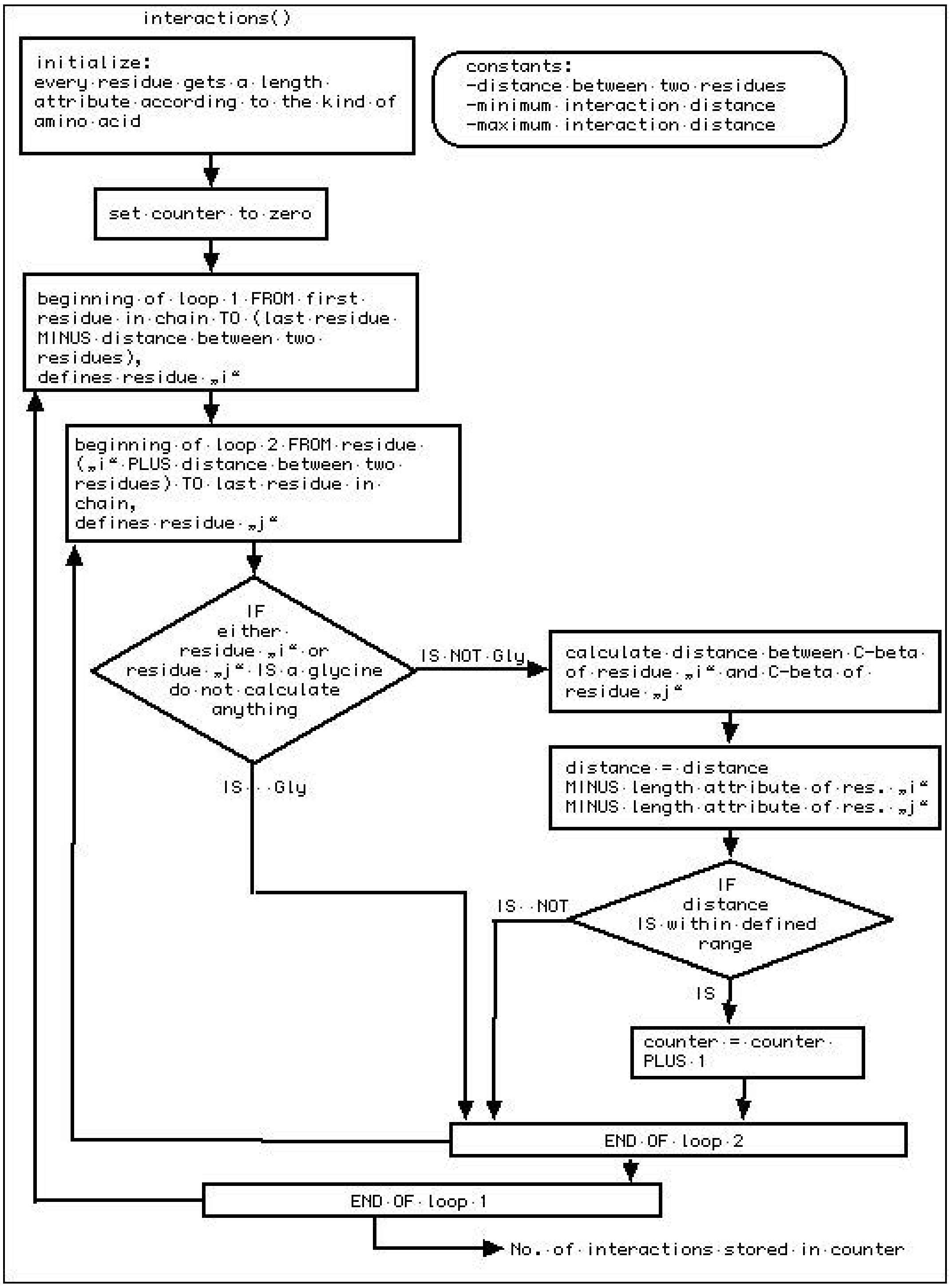
Fitness Function parameters

Different parameters or criteria judge the quality of a protein structure during simulation. However, the absolute values of fitness parameters alone are not meaningful. It is difficult to write fitness functions in such a way where the resulting value indicates higher fitness by a higher or lower value directly. The problem consists in the missing information on the correlation between the function value and a correct structure.

The newly implemented fitness parameters return function values that are context defined. Optimization towards a maximum or minimum would never succeed here. Instead we test known protein structures from the PDB against these functions and store the resulting numeric values in a database file. This process derives threading based optimal values for the fitness parameters to select the best 3D prediction trials for a given sequence. It effectively serves for the calibration of our fitness functions. The program is now able to determine expected function values and optimizes solution models in order to achieve them. Some of the calibrated functions have turned out to show a linear correlation between the chain length of a protein and the function value. Even though one might propose to add this linearity to the function and use it without calibration afterwards we do not convert it that way. The threading procedure is much more flexible than the method with fixed fitness functions since the increasing data from the PDB may always be included for calibration.

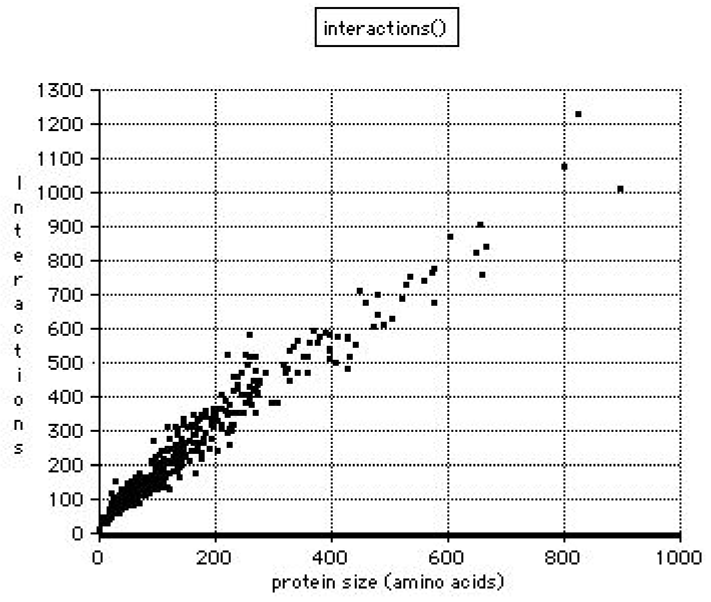
The interactions fitness parameter is a good example to explain the capabilities of our threading and GA combination. Figure IV.3.1. shows a flow diagram of the function.

Figure IV.3.1.: *Flow diagram of the interactions function*


interactions()
initialize: every residue gets a length attribute according to the kind of amino acid
constants: -distance between two residues -minimum interaction distance -maximum interaction distance
set counter to zero
beginning of loop 1 FROM first residue in chain TO (last residue MINUS distance between two residues), defines residue „i“
beginning of loop 2 FROM residue („i“ PLUS distance between two residues) TO last residue in chain, defines residue „j“
IF either residue „i“ or residue „j“ IS a glycine do not calculate anything
IS NOT Gly
IS Gly
calculate distance between C-beta of residue „i“ and C-beta of residue „j“
distance = distance MINUS length attribute of res. „i“ MINUS length attribute of res. „j“
IF distance IS within defined range
IS NOT
IS
counter = counter PLUS 1
END OF loop 2
END OF loop 1
No. of interactions stored in counter

The resulting function value will be a number representing the amount of intramolecular interaction. Although it does not correlate with any specific scale it is possible to evaluate these function values by comparison to structural features of unknown protein folds. Such a feature is the chain length which the user already knows from the sequence. Figure IV.3.2. depicts the correlation within large parts of the PDB.

Figure IV.3.2.: *Function values from the interactions function compared to chain length of known proteins*

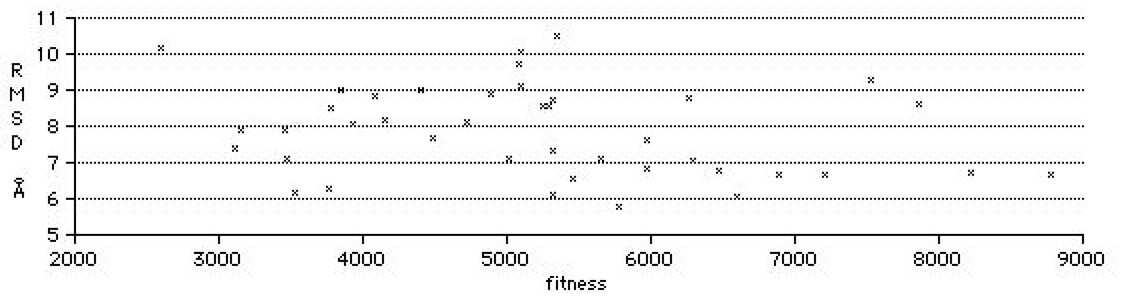


Our application relies on several fitness parameters since the description of a structure is too complex to be put into a single criterion. Similar to prior versions we use a certain number of fitness criteria. In contrast to those versions we try to keep their relative weights equal. This standardized setting simplifies the use of the software especially if there are no precise data on the unknown structure. Since the fitness functions do all return values from zero to one hundred percent there is no need for adjusted correction or weight factors. As soon as the user gets more information on the protein, e. g. through the simulation output, he may choose to emphasize on certain parameters more than on others. The user may now adjust the weights in terms of an increment in the factor for one fitness function and decrement of another. In case a criterion turns out to be difficult to optimize for the GA, it may also be given a higher weight.

We did experiments on automated weight adjustment during simulation. These trials did not succeed since continous change of the environment for the GA interrupts the optimization. These events can be compared to harsh climatic changes in the real world where even well-adapted species would die out. Exactly the same happens to specialized solutions in the synthetic evolution that evolve while a certain fitness criterion has a high priority. These circumstances suggest that it is a wise decision to keep weights stable during these folding simulations and to keep them equal in order not to overrate a single criterion.

Using standardized equal weights for four of our fitness parameters we find low RMSD structures at the end of the simulations for the Crambin molecule. Figure IV.3.3. shows a plot from hundred simulations that all used the functions 'H-bond formation', 'interactions', 'beta interactions' and 'loops' with twenty five percent weight each.

Figure IV.3.3.: *Plot of RMSD of 100 standard simulations of Crambin and the fitness values*



We see that in this example there are individuals with a low RMSD in the medium fitness regions. These regions also show a large scatter of RMSD. If this region is compared to the higher fitness region the scatter disappears. It may be concluded that molecules with a high fitness will turn out to have a low RMSD.

The advantage of the new combined algorithm is the automatic adaptation of optima for each different protein that is simulated. A medical application of this method is the fold prediction of small toxin proteins that mostly are ligands to larger macromolecular cell components.

IV. 4. Important Subroutines

An optimized and completely new implementation of fitness functions was necessary for the application software presented in this thesis. These criteria partly are further developed modifications of functions that were used previously (Dandekar and Argos 1996, Saxena et al. 2001). They were tested in numerous runs. For clarity and simplicity their final form yielding best results in our simulations are given here only.

At this point we describe the most important subroutines of our application. Most of these routines are called for a standard prediction run. They are part of the result of our research and development. We describe them in order of their occurrence if this is possible. The following listing should give the reader an overview of the core functions that we have integrated. This chapter mainly contains simplified descriptions of the program subroutines which may usually not described in form text. Basic computer needs are not discussed here and the summaries usually are not complete. Such basic need include e. g. memory and file management as well as variable and pointer types. This information is part of the digital appendix of this work which includes a copy of the commented source code.

Subroutine `void chromosome_generator();`

This subroutine randomly generates a population of solution trials. The user has defined a population size, and the protein that will be modeled determines the length of the string that will be used. These values are provided to the `chromosome_generator()` function through global variables. To illustrate what this routine does, let us imagine a protein with 10 amino acids, all glycine for simplicity, and one tiny alpha helix - an assumed result from a secondary structure prediction - in the middle. This is artificial, of course, but will come out very clear.

The secondary structure data which were entered by the user would look like this:

```
uuuAAAAAuu
```

And the amino acid sequence is quite simple:

```
GGGGGGGGGG
```

With a set of the first four building blocks (chapter II.1.), the GA uses one integer number for each block which makes a total of 10 integers for each individual. We exclude the extended conformations from this example for simplicity.

The random number generator is called and produces the following solution trials:

Figure IV.4.1.:*10 random solution trials*

```
3110310302  =>  PCCAPCAPAB
2313301331  =>  BPCPPACPPC
1231100130  =>  CBPCCAACPA
0111111032  =>  ACCCCCCAPB
1130320011  =>  CCPAPBAACC
0213001201  =>  ABCPAACBAC
1323101322  =>  CPBPCACPBB
3130313012  =>  PCPAPCPACB
2021112301  =>  BABCCCBPAC
1312000100  =>  CPCBAAACAA
```

These would not be correct with the given secondary structure prediction in mind. Although it is possible to let the GA find the helix by running it for many generations, every redundant calculation should be avoided. For that reason we call the routine that is described next.

Subroutine `void secondary_formation();`

This subroutine adds given secondary structure data to the random solution trials by overwriting these regions. The routine replaces the random patterns in locations were a secondary structure prediction is given. This does not mean the GA will be unable to change these at all. It is allowed to change everything if necessary to reach a higher fitness. There will be an exception to this adjustment but for the moment we will not consider it. Since the secondary structure pattern is checked by one of the fitness functions, the corresponding value will start from a higher level when the `secondary_formation()` routine is called before the GA starts. Figure IV.4.2. shows the result of `secondary_formation()` after it has processed the data from figure IV.4.1..

Figure IV.4.2.: *The 10 random solution trials after the* `secondary_formation()` *routine has been called*

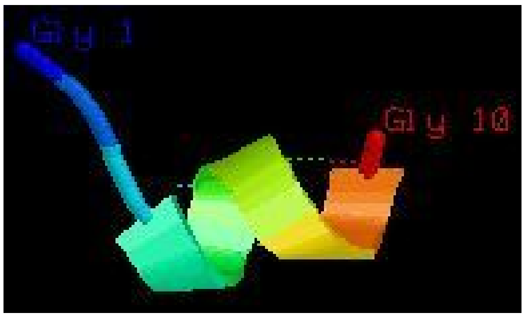
```
uuuAAAAAuu       uuuAAAAAuu

3110000002  =>   PCCAAAAAAB => model shown in figure IV.4.3.
2310000031  =>   BPCAAAAAPC
1230000030  =>   CBPAAAAAPA
0110000032  =>   ACCAAAAAPB
1130000011  =>   CCPAAAAACC
0210000001  =>   ABCAAAAAAC
1320000022  =>   CPBAAAAABB
3130000012  =>   PCPAAAAACB
2020000001  =>   BABAAAAAAC
1310000000  =>   CPCAAAAAAA
```

To visualize the corresponding PDB files to these ten trials they are included in the online version and can be visualized with the chime plugin. (click here to open the files in a new window).

Figure IV.4.3.: *rendered structure from the first trial in figure IV.4.2.., i. e.*

```
3110000002  =>   PCCAAAAAAB
```



In order to avoid misunderstandings with these two routines it must be said that they can not convert the integer string data into building block sequences nor do they give Cartesian coordinate output. The data presented here have been obtained from an original run of the complete application with all representation subroutines. The conversion to the final output format, i. e. PDB has only been done to illustrate what happens with the integer strings.

Subroutine `void decode_conf_solo();`

This function is needed to build up a set of Cartesian coordinates for the current solution trial using the GA determined building block sequence. This routine is the bridge between the genetic algorithm and the protein representation. Its input data are a chromosome consisting of integer values and the amino acid sequence. A complete set of Cartesian coordinates is generated from these data. It includes all atoms that are represented in the simulation (see chapter II). This set of coordinates is stored in form of floating point values in a three dimensional array. The three dimensions in this array do not represent the three spatial dimensions. The values are encoded as follows (Dandekar & Argos, 1994):

```
xyz[AA][REP][coordinate] = float point value
```

`AA` = residue No.

`REP` = number of atoms in residue using the following coding table

0: N
1: C-alpha
2: C'
3: O
4: C-beta

`coordinate` = kind of coordinate: x, y or z

0: x
1: y
2: z

The kind of amino acid is interesting to decide whether to build a C-beta or not in case of glycine. In addition there is the conformation for *cis-proline*. In some cases this must be used for proline. Otherwise there is no difference made between the amino acids.

The calculation of coordinates (Dandekar and Argos 1994) starts at the N-terminus of the protein. It continues towards the carboxy-terminus. This explains why the first nitrogen always gets the coordinates x,y,z = 0,0,0. Basically the first residue of any simulation will have the same coordinates. The first phi, psi and omega angles come in when the routine reaches the second residue. The calculation is stereotypic from that point. One could imagine someone putting preformed building blocks, defined by the three angles, together. The GA chromosome serves as template. The calculation mainly consists of three dimensional geometry and vector operations.

Since the position of any amino acid must be calculated relative to the previous residue it becomes clear that the coordinates may only be calculated as a whole set in a sequential manner. It is impossible to calculate the atomic coordinates for a single residue separately.

Subroutine `void calibrate();`

This is the master function that calibrates the system by execution of the threading process. Calibration tests fitness functions on known crystal structures. The calibration routine applies the fitness subroutines to known crystal structures from the protein database (PDB). Coordinate data are transferred to the common x, y, z array. Amino acid and secondary data are translated into strings and stored in the corresponding arrays. The fitness subroutines are called sequentially on this set of information as if it was a structure returned by the GA. The achieved absolute values for each crystal structure must not be confused with fitness values. They will be written into a file. These values are raw numbers. A short example will point this out clearly:

The `interactions()` routine counts the number of interactions between amino acid residues. The result returned by `interactions()` is a number like "15". This value will be stored for the calibration sample protein in the calibration file. Browsing the database, the algorithm will store the corresponding values for each of the proteins used for calibration. This value will be of general importance during a simulation.

Some descriptive information on the proteins that are used to calibrate is absolutely necessary to conserve within the calibration file. This information includes:

- location / path to the PDB file
- length of the protein

- amino acid sequence
- secondary structure data read from PDB file if present.
- number of hydrophobic residues - the program has already counted them, so we store them since the value will be needed later anyway

The following data are the fitness criteria information. Everything in the calibration data file is written in a machine-friendly format without comment lines. Finally the file contains the following fitness data for each structure used. Although absolute values for some of these criteria may turn out not to be required in a specific simulation their calibration data are listed here, too. Additional criteria will be added if the simulations are developed further.

- path
- length
- sequence
- `tertiary()` value
- `hydrophobicity()` value
- number hydrophobic residues with glycine
- number of glycines
- number of H-bonds
- scatter histogram "all" values: 10 numbers
- scatter histogram "fatty AA" values: 10 numbers
- number of interactions between residues

The PDB code "TER" terminates the input from the PDB file. That is where the first chain terminates. If no "TER" tag is found the file is read until EOF occurs.

A new text file called "calibration.out" is created to store the resulting absolute values in sequential manner. The data are encoded using the following convention, here the example from 1erp:

Table IV.4.1.: Format convention for the "calibration.out" file.

Line	**Parameter**	**Value**
# 1	filename:	path to 1erp.pdb
# 2	chain length:	38
# 3	AA seq:	`DLCEQSALQCNEQGCHNFCS PEDKPGCLGMVWNPELCP`
# 4	secondary seq:	`uAAAAAAAuuuAAAAAAAAu uuuAAAAAAAAAAuuuuu`
# 5	tertiary:	103.062553
# 6	hydrophobicity:	29.000000
# 7	No. of hydrophobic AA:	16
# 8	No. of Gly:	3
# 9	No. of H-bonds:	0
# 10	scatter histogram 1:	0
# 11	scatter hist.2:	0
# 12	scatter hist.3:	10
# 13	scatter hist.4:	7
# 14	scatter hist.5:	13
# 15	scatter hist.6:	31
# 16	scatter hist.7:	7
# 17	scatter hist.8:	13
# 18	scatter hist.9:	7
# 19	scatter hist.10:	7
# 20	scatter hist. fatty 1:	0
# 21	sc. hist. f. 2	0
# 22	sc. hist. f. 3	15
# 23	sc. hist. f. 4	7
# 24	sc. hist. f. 5	7
# 25	sc. hist. f. 6+	23
# 26	sc. hist. f. 7	7
# 27	sc. hist. f. 8	15
# 28	sc. hist. f. 9	7
# 29	sc. hist. f. 10	15
# 30	inter_quotient: [inter_quotient = interactions() / lastresidue]	0.263158

This information should not be manipulated by the user. In case of unwanted data sets it is recommended to remove the corresponding filenames from the "calibration.cal" file and to calibrate again.

The flow scheme for the database generation is given in table IV.4.2.:

Table IV.4.2.: *Flow scheme of the database generation*

STEP	
1	open "calibration.cal" for input
2	read one line until carriage return
3	open path indicated in STEP 2, i. e. PDB file
4	read PDB file until "TER" tag or until EOF and count residues, determine whether this file is longer than the previous
5	go to STEP 2 until EOF occurs in file "calibration.cal"
6	close "calibration.cal"
7	allocate as much memory as the longest model will need
8	open "calibration.cal" for input and "calibration.out" for output
9	read one line until carriage return
10	open path indicated in STEP 9
11	read PDB file into array in memory
12	apply all calibrated fitness functions
13	write the function values to "calibration.out" according to the format in table IV.4.1.
14	go to STEP 10 until EOF occurs in file "calibration.cal"
15	close "calibration.cal" and "calibration.out"

Subroutine `void read_native_prefs();`

This routine reads the calibration data. This function is the counterpart of calibration at the beginning of any simulation. It needs to choose some of the calibration data to be applied in the following structure simulation. This decision is made on parameters that one has to provide to run the whole program. These are:

- sequence of the unknown protein

- secondary structure information that should be as precise as possible

A selection on the two most related proteins in the protein structure database (PDB, Francis and Bernstein) is made for the current GA simulation using the following criteria:

- length of the protein
- shared hydrophobic residues within the protein
- shared alpha-helical sequences
- shared beta-sheets

These values of any protein in the calibration database are compared to those of the protein in question. This comparison will lead to two structures which are closest to the unknown protein. Using these two structures the expected values from the fitness functions are calculated by linear interpolation (figure II.3.1.). The final fitness value will then represent the deviation from the expected values and may be given in percent.

It is advisable to re-check the files chosen by the application before letting the application use them. A software solution that is capable of reading multi-chain PDB files is necessary since many files on monomeric proteins contain more than one crystal structure of the same protein. All these files would be dropped by a single-chain selection method.

Subroutine `void objective_function();`

This function calls all fitness criteria functions and returns fitness values to the GA. This function is the master relative to the whole set of fitness functions that are called through the decode_fitness(); function. All fitness subroutines are applied to each new model generated by the decode_conf_solo(); function. The model is temporarily stored in the RAM for that purpose. The values returned by the fitness functions are filtered to avoid unexpected results such as negative fitness values. All fitness functions return a value encoded in percent meaning that a high correlation with threading data corresponds to a high fitness value.

Formula IV.4.1.:

fitness value [%]= function value [absolute] / calibration value [absolute] * 100

Since the objective function needs to summarize all fitness values in one single return value it is necessary to define a distribution pattern or so-called weights for the single fitness criteria. Let's imagine the following setting:

- function for packaging in space, called `tertiary()`, returns 75%
- function for residue interactions, called `interactions()`, returns 67%
- function for beta sheet interactions, called `beta_interactions()`, returns 70%
- function for loops, `called loops()`, returns 60%

These values would make a scientist say: "Well, there are some nice loop data, but the overall packaging has been achieved in a better way." The GA is unable to generate such an interpretation. It needs to get a value like "76" or "88" and would then find that "88" > "76". A conditional clause would define ">" as "better" or "fitter". The next task is to summarize the values to a single one which represents the structure in the best way. The user is asked to assign so called weights to each parameter. These weights act as factors for multiplication with the fitness values. Finally the sum of all weights should be a number like 100 or 1000 indicating a good overall fitness of the structure. Testing has shown that it is most efficient to equally distribute the weights between the fitness criteria although some of them are much easier to achieve than others. We prefer to find a structure that shows high fitness values in all criteria at a time. This is also justified by the fact that we included only few, but equally important criteria to judge protein structure quality. An extremely high weight given to a specific fitness function will lead to structures that fulfill this criterion easily. Still, the resulting structure will be a bad prediction since all other criteria would not have been optimized anymore.

Subroutine `double clash();`

This is the clash fitness criterion. It returns the number of residue overlaps. The clash function searches the structure for overlaps and crossings of the polypeptide chain. Such clashes do not occur in reality and are physically impossible. Models containing such clashes must not be kept as valid and should therefore be eliminated during the simulation. Despite this it is possible that a model with a high overall fitness does contain single clashes, especially in case of long chains. Single clashes should not immediately lead to the destruction of a model. The user may set a number of allowed clashes for any simulation. This number is generally limited to two clashes. With this setting most final models still will not contain any clashes. Since the clash function has some influence on the fitness of the structure it may be considered as a fitness function, too. In contrast to all other fitness functions it does not return a value between 1 and 100. It may be considered as a master function that acts completely independent of all other criteria. It contains the following condition clause:

> IF the number of clashes found EXCEEDS the given MAXIMUM the fitness value of this structure WILL BE SET TO "1".

Fitness values lower than one are not allowed within the genetic algorithm due to possible division by zero. The `clash()` function enables the GA to eliminate the majority of clash containing structures efficiently while on the other side it does not destroy any new "thought" immediately as soon as it contains one or two clashes. In case of structures that are difficult to predict to the GA, the user may decide to allow more than two clashes. Even though this will bear the risk of clashes in the final result the overall fitness of the model may rise to a higher value.

Figures IV.4.4. and IV.4.5. to IV.4.7. show the resulting models and optimization processes in simulations with Crambin (1crn). Crambin contains both helical and stranded elements and thus is suitable for such example calculations. In all cases, i. e. zero, two and ten clashes allowed during the simulation, we are able to achieve models with a tolerable overall fitness and low RMSD (below 6Å). However, there are markable differences in quality depending on the number of clashes allowed. We compare the two samples with zero and ten clashes to our standard setting of two clashes. While ten clashes usually lead to a fast optimization due to a higher conformational flexibility the GA exploits these ten clashes in most cases (7.9 clashes mean, maximum 10, minimum 7). This means they remain part of the final model in the end of each simulation. Such a series of clashes is depicted in figure IV.4.4. '10' (arrow). They are located between the unfolded carboxyl terminus and one strand of the beta sheet. Compared to the two other optimization plots, a simulation with 10 allowed clashes clearly shows the best increment both in fitness quality and speed. Still the overall RMSD similarly low as in the other simulations, but a model that contains such an obvious error is not useful in practice.

Interestingly, the final outcomes with zero and two allowed clashes are very similar. A difference may be seen in optimization plots. Two allowed clashes show a slightly higher increment in optimization speed which makes the evolutional search more efficient. This is important for our goal, namely a fast desktop application. A low number of possible clashes is not exploited by the GA finally. Ten of ten simulations have turned out to contain zero clashes in the final result, even if two clashes were allowed.

We therefore propose the usage of a limited number of clashes adapted to the protein length. Larger molecules will possibly bear more clashes and the user may allow more. In general, it is advisable to start with a low number and increase the clash possibility if the optimization gets stuck quickly. The optimization plots may be a helpful tool for determining when this happens.

Figure IV.4.4.: *Models compared to the original structure using different clash preferences*

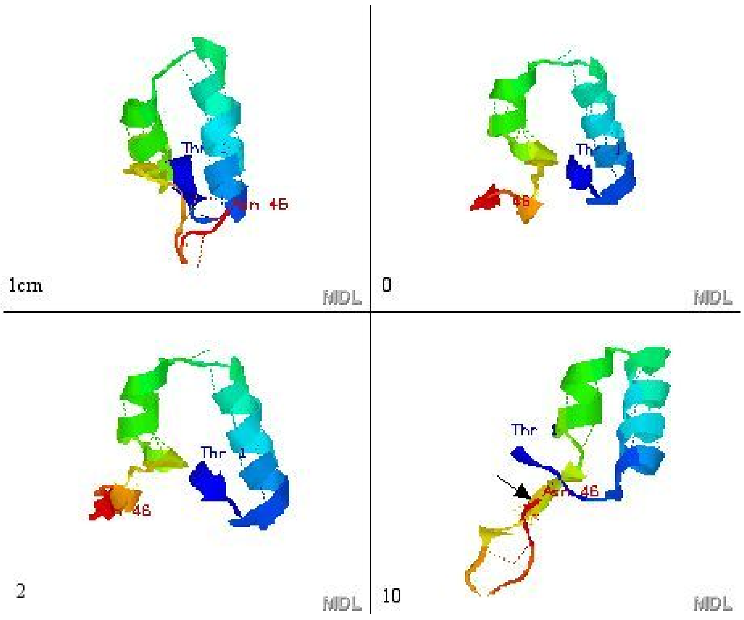
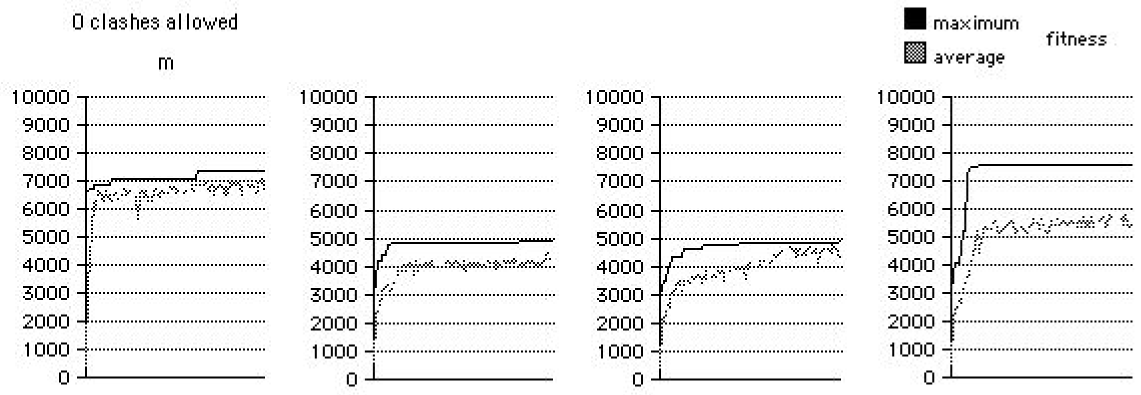
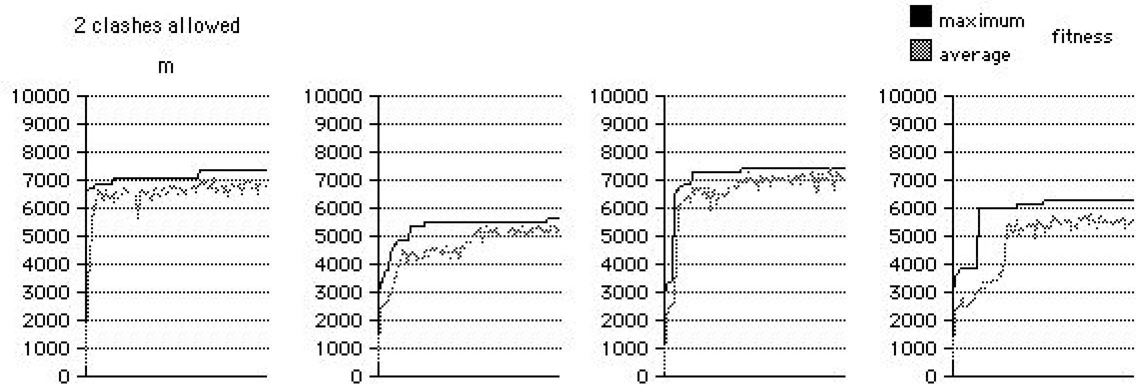
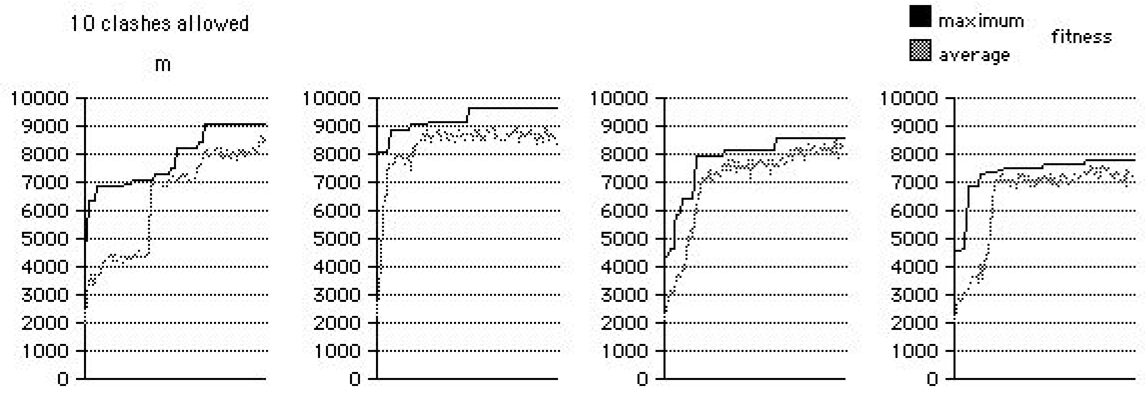


Figure IV.4.5.-IV.4.7.: *optimization plots with different clash preferences.*







Subroutine `float tertiary();`

This fitness function returns the atom scatter around the geometrical center of the molecule as a single value. It needs calibration. The tertiary criterion is the first parameter presented in this work that needs calibration. It calculates the sum of all distances between the center of geometry of the model and all C-alpha atoms in the chain. The resulting value is compared to the expected value from calibration and then given in percents of deviation. A smaller value means a tighter packing of the structure. To avoid errors values below zero are not allowed and are set to zero.

Subroutine `float interactions();`

This fitness function returns the number of spatially neighbored residues. It needs calibration. The routine counts all closely related residues in the chain. The criteria on which the calculation is based are the following:

- Residues may not be direct neighbors in the chain. There must be at least five residues between them.
- The interaction distance must be between zero and one. This distance is calculated by measuring the distance between the two C-beta atoms. Then, the sum of the interaction radii of the residues involved are subtracted. The resulting value is the interaction distance.

The interaction radii for all amino acids are derived from native structures and are approximations only. The important thing about this calculation is that the program always uses the same parameters and therefore they are encoded as constants in the source code. Even if one may disagree with some of these constants it does not really matter to be very precise here as long as the conditions are the same for both calibration and judgment by the GA.

Finally the amount of all interactions found is returned by the function. In case of beta-sheet containing proteins the routine uses the calculation time consuming distance results to detect closely neighbored beta strands as well, and to reward those. For this kind of interactions the distance is allowed to vary within a lager range. Measurements on some native state sheets has shown that two to five Å are a good setting. This range is encoded as a constant, too. Of course, searching for beta sheets in this manner is only applied to predicted strand regions and not to strands which have been generated by the algorithm. The criterion must not be able to transform regions to sheets to avoid unexpected results and of course to emphasize on building up the predicted sheets. This criterion does thus operate only on the beta-strands derived from secondary structure prediction whereas the beta-bonds function (i. e. `count_hbonds`) creates (and shrinks) newly evolved beta-strands as well.

This function is a very important to the overall fitness. To emphasize on the flow of operation a quick flow diagram is shown in figure IV.3.1. (paragraph IV.3.).

The protein database clearly shows that there is a good linear correlation between the function values returned by interactions() and the number of amino acids in the proteins. 398 PDB proteins have been tested on this function. The linear regression coefficient is r=0.975530816. The plot is depicted in figure IV.3.2. (paragraph IV.3.).

Subroutine double `count_hbonds();`

This fitness function scans for hydrogen bonds within predicted beta sheets it works with or without calibration. This routine applies to beta sheet regions. It counts existing hydrogen bonds between O's and N's. Each donor or acceptor atom may only contribute one simple bond to avoid exceeding the actual number of bonds. The distances allowed for a H-bond between an O and N are encoded in constants, two to five Å have been successfully used. One must clearly see that these distances are not always realistic. The advantage of this adjustment is a more efficient search for the correct structure. If the criteria are too difficult to reach, the GA might be able to achieve them with only a low probability. The status of achieving the criterion gradually is impossible with a strict criterion whereas a mild judgment will give more hints to the GA. Spoken in pictures one may imagine a blindfolded person that should find something by being given hints from a person that is able to see everything. The latter person is only allowed to say "closer" or "you're moving away". The radius around the object to find from which the "closer" may be applied corresponds to the maximum distance for an hydrogen bond. If one imagines a larger radius the blind person will reach this range quickly and will then get more hints. As soon as the radius is set smaller the blind person will get into it with a smaller probability and finally will need much more time to find the object.

Another point to deal with in this function is the calibration. Since it searches for H-bonds within sheets, an expected number of bonds must be given to calculate a fitness value in percent from the number of bonds that were found. This expected number mainly depends on the number of amino acids in the sheet regions. The problem of sheets that consist of more than two parallel strands may be solved by a calibration on known beta proteins. Another way is to estimate the number of bonds from the number of amino acids with strand conformation.

This mainly applies to double-stranded sheets. The decision on using either the calibration or the estimation therefore depends on the known structures which have been chosen for calibration. If these resemble the unknown protein in their beta content, the number of expected H-bonds will be derived from the crystal structure. If the deviation is too high the estimation will be based on the beta sheet prediction for the unknown protein. In any case the program will tell the user which method has been applied.

Subroutine double `secondary();`

This fitness function returns the similarity between the given and the evaluated secondary structure element sequence. This is a very simple fitness function. It determines how many percent of the given secondary structure prediction has been fulfilled in the model. To some extend this function becomes obsolete. The user may enable the protection mode to given secondary structure data. This mode does not allow any point mutation within the predefined structure elements. The protection mode should only be run if the user wants the algorithm to fully build in the prediction. In such a case the modeling process will be much shorter since the conformational space is reduced. When the protection mode is turned on, the secondary() function should be turned off. It will then always return 100% and the only thing it does is wasting processor time. Turning the secondary_formation() routine off while having the secondary protection mode on is not advisable at all. The randomly generated elements can not be changed anymore even in case they are completely wrong. The software will come up with a warning if these parameters are set in such a way.

In case of a good secondary structure prediction the following mode will return the best results:

- secondary_formation(): ON
- secondary fitness function: 0% = OFF
- secondary structure protection: ON = no mutations allowed here

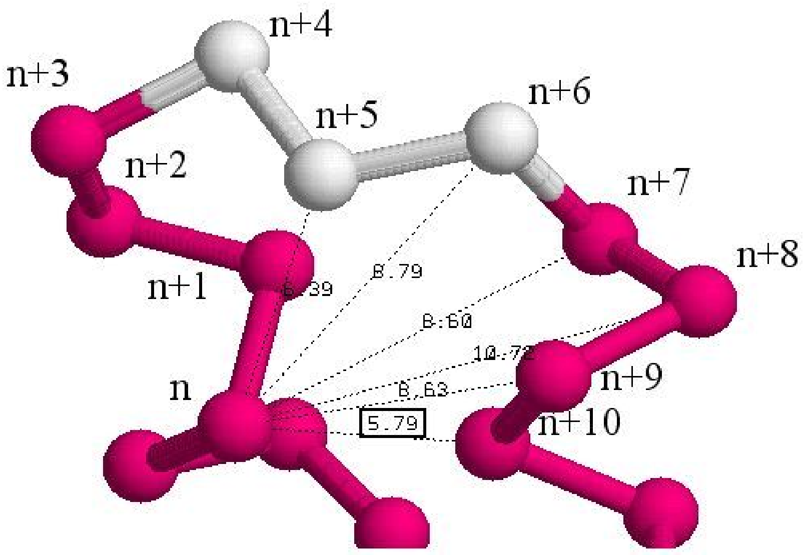
Subroutine `void loops();`

This fitness function scans for loops in regions that were predefined by the user. In case of known loop regions for the protein that is modeled, the user may predefine these loops in the secondary structure sequence. We have empirically found the distance settings from table IV.4.2. that are able to identify correctly folded loops.

Table IV.4.2.: *Loop distance settings*

Parameter	**Value**
minimum number of residues to measure between	5
maximum number of residues to measure between	10
minimum distance [Å]	6.0
maximum distance [Å]	4.7

Figure IV.4.8.: *Length and distance convention for loop detection; helical elements dark and loop region bright*



The routine starts with n for the first residue and scans the entire molecule. As soon as a distance between minimum and maximum is found we count a loop. Figure IV.4.8. shows a successful detection process in the Crambin molecule.
The loop fitness value is the percentage of found loops relative to the number of expected loops though user input.

The structural simulation package has been tested by the means of known protein atomic coordinate sets. Several proteins have been used for testing some of which are of medical importance (see next paragraph). Testing has revealed certain parameters and subroutines as unnecessary for optimal results. Even though their resulting value correlates with correct structural motives and features they do not contribute to optimization. However, these functions are still part of this application and stay turned off with conventional preference data settings. In special cases (e. g. research; structure modeling etc.) these functions may still be useful.

IV. 5. Prediction of Small Protein Folds Including Toxins

Any algorithm needs testing in order to find potential hidden errors and to make statement on its and quality. Many trials with a GA in protein fold prediction were made by our group earlier. This version differs from these prior versions in terms that it combines the GA to a threading procedure. This method does no longer rely on empirical testing, however it is immensely important to provide successful sample runs for potential users. We show here some sample predictions on medically interesting proteins such as small toxins and an Ig binding domain. This is, at the same time, a performance testing of our program, incorporating all the detailed results on the different parameters and subroutines described in the previous section.

Former work (Dandekar and Argos, 1994, 1996) shows that the GA could be successfully applied to the non polynomial problem of fold prediction in small proteins. This algorithm was guided by empirically determined fitness functions. We have modified this part in this thesis and need to know whether the newly implemented threading procedure is able to reproduce the results. In order to test our program we have used small proteins with known structure and give the quality in terms of the root mean square deviation (RMSD) of our models from the original. We show that the results are satisfactory and present at the same time a high gain in speed.

An important application of structure prediction is medical research. A possible question the we would like to address with the current program is the prediction of small toxins some of which are interesting ligands for pharmacological target structures. The next step could be the prediction of other soluble effector molecules such as interleukins. Finally, we have tried to make our GA structure prediction system capable of handling helical TM segments as they occur in GTPase coupled 7-TM receptor proteins. These proteins are of increased size compared to our regular target molecules, however, we need to extensively test our approach to reach further improvement.

Since we would like to help experimental scientists with our program, too, we have integrated the possibility of continuously adding newly acquired data. These data are entered by the user in form of distance constraints. They may aid the fitness functions in detection of atypic and unusual folds which have been elucidated recently. We present tests for these routines as well.

Known protein structures are chosen and certain intramolecular interactions are added to the standard sequence information in from of distance constraints, simulating a potential user who enters his experimental data. Again, we compare the resulting structures to the X-ray diffraction derived model and give the RMSD.

IV. 5 . 1. Test Proteins

With all these criteria we achieved suitable folding results on a number of proteins. To show that our application software masters many different topologies, the following proteins have been used for general performance testing:

Table IV.5.1.: *Test proteins*

Protein Name	**PDB code**	**length (AA)**	**structural features**	**biological function**	**fitness criteria**	**best result from a batch of simulations (10-1000) (RMSD, Å)**
ER-10	1ERP	38	3 helix bundle a-a-a	mating pheromone from *Euplotes Raikovi*	tertiary interactions loops	3.76
Crambin	1CRN	46	2 helix bundle paired beta-sheet b-a-a-b	plant seed protein from *Crambe Abyssinica*	tertiary h-bonds interactions loops	4.77
'protein G' Ig binding domain	2GB1	56	helix with 4 beta strands b-b-a-b-b	immunoglobulin binding protein	tertiary interactions loops	5.98
Scorpion toxin	1PNH	31	helix with paired beta-sheet a-b-b	toxin from *Androctonus mauretanicus* potassium channel (apamin sensitive) blocker	interactions beta interactions loops	4.34
Sea anemone toxin	1ATX	46	atypical sheet structure b-b-b-b	toxin from *Anemonia sulcata*, sodium channel blocker (mammalian heart), neurotoxin	tertiary h-bonds interactions loops	7.58 to 8.43 depending on conformation

The figures below show the simulation results that were summarized in table IV.5.1.. In the online version, you may open the corresponding models for 3D visualization in a new browser window by clicking the links above the figures.

Figure IV.5.1.: *1ER-10 (open in 3D window)*

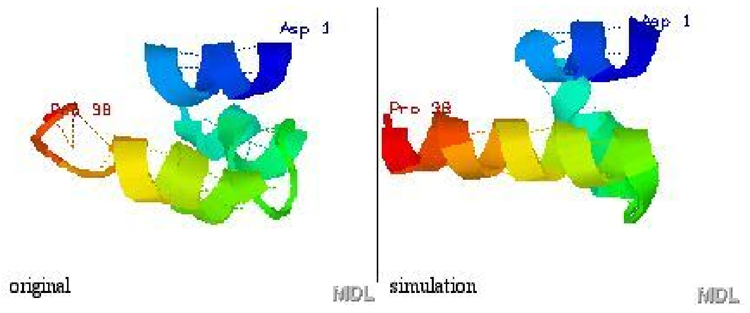


Figure IV.5.2.: *Crambin (open in 3D window)*

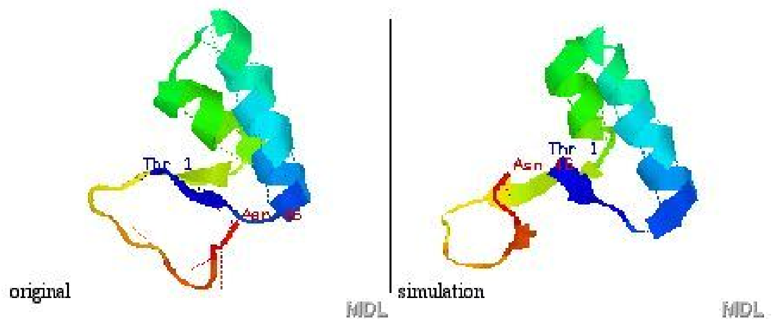


Figure IV.5.3.: *Protein G Ig binding domain (open in 3D window)*

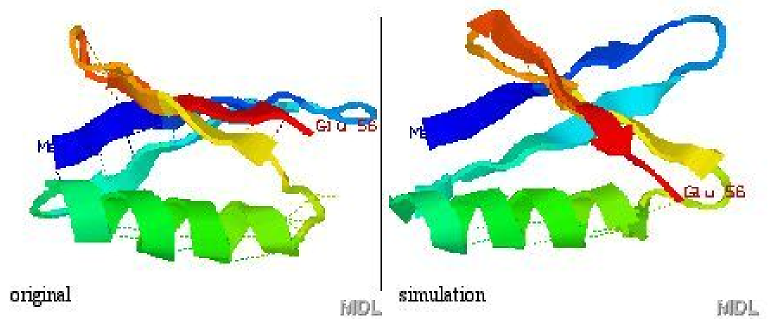
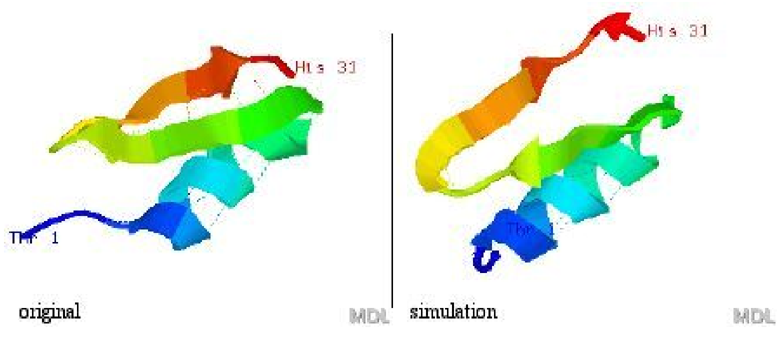


Figure IV.5.4.: *Scorpion toxin (open in 3D window)*



Modeling of dynamic and atypically folded structures such as sea anemone toxin (1ATX) is a difficult task. However, it is possible to predict their flexibility since simulations with the analogous fitness criteria settings will result in a range of related folds. We compare here eight NMR data derived structures from the protein database to our prediction. The overall RMSD is not as low as in other proteins. Still, our resulting models share certain elements with the natural fold. Figure IV.5.5. shows the eight NMR structures and figure IV.5.6. represents two predicted folds, both with different settings. The model to the left was calculated using standard criteria only. The GA was given the positions of the three disulfide bonds within the toxin as distance constraints and then predicted the model on the right side. The RMSD's are given in table IV.5.1.. (The online HTML version includes all models rendered with the chime plugin. Click here to open them in a new window.) Certainly, this is only a first order estimation for such challenging protein structures but the two simulations cover the range of different NMR structures for this protein to some extent.

Figure IV.5.5.: *8 NMR structures with different conformations of sea anemone toxin (1ATX)*

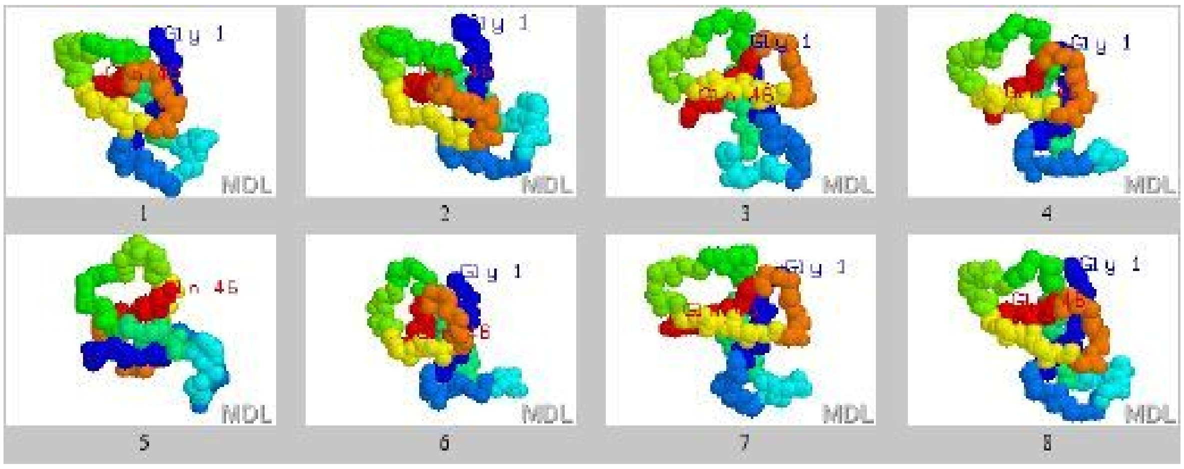
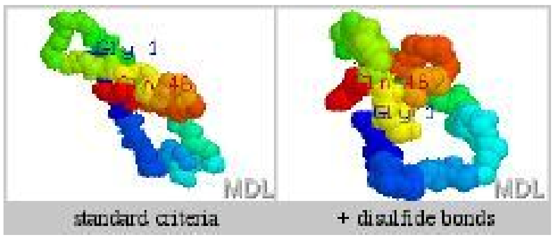


Figure IV.5.6.: *2 predicted folds for sea anemone toxin*



IV. 5. 2. Test Parameters

Generalized fitness function weights have been used building on knowledge of previous efforts (Dandekar and Argos, 1994, 1996; Saxena et al., 2001). However, in this software version we present a new implementation of these functions, mostly in a modified form. Moreover, in order to eliminate empirical weights for fitness function parameters, we have added automatic sequence guided adaptation by database threading. Alignment of helical TM segments was made possible by the creation of a linking loop library which may be generated by the application software directly or which alternatively may be imported. The scoring system for the distinct fitness functions was unified. Most functions finally return relative fitness values in percent according to known 3D structures which are most closest related and whose general parameters match best.

The major fitness functions are

- interactions
- tertiary
- secondary
- loops
- beta interactions, if there are sheets to be predicted
- clashes

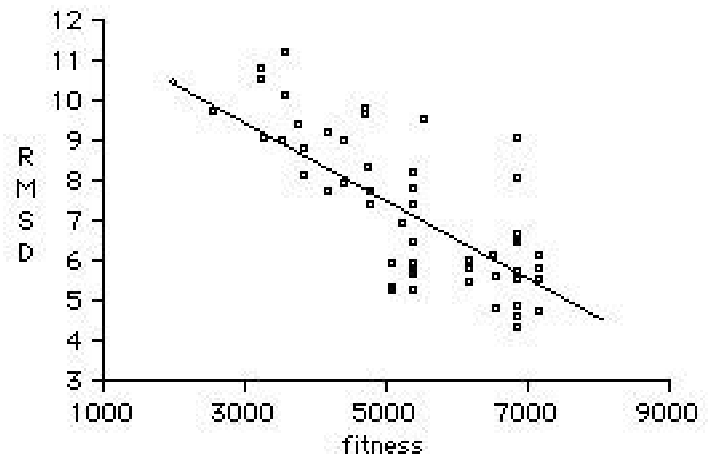
Table IV.5.1. indicates which fitness functions had been used.

IV. 5. 3. Evaluation

It is important to respect the random part within GA simulations. In the example of a simulation for scorpion toxin (1PNH) we show that our fitness parameters mostly are able to distinguish correct from incorrectly folded structures. However the GA will produce some escaping structures. It is the task of the user to carefully interpret the results.

For this combined helix and sheet molecule we have used three fitness functions with equal weights. These were interactions, beta interactions and loops. Figure IV.5.7. shows a diagram representing the fitness values and the RMSD from the experimentally derived structure. We have used the secondary structure data including user defined loops and the amino acid sequence as input.

Figure IV.5.7.: *100 simulations for 1PNH*



The lowest RMSD reached here was 4.34 Å. If we look at the best simulations (equal or more than 75% of the optimal parameter values achieved overall) which include 39 of 100 trials we find that the mean RMSD achieved is 5.93 Å (Fitness values from 6005 to 7163, i. e. 75% of total and above). We see a linear correlation with r = -0.71 between the RMSD and the fitness value, meaning that the better structures have a lower RMSD.

IV. 6. Prediction of Transmembrane Helical Segments

This work presents a method that will enhance the prediction of protein structures with helical transmembrane segments. Such proteins are of major medical interest since a number of ion selective membrane channels and receptors are integrated into the cell membrane. This part of the software will be developed further, and results of the current method are shown here.

The major problem that comes up with the simulation of transmembrane models is the size of such proteins. While smaller soluble proteins consist of only 50 to 100 amino acids, the smaller transmembrane molecules have already more than 250 amino acids. It is possible to partially break down the size in terms of data processing using fixed transmembrane helical elements. Transmembrane proteins are usually stabilized by the lipid bilayer membrane. This fact may explain why interhelical linking loops of these molecules do not match standard conformations. A simple experiment shows that it is impossible to rely on the eight standard conformations inside loop regions: Two 25 amino acid long alpha helical elements that are linked by several standard conformations are aligned in a mostly parallel manner so that they potentially could span a membrane. Such helical elements occur in transmembrane molecules like the photoreceptor bacteriorhodopsin. This molecule belongs to the group of receptors with seven transmembrane segments. The outcome shows the following:

Table IV.6.1.: *Transmembrane loop segments generated from standard conformations*

Number of AA in loop region	Number of possible standard conformations
1	0
2	0
3	0
4	5
5	354

Most known linking loops in transmembrane proteins consist of less than four residues. This fact makes the application of standard conformations unsuitable.

Another approach is the use of the few experimentally determined loops as templates for the unknown molecule. The disadvantages of this method are the deviation from the original concept of ab-initio modeling and the structural incompatibility of standardized helices with natural linking loops. Several adapter conformations would be required or the vast majority of suitable building blocks would need do be discarded.

The current approach with systematically generated short linking loops allows complete modeling of transmembrane proteins that yield a correct membrane topology. The N- an C-terminus are located correctly for the 7-TM protein bacteriorhodopsin.

Table IV.6.2.: *correlation with TM angles and final fitness in five simulations of bacteriorhodopsin*

TM angles function return value [%]	**number of helices oriented correctly [7 total]**	**relative fitness [%]**	**RMSD [Å] (whole molecule)**
83.333	5	89.46	16.138
83.333	5	86.4	21.242
83.333	5	72.8	17.851
100	7	83.89	14.556
100	7	78.57	14.963

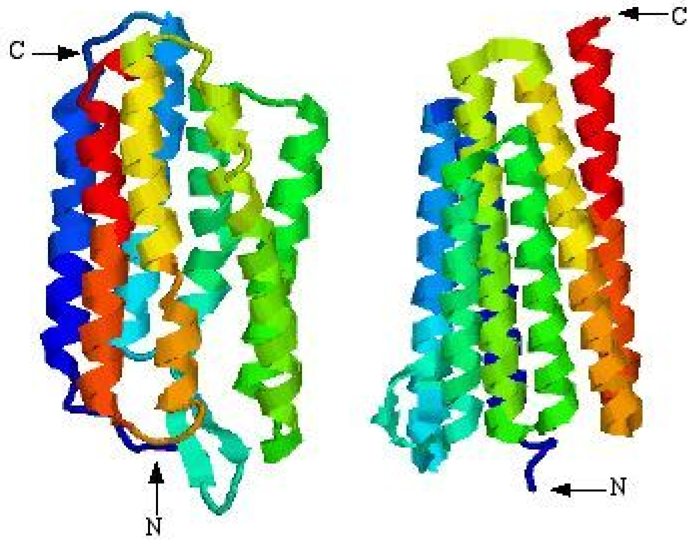
There is no clear correlation between the relative fitness and the actual RMSD of the simulated fold from the original structure. The general fitness parameters do not respect the stabilizing lipid membrane around the molecule as they model a globular protein in water. However, an implemented function function that does so is the transmembrane angle calculation which checks whether the helices are oriented in parallel or not. This parameter correlates with the quality of the folds and may therefore be applied to sort the results as it has been done with the five examples.
Still, the RMSD is far too high for the complete protein (without the application of extra transmembrane linking loops the mean RMSD is 35.98 Å: n = 9, minimum = 28.97 Å, maximum = 43.31 Å). However, the individual TM loop is already well modeled with RMSD's around 5 Å. Table IV.6.3. contains the individual segment-turn-segment RMSD's from the last example in table IV.6.2.. Figure IV.6.1. shows the model with the lowest RMSD in the trial series.

Table IV.6.3.: *individual linking loops and helix RMSD's compared to 1QM8*

Start AA	**End AA**	**motif**	**RMSD [Å]**
9	56	turn	5.525
35	99	turn - sheet - turn - sheet - turn	5.592
80	126	turn	4.815
105	159	turn	2.431
134	191	turn	3.900
167	230	turn	3.385

We conclude that modeling of individual loops is now possible with our program whereas additional fitness parameters are required for correct modeling of the overall structure of transmembrane proteins.

Figure IV.6.1.: *PDB Model of bacteriorhodopsin (1QM8) compared to a prediction*



IV. 7. Application of User Defined Distance Constraints

Known interactions of side chains such as disulfide bonds or ligand binding sites may be entered by the user in form of so-called distance constraints. The user can specify fixed distances between any of the represented side chain atoms. The use of this feature may be useful to integrate experimental results into the protein fold simulations. The same corresponds to known secondary structure elements. Experiments with known structures show that user defined distance constraints improve fitness sorting especially after a batch of simulations has been run. Nevertheless good results can not be achieved by optimizing against these distance constraints alone. Still, the sum of general fitness functions remains the only mean within this application to reach a suitable optimization process. This becomes apparent if the distance constraints were not given a weight during the simulation: Their value clearly correlates with the RMSD of the newly simulated model. With this further option, the user is now able to select the most correct models from a batch of simulations by analysis of the fulfillment of the distance constraint parameter.

V. Discussion

V. 1. General Aspects

The primary motivation for this work was the fact that currently there is no protein structure prediction tool available for desktop and laboratory computers. This application software should be a compromise between high power prediction systems as they are presented at the Critical Assessment of Techniques for Protein Structure Prediction (CASP) (e. g. Bradley et al. 2003, Colubri, 2004, Eyrich et al., 2003, Fang and Shortle, 2003, Skolnick et al, 2003, Xu et al., 2003) conventions (e. g. ROSETTA, TOUCHSTONE, RAPTOR) and the need for fast structural information on proteins in the experimental laboratory. The program should provide the user with a quick glance at the structure of unknown proteins. This, of course, is difficult to achieve since protein folds are complex systems and the computational capacity of desktop machines is still limited. The target user group for this application are researchers in medicine, pharmacology and biology. These professionals may not have experience in bioinformatics and the human interface must be user friendly. Finally, the program should run on several desktop computer operating systems in order not to restrict any user group from the use of the software.

A program that fulfills these criteria must match the following technical features:

- Compact source code and elimination of redundant tasks for fast execution
- Standardized programming language
- Standardized user interface
- Simplicity of operation

As described in Materials and Methods the software was written in C. The corresponding compilers usually generate fast executable binaries. Genetic algorithms in general are tools to break down calculation time to a minimum. Therefore the GA is the core component of this package. A single simulation of a 50 AA long protein usually takes 15-30 seconds using a common desktop computer for the program. Such amounts of times make work flow between the machine and the user reasonably fast.

The user interface is partly based on a command line and partly on a web browser. These two features are available on different computer platforms. The user friendliness may suffer from the textual interface but still it provides a very easy and accurate medium to communicate. It lacks the possibility of visual transmission of results. This work is done by an automated visualization in a web browser. No specific computer experience is needed to use this option due to the standardized PDB molecular data format. Professional users are able to render, manipulate and import their results with other software packages available.

Complex formatting conventions such as user defined distance constraints are simplified with a Java Script Form. This kind of editor will work on any Java capable web browser and dramatically speeds up data input.

Preference files with standard values are generated by the application directly if they are needed. The user may modify them but he is not confronted with the specific formatting conventions. This principle is very common on Macintosh computers and increases the user friendliness.

Structures predicted by this application should be interpreted carefully. This software does not replace human combinatorial thinking. On the contrary the user should always be aware of errors and should critically analyze the results. The application theoretically consists in an ab-initio prediction system which is not very accurate. The amount of additional information supplied by the user largely determines the quality of the results. Therefore he should operate the software as a tool to put bricks of information together to generate three dimensional models of the protein he tries to predict.

The Genetic Algorithm used in this application has several modifications that eliminate a large number of calculations needed for a standardized GA. These features resemble breeding procedures in order to optimize species, i. e. solutions faster. The backcrossing mechanisms increase the risk of trapping in local minima of suboptimal evolution but may be compensated by running the algorithm several times. Batch processing routines provide automation of this feature.

A general and well known problem with the use of GA's consists in the use of random numbers as a motor of evolution. Starting the program with the same random number seed will exactly reproduce all structures obtained. However, for a certain random start population only a part of the solution space is explored. Statistic evidence shows that evolution still may be directed and large numbers of simulations which all start from different random numbers

will converge towards the optimal result. This fact legitimizes the use of a GA.
The second innovation examined in this thesis is based on a threading procedure. Data from experimentally determined protein structures are scanned for generalized and simplified architectures. The threading results are summarized and are compactly stored for calibration of several fitness functions called by the GA. In contrast to homology modeling systems no structural components are copied from the protein database in our application. It is theoretically possible to correctly determine a structural motive that does not occur in the database by this method.

V. 2. Limitations of the Grid Free Building Block Protein Representation

In contrast to a primarily grid based model on a Cartesian coordinate modeling system our system uses pairs of predefined dihedral angles. Amino acids are hereby represented as building blocks and a model will strictly contain these predefined angles. This, of course, leads to a certain inflexibility in contrast to natural protein structures. In case of proteins that mainly show these standardized angles with low deviations our system may predict these structures correctly. As soon as atypical structural elements occur within a protein our algorithm is unable to model these abnormalities. Still, the overall structure may be modeled correctly if it is possible to represent the unusual conformation with an atypical mixture of standard conformations.
In accordance with previous results (Dandekar and Argos, 1994, 1996; Saxena et al., 2001) we find that the advantages in speed and simplicity outweigh the limitations of the standard conformation modeling technique and prefer it for our desktop computer application. Similar observations have also been made for other types of reduced representations of protein molecules (e. g. Skolnick and Kolinski, 2001).
A crucial aspect of search algorithms is the size of the search space. This dimension is the most limiting or extending factor for that kind of prediction algorithms. Here, we try to find possible folds for proteins which are highly complex (Schulz and Schirmer, 1979). The building block mechanism narrows the search space dramatically. It is a conversion from the almost infinite search space of floating point dihedral angles to eight applied conformations. We accept that our application will be unable to do a fine tuning on protein structures and we get a fast software in return. It is always important to make a fair compromise between accuracy and computational efficiency.

Our first experiments with transmembrane models have shown that the building blocks were insufficient in those simulations. However, we would not accept the necessity of stepping into an infinite search space that is based on freely selectable dihedral angles. We have tried to break up the problem to pieces. Individual solutions will be discussed in the corresponding subchapters.

V. 3. Performance of the Genetic Algorithm

In order to judge and improve efficiency of the GA one can distinguish the GA optimization conventions from the optimized weights and parameters of the objective functions. GA conventions include variables such as mutation and recombination rates, population size or types of recombination. The GA can process binary or string data only while most problems are obviously not binary. The objective function must therefore serve as a bridge or as an interface between the non-binary problem and the GA. It is obvious that we can describe the performance in terms of optimization whatever the subject that becomes optimized will be.

As mentioned earlier, many people have used evolutional computation in technical problems. In the 1960s, Schwefel and Rechenberg worked on principles for a machine that would exploit principles of evolution. Similar to them, J. Holland (1975) introduced the so called genetic algorithm. In the beginning, this work was not taken seriously but as soon as it was able to solve technical problems their methods became popular. Nowadays, evolutional calculation has evolved to a robust tool in optimization and a topic for international conferences (Schwefel and Rechenberg, Holland).
A GA needs several environmental parameters to run. The main task in building a Genetic Algorithm is the determination of suitable values for these parameters. We use empirically determined mutation rates, population sizes and numbers of generations. These values at first may not be optimal but as soon as a fast evolutional simulation is achieved by a certain set of conditions, we are able to apply these to different protein folding simulations and protein sequences.
We found that a standard GA like the one described by Goldberg et al. (1989) is already able to serve for protein fold prediction (Dandekar and Argos, 1994). Still, it requires a lot of calculation time due to the possible occurrence of fitness reduction. This loss may spontaneously occur if mutations randomly destroy regions in a solution string that already had been optimized. During the following generations the standard GA would rebuild these regions and the necessary calculations become redundant. In the GA version presented here we introduce a backcrossing procedure to conserve the highest fitness levels during each simulation. This procedure may be compared to breeding procedures. The positive effect of this phenomenon has been used by human beings for centuries or even millenniums, so why should it not turn out to be useful in a synthetic evolution as well? Our experiments show that the backcrossing procedure clearly increases the calculation speed by elimination of

redundant tasks.

A further advantage of our program is that it can teach people in the function and use of the GA. Our application provides output files that contain simulation data from the genetic algorithm that is independent from the structure simulation. Not only do these data help the user to empirically adjust the environmental parameters for the GA, it also allows experiments with a GA as a stand alone teaching instrument for students and schooling.

If we would replace the Genetic Algorithm with a systematic or random search, we would have to process large amounts of data with the objective function. Both would yield much too large sets of calculations for a desktop computer. One argument against a GA is the possibility of exclusion of the good results hidden in some corners of the solution space. This event may potentially happen with the GA in contrast to a total coverage of the entire search space, where it could never occur. However, in general the systematic optimization of the randomized solution trials will prevent this. The user will see these useless events occur during the first generation of new simulations. While some simulations already contain at least one individual with a fitness value above ground level, some populations do not have such a solution trial from the beginning. However, in that case all individuals are given similar chances for propagation. As soon as the mutations and crossing over start, some solutions will be fitter than the baseline. We find that the randomized solution mix from the start may speed up the evolution for some generations if it already contained suitable strings. Moreover, the GA is able to turn nearly any random population into a set of suitable solutions. The only conceivable problem would be a starting population with all strings being equal. Indeed, this is a setting that hardly appears with a computer derived random number set for any population with reasonable size.

V. 4. Threading and GA Combination

The calibration process makes it necessary to provide at large set of known protein structure coordinate sets. In case of the lack of structural information on a special and unusual protein fold the calibration routine may fail. As mentioned above the calibration is not a process that copies distinct structural patterns of known molecules to a database. This opens up the possibility to even detect folds that have not been used during the calibration process if those are suitably covered and judgeable with the generalized fitness criteria.

Even a very small set of calibration proteins may be sufficient to simulate a structure that belongs to the same kind of proteins as the ones used for calibration. However if the the database grows larger the predictions may cover a more extended field of structural patterns. Atomic coordinate files containing errors are usually dropped during the calibration process since they cause errors or unreadable data. Oligomeric proteins that are unstable as monomers should be excluded from the calibration process. In case of inclusion of a large percentage of those structures some of the fitness functions may become miscalibrated. Should there be any doubt whether this happened or not the user may check the files that were used in the current simulation. If an oligomeric structure was included, the user may delete it from the calibration file and re-calibrate the system.

In this work we introduce an efficient combination of two well established and stable methods. This linkage makes our application a fast tool since a GA usually decreases the number of necessary calculations. The threading process is unsuitable in terms of speed since analysis is a time consuming task. We bypass the time consumption during the definitive simulations through a summary file resulting from previous database indexing (all files are summarized regarding their values for individual fitness parameters). Now, the time is spent only once for a lot of simulations. Such a combination has been used by others for different tasks such as in loop optimization and drug design. However, this is to our knowledge the first time that such a combination is exploited for rapid (desktop computer) ab initio predictions for research and teaching.

V. 5. Application for Teaching Purposes

Our new program is also useful for teaching purposes. We avoid to confuse the user with prompts for fitness weight attribution. Since the functions we use are calibrated on the threading summary or otherwise guided by user data such as secondary structure or loop positions, we can achieve useful results with just equally distributed fitness weights. The user may or may not decide which fitness criteria should be used for a specific simulation. In case of no decision the application automatically uses a set of default values. Binary files illustrating the progress of the simulation, population size and generations are generated for each run and may be analyzed if desired. Clickable graphical displays, embedded in HTML format compare simulated structures with the crystal structure (if known) and among each other. All input is either self-explanatory or automatically adjusted using default values.

V. 6. User Defined Distance Constraints

Every protein structure prediction can profit from additional structural information (e. g. Dandekar and Argos, 1997; Sibbald, 1995), and we need a method to enter these data into our program. We have chosen the way of a distance constraint fitness criterion. This subroutine works similar to other distance calculation routines. The user is able to define experimentally derived or otherwise known distances between any atom that is represented by our application.

Our application lacked a graphic user interface for these data and we have decided to implement it for this routine as a javascript form. The text returned by this form may be copied into a text file and should be saved by the user under a given filename in the working directory. The distance constraints entering form checks whether the format for these data are correct. A textual interface is incapable of generating such a file quickly since the protein sequence must not be processed in a linear manner for this purpose.

The resulting fitness value corresponds to the fulfillment of the defined distance constraints. It is given a weight like any other function and may be used to direct the GA evolution. As an alternative we offer to use distance constraints after a number of simulations has been processed.

If the distance constraints are used for optimization there, more than only one distance like a known disulfide bond should be defined. The less distance constraints are defined the more the resulting fitness will alter between high and low values. The absolutely necessary and continous increment of the value is now missing and the function is unable to efficiently guide the evolution. Therefore the user should decide whether the use of only few known distances is possible or if there are more experimental data to add. As soon as we provide more distances to the algorithm, the distance constraints function turns into a solid fitness criterion. The returned value may increase sequentially during the evolution and lead to models that fulfill a large part of the user definitions.

The distance constraint function makes our application a useful tool in experimental protein analysis including prediction of unknown protein structures in collaboration with experimental researchers.. Researchers may continuously interact with the program and add their newest data directly. Our application runs quickly on desktop machines making the workflow even more efficient for such tasks as a design help or as a teaching and simple testing tool.

V. 7. Transmembrane Proteins

We present a method for prediction of helical transmembrane protein structures. The current routines have not achieved overall suitable RMSD values yet. Still, we are able to show here in new work from our group that TM modeling is possible with the GA and a modified building block concept. An extended library of dihedral angles based on a scan through the RAMACHANDRAN map allows the representation of transmembrane hairpin conformations with tight and short linking loops. Data from this library are added to the building blocks and increases the flexibility for the GA optimization in extramembranous regions of the protein. The RMSD values for single two helix segments may be tolerated. The parallel orientation of the helical transmembrane segments is judged by trigonometric angle calculation and returned in percent of 100% optimum to fulfill the fitness criterion as implemented.
By these means we show that the GA is now able to predict a correct topology for a 7-TM protein such as bacteriorhodopsin. There remains considerable work left to do on this promising method, especially in terms of extension of the library with extra conformations. The opposite demands for computation regarding a fast and stable desktop application and a detailed, flexible modeling system are obvious. We will continue the development and will always try to find a compromise between speed and accuracy. Libraries of loops (Forrest et al, 2003) as well as rotamers have been successfully applied to different problems in structure solution, and we are confident that this approach is suitable for our challenge as well. Libraries are a suitable way to encode the required information. In contrast to free float point values a library always features a specific number of loops and hence a specific resolution. Adjustment of this parameter makes it in turn possible to adapt the search space to the computational capacity of the system. This is important for a desktop computer application. If the capacity is not limiting we can either use an increment in the resolution of the library for more accuracy within the representation, or we can spend additional calculations for extra fitness criteria in order to better detect local optima.

V. 8. Conclusion

We now conclude that is possible to achieve protein structure prediction on a regular desktop computer. We are able to demonstrate the potential of our new software in the case of medically interesting proteins such as interleukins, hormones and different toxins. Our application is simple to use and portable to several common computer operating systems. We have been able to reduce the calculation time to an acceptable amount of only several minutes for a batch of predictions for short protein chains (around 50 residues) on a standard processor type. Thus, our software package may serve as a tool in research and teaching.

We will persue our effort on a prediction system for helical transmembrane proteins and on more capacity to model larger proteins, always with the aspect of calculation speed in mind. Protein structure prediction from protein sequences is a critical point for computer aided drug design and medicine, in particular in times when genomic data is growing fast (e. g. sequence databases for many species). The informational values of these data will dramatically increase with first structural knowledge of the resulting proteins that are encoded here.

V. 9. Summary

We present new approaches and programs in our study of protein tertiary structure prediction based on the genetic algorithm (GA), applying also knowledge guided and database guided methods. Common desktop computers are able to run our new software which makes its application possible in the laboratory and in student teaching.

We introduce a new fitness function weight estimate for the genetic algorithm based on similarity of predicted secondary element content to known x-ray crystal structures from the Brookhaven Protein Database. In analogy to existing strategies we call this principle "threading-GA". This allows us to assist previous folding routines with a more natural approach. The evaluation routines are included within the application software so that future crystal structures may be easily included later and directly by the user.

As examples we show the simulation for several proteins. Some of them, like small toxin proteins, are of medical importance and should underline the need of prediction software in this research field. In times where protein sequence information is growing daily, tools are needed to derive structure predictions from it. Our software may be directly used for *ab initio* prediction of protein structure from sequence. In addition, the user may add specific information on the protein in question. This increases the quality of the resulting model.

We show further that a GA may have the potential to serve in prediction of structures for helical transmembrane proteins. Here we combine a systematic plot of the Ramachandran protein conformation map with our existing application. This setting allows us to generate models with correct membrane topology. However, further investigation will be necessary on that challenging topic.

The core GA has been optimized for the specific task of protein structure prediction. The program is now able to simulate small protein structures in less than a minute. Our software is completely written in ANSI C programming language. A system independent interface allows our application to run on the major operating systems.

V. 10. Zusammenfassung

Wir stellen neue Ansätze und Programme im Rahmen unserer Forschung zur Vorhersage von Proteinstrukturen vor. Dabei wenden wir neben dem genetischen Algorithmus (GA) auch Methoden an, die auf vorhandenen Informationen beruhen und durch eine Datenbank unterstützt werden. Herkömmliche Heimcomputer können unsere neue Software ausführen, was ihre Verwendung im Labor und beim Studentenunterricht ermöglicht.

Wir führen eine neue Abschätzung der Gewichtung für Fitnessfunktionen des GA ein, die auf Ähnlichkeiten beruhen zwischen den vorhergesagtem Strukturelementen und durch Röntgenkristallstrukturanalyse bekannten Strukturen aus der Brookhaven-Proteindatenbank. In Analogie zu existierenden Strategien nennen wir dieses Prinzip "threading-GA" (threading = Sequenz-Struktur-Datenbankvergleich). Dies erlaubt eine Unterstützung der früheren Faltungsroutinen mit einem naturnahen Datenbankansatz. Die Auswertungsroutinen sind Teil der Software, so dass zukünftige Kristallstrukturen später durch den Anwender mitverwendet werden können.

Als Beispiele zeigen wir die Simulation einiger Proteine. Einige davon, wie kleine Toxin-Proteine, sind von medizinischer Relevanz. In Zeiten, wo die Sequenzinformationen über Proteine täglich zunehmen, sind Werkzeuge wichtig, die möglichst viele Strukturdaten daraus ableiten. Unsere Software kann direkt zur *ab initio* Vorhersage von Proteinstrukturen aus der Sequenz verwendet werden. Zusätzlich kann der Anwender spezifische Informationen zu der fraglichen Proteinstruktur angeben, was die Qualität des Modells verbessert.

Wir zeigen weiterhin, daß ein GA das Potential aufweisen kann, Strukturen für helikale transmembranäre Proteine vorherzusagen. Dieser Ansatz erlaubt uns die Erzeugung von Modellen mit korrekter Membrantopologie. Dennoch ist auf diesem anspruchsvollen Gebiet der Strukturvorhersage noch weitere Entwicklung erforderlich.

Der GA-Kern wurde für die spezifische Aufgabe der Proteinstrukturvorhersage optimiert und kann kleine Proteinstrukturen in weniger als einer Minute simulieren. Unsere Software ist vollständig in der Programmiersprache ANSI C geschrieben. Mit seiner systemunabhängigen Schnittstelle läuft unser Programm direkt auf den wichtigsten Betriebssystemen.

VI. References

The online version of this document includes direct links to references that have been indexed by PubMed. The links, i. e. [Abstract] behind the PMID's will directly open the corresponding abstract in the web browser.

1. Abagyan R, Totrov M (1994). Biased probability Monte Carlo conformational searches and electrostatic calculations for peptides and proteins. J Mol Biol. Jan 21;235(3):983-1002. PMID: 8289329 [Abstract]
2. Bernstein FC, Koetzle TF, Williams GJ, Meyer EF Jr, Brice MD, Rodgers JR, Kennard O, Shimanouchi T, Tasumi M (1977). The Protein Data Bank: a computer-based archival file for macromolecular structures. J Mol Biol. May 25;112(3):535-42. PMID: 875032 [Abstract]
3. Bradley P, Chivian D, Meiler J, Misura KM, Rohl CA, Schief WR, Wedemeyer WJ, Schueler-Furman O, Murphy P, Schonbrun J, Strauss CE, Baker D (2003). Rosetta predictions in CASP5: successes, failures, and prospects for complete automation. Proteins. 53 Suppl 6:457-68. PMID: 14579334 [Abstract]
4. Cohen FE, Kelly JW (2003). Therapeutic approaches to protein-misfolding diseases. Nature. Dec 18;426(6968):905-9. PMID: 14685252 [Abstract]
5. Colubri A (2004). Prediction of Protein Structure by Simulating Coarse-grained Folding Pathways: A Preliminary Report. J Biomol Struct Dyn. Apr;21(5):625-38 PMID: 14769055 [Abstract]
6. Creighton TE (1993). Proteins: Structures and Molecular Properties. WH Freeman & Co.; 2nd edition ISBN code: 071677030X
7. Dandekar T (1997).Improving Protein Structure Prediction by New Strategies: Experimental Insights and the Genetic Algorithm. J. Mol. Model. 3, 1 -5; available at http://www.embl-heidelberg.de/~dandekar/jmm97018.pdf
8. Dandekar T, Argos P (1992). Potential of genetic algorithms in protein folding and protein engineering simulations. Protein Eng. Oct;5(7):637-45. PMID: 1480618 [Abstract]
9. Dandekar T, Argos P (1994). Folding the main chain of small proteins with the genetic algorithm. J Mol Biol. Feb 25;236(3):844-61. PMID: 8114098 [Abstract]

10. Dandekar T, Argos P (1996). Ab initio tertiary-fold prediction of helical and non-helical protein chains using a genetic algorithm. Int J Biol Macromol. Feb;18(1-2):1-4. PMID: 8852746 [Abstract]
11. Dandekar T, Argos P (1996). Identifying the tertiary fold of small proteins with different topologies from sequence and secondary structure using the genetic algorithm and extended criteria specific for strand regions. J Mol Biol. Mar 1;256(3):645-60. PMID: 8604145 [Abstract]
12. Dandekar T, Argos P (1997). Applying experimental data to protein fold prediction with the genetic algorithm. Protein Eng. Aug;10(8):877-93. PMID: 9415438 [Abstract]
13. Dandekar T, Leippe M (1997). Molecular modeling of amoebapore and NK-lysin: a four-alpha-helix bundle motif of cytolytic peptides from distantly related organisms. Fold Des.; 2(1):47-52.PMID: 9080198 [Abstract]
14. Dobson CM (2003). Protein folding and misfolding. Nature. Dec 18;426(6968):884-90. PMID: 14685248 [Abstract]
15. Dandekar T , Du F (1999). Analyzing the interplay between secondary and tertiary structure predictions in folding simulations with the genetic algorithm. Journal of Molecular Modeling 5, 78-89
16. Du F, Dandekar T (2000). Modelling domains involved in host parasite interactions using standard conformations and the genetic algorithm. Lecture at 14th workshopon Molecular Modelling, TU Darmstadt, May 2000
17. Eyrich VA, Przybylski D, Koh IY, Grana O, Pazos F, Valencia A, Rost B (2003). CAFASP3 in the spotlight of EVA. Proteins. 53 Suppl 6:548-60. PMID: 14579345 [Abstract]
18. Fang Q, Shortle D (2003). Prediction of protein structure by emphasizing local side-chain/backbone interactions in ensembles of turn fragments. Proteins. 53 Suppl 6:486-90. PMID: 14579337 [Abstract]
19. Fogel GB, Corne DW (2003). Computational intelligence in bioinformatics. Biosystems. Nov;72(1-2):1-4. PMID: 14642654 [Abstract]
20. Forrest LR, Woolf TB (2003). Discrimination of native loop conformations in membrane proteins: decoy library design and evaluation of effective energy scoring functions. Proteins. Sep 1;52(4):492-509. PMID: 12910450 [Abstract]
21. Goldberg DA (1989), Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley

22. Goodfellow JM, Moss DS (1992). Computer Modelling of Biomolecular Processes {superposition routine}. Ellis Horwood Limited, 11
23. Jones DT, McGuffin LJ (2003). Assembling novel protein folds from super-secondary structural fragments. Proteins. 53 Suppl 6:480-5. PMID: 14579336 [Abstract]
24. Karplus PA (1996). Experimentally observed conformation-dependent geometry and hidden strain inproteins. Protein Sci. Jul 5(7):1406-20.
25. Kihara D, Lu H, Kolinski A, Skolnick J (2001). TOUCHSTONE: an ab initio protein structure prediction method that uses threading-based tertiary restraints. Proc Natl Acad Sci U S A. Aug 28;98(18):10125-30. Epub 2001 Aug 14. PMID: 11504922 [Abstract]
26. Koonin EV, Wolf YI, Karev GP (2002). The structure of the protein universe and genome evolution. Nature. Nov 14;420(6912):218-23. PMID: 12432406 [Abstract]
27. Li W, Zhang Y, Kihara D, Huang YJ, Zheng D, Montelione GT, Kolinski A, Skolnick J (2003). TOUCHSTONEX: protein structure prediction with sparse NMR data. Proteins. Nov 1;53(2):290-306. PMID: 14517980 [Abstract]
28. Notredame C, Higgins DG (1996). SAGA: sequence alignment by genetic algorithm. Nucleic Acids Res. Apr 15;24(8):1515-24. PMID: 8628686 [Abstract]
29. Ramachandran GN, Chandrasekaran R (1971). Conformational energy map of a dipeptide unit in relation to infrared and nuclear magnetic resonance data., Biopolymers;10(5):935-9.
30. Rooman MJ, Kocher JP, Wodak SJ (1991). Prediction of protein backbone conformation based on seven structure assignments. Influence of local interactions. J Mol Biol. Oct 5;221(3):961-79. PMID: 1942039 [Abstract]
31. Saxena IM, Brown RM Jr, Dandekar T (2001). Structure--function characterization of cellulose synthase: relationship to other glycosyltransferases. Phytochemistry. Aug;57(7):1135-48. PMID: 11430986 [Abstract]
32. Sayle RA, Milner-White EJ (1995). RASMOL: biomolecular graphics for all. Trends Biochem Sci. Sep;20(9):374. PMID: 7482707 [Abstract]
33. Schultz GE, Schimer RH (1979). Principles of Protein Structure. Springer-Verlag, (2nd ed. 1989). ISBN Code: 0-387-90334-8
34. Schwefel HP, Bäck T (1993). Evolutionary Computation. MIT Press Cambridge, MA, USA. Volume 1 , Issue 1, Pages: 1 - 23. ISSN:1063-6560

35. Sibbald PR (1995). Deducing protein structures using logic programming: exploiting minimum data of diverse types. J Theor Biol. Apr 21;173(4):361-75. PMID: 7783449 [Abstract]
36. Skolnick J, Kolinski A, Kihara D, Betancourt M, Rotkiewicz P, Boniecki M (2001). Ab initio protein structure prediction via a combination of threading, lattice folding, clustering, and structure refinement. Proteins. Suppl 5:149-56. PMID: 11835492 [Abstract]
37. Skolnick J, Zhang Y, Arakaki AK, Kolinski A, Boniecki M, Szilagyi A, Kihara D (2003). TOUCHSTONE: a unified approach to protein structure prediction.Proteins. 53 Suppl 6:469-79. PMID: 14579335 [Abstract]
38. Snow CD, Nguyen H, Pande VS, Gruebele M (2002). Absolute comparison of simulated and experimental protein-folding dynamics. Nature. Nov 7;420(6911):102-6. Epub 2002 Oct 20. PMID: 12422224 [Abstract]
39. Wallner B, Fang H, Elofsson A (2003). Automatic consensus-based fold recognition using Pcons, ProQ, and Pmodeller. Proteins. ;53 Suppl 6:534-41. PMID: 14579343 [Abstract]
40. Xu J, Li M (2003). Assessment of RAPTOR's linear programming approach in CAFASP3. Proteins. 53 Suppl 6:579-84. PMID: 14579349 [Abstract]
41. Yang WY, Gruebele M (2003). Folding at the speed limit. Nature. May 8;423(6936):193-7. PMID: 12736690 [Abstract]
42. Zhang Y, Kolinski A, Skolnick J (2003). TOUCHSTONE II: a new approach to ab initio protein structure prediction. Biophys J. Aug;85(2):1145-64. PMID: 12885659 [Abstract]

Appendix

A. 1. Source code written in C

The source code is written in ANSI C. It may be compiled on several computer platforms such as Apple MacOS®, Microsoft Windows® and UNIX systems like Linux or Mac OS X®. The dimension of the source is too large to be printed out here. Please refer to the digital version of this document on CD-ROM or download the source from URL:
http://www.biozentrum.uni-wuerzburg.de/bioinformatik/links/links.htm
Please follow the corresponding link
user: Hench
password: 1CRN.pdb

A. 2. Technical Specifications

This program is completely written in ANSI C and has been successfully compiled on several platforms which are listed in Materials and Methods. The application is a stand-alone executable that runs directly from a command line or equivalent. It uses a textual user interface and several input / output files to communicate with the user. All of these necessary files with exception of user data supplies can be generated by the program itself. They will contain standard values and may be modified by the user with any text editor.

Output files containing cartesian coordinate sets of predicted protein structures are stored in PDB format. These files contain all calculated atomic coordinates and several simulation details in order to locate specific problems that may occur. In addition the coordinate sets are accompanied by automatically generated HTML and script files. These files are linked together and allow the user to visualize both structures and summarized simulation details with a web browser that features the well known Chime Plugin from MDL. This plugin is a scritptable version of the standard PDB viewer from R. Sayle, Rasmol. The direct visualization is available for Apple Mac OS and MS Windows. There is currently no version of the Chime plugin for LINUX or UNIX.

The system independence does not allow any file selection dialogs since they are not part of the standard ANSI C libraries. The alternative is the use of running directories which contain all necessary files for a simulation and which are created by the user. It is possible to automatize this process on some operating platforms such as Mac OS which provides Apple Script. The executable itself only requires a small amount of storage capacity, e. g. approximately 160 kb for the Mac OS version. No specific installation procedure is needed except that a copy of the executable must be located in the running directory.

The use of ANSI C allows porting of the application to nearly any operating platform without any or with only few modifications. The same argument applies to output in PDB and HTML format.

Random access memory is mostly allocated dynamically. This makes the size of the unknown protein the major determinant of the necessary amount of RAM. In most cases five to ten MB are sufficient. Only a few hundred kb needed for a standard simulation.

Additional features of the application include analysis of existing PDB structures, loop database creation and RMSD calculation (Goldfellow and others, 1992).

In order to enter user defined distance constraints correctly the package provides an editor in form of a Java Script form embedded in an HTML file. This editor should work independently of the operating system. The results may by transferred to text files with the system specific text editors by means of the system clipboard.

A. 3. Brief Listing of all Subroutines

The following list contains all of the existing subroutines regardless of their size and their importance. This section is some kind of index where one can look up the relations between the routines. For those readers who do not know the C programming language here is a short summary on how C defines function types:
-*void* is a function that does not return a specific value when it is called. It is able to process global variables and may be primarily used if a complex array of values is being processed at a time. These values are stored in global variables and arrays.
-*double* returns a long decimal number, basically a long signed integer. Like any function it processes global data, too.
-*float* is very similar to double. The difference is, that float returns a value in scientific format and highest precision.

You may open the list of subroutines from the CD-ROM included with this document. It is located in the beginning of the source code. Alternatively, you may download the source from the URL given in A. 1.