Protein Engineering and Pattern Recognition

by

Georg Karl-Heinz Füllen
M.Sc. in Computer Science ("Diplom-Informatiker"),
University of Saarbrücken, Germany
(October 1992)

SUBMITTED TO THE DEPARTMENT OF CHEMISTRY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE IN CHEMISTRY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February, 1994

Signature of Author ______________________________
Department of Chemistry
December 2, 1993

Certified by ______________________________
Douglas C. Youvan,
Thesis Supervisor

Accepted by ______________________________
Glenn A. Berchtold, Chairman,
Departmental Committee on Graduate Students

To all the folks on the Internet.
Especially to Drew van Camp, Tony Plate and Geoffrey Hinton.
They provided great software, and I returned them bug reports :-)
Yeah, I even fixed one bug 8-)

Protein Engineering and Pattern Recognition

by Georg Karl-Heinz Füllen

Submitted to the Department of Chemistry on Dec. 2, 1993 in partial fulfillment of the requirements for the Degree of Master of Science in Chemistry

Abstract

For project no. 1, correlations between protein sequences and phenotypes were explored using combinatorial cassette databases of sequence information and corresponding spectral information derived from the expression of mutagenized pigment-protein complexes for the reaction center (RC) and light harvesting II (LHII) antenna. Heuristically formulated decision algorithms and computer implemented neural networks were compared to determine their accuracy in classification of spectroscopic (i.e., phenotypic) categories. For the databases examined, decision algorithms employing very simple rules were able to separate spectral classes 80-84% of the time, based only on the amino acid sequence of the mutagenized region. Such decision algorithms did not require the formulation of any heuristic rules that involved site-to-site interactions, but rather, performed well based on the stringency of specific critical sites in the protein that accept only a restricted set of amino acids. In some cases, neural networks scored almost 10% higher than these decision algorithms on the same sequence databases. If sites critical for the decision algorithms are omitted, the efficiency of the neural network is still much better than random: 74% of the members of the LHII library and 87% of the RC library are correctly sorted. This shows that there are determining factors in the sequence of these proteins outside the highly stringent sites used by the decision algorithms. Because a linear perceptron scores lower than a more sophisticated three-layer network, some of the factors important for sorting sequences may involve nonlinear (site-to-site) interactions. However, the success of the primitive decision algorithms and perceptrons at sorting sequences into categories suggests that to a first approximation linear effects predominate in the determination of a phenotype.
For project no. 2, Genetic Algorithms (GAs) and an existing protein engineering method called Recursive Ensemble Mutagenesis (REM)

were compared. REM has proven highly effective both in experimental work and computer simulations. REM tries to find a set of "optimal" amino acids at preselected positions in a given protein so that its functional behavior is enhanced or altered appropriately. REM utilizes the natural distinction between genotype (DNA) and phenotype (protein). After heuristical or experimental identification of fit proteins coded by a random initial DNA population, REM calculates a *new* set of fit DNA sequences, exploiting properties of the genetic code. Moreover, as it is too tedious to resynthesize every single DNA sequence having coded a fit protein, generalized schemata (formae) are calculated that serve as a partially wild-card input for the DNA synthesizer producing the new generation. This procedure has an affect similar to uniform crossover, but it also introduces completely new amino acid residues with related fitness. Although developed independently, REM may be seen as a GA constrained by experimental requirements. However, these constraints give rise to a powerful new optimization and diversification component not found in traditional GAs.

Thesis Supervisor: Douglas C. Youvan
Title: Associate Professor of Chemistry

Acknowledgments

You really wanna read an acknowledgment ? So there !
Step 1. Go to the library and collect a sample of at least 25 acknowledgment pages and 25 "near misses" (newspaper articles, ...). Failing the latter, you may also generate your "nulls" completely at random.
Step 2. Train a standard backpropagation neural network to distinguish "positives" (acknowledgments) from "nulls" (other junk).
Step 3. Try to find a stereotypical input which activates the network maximally by displaying the typical features of an acknowledgment page. Collect the text chunks associated with large weights. In this way you have calculated an acknowledgment template, with gaps at the lightly weighted positions of the stereotypical input.
Step 4. Obtain a natural language processing tool from the Internet to parse your acknowledgment page. If the program dies, go back to step 1, or file yet another bug report.
Step 5. Try to locate as many of the following people: *Douglas Youvan, Kai Griebenow, Ellen Goldman, Simon Delagrave, Steve Robles, Mary Yang, Celeste Winant, Bill Coleman, Matt Scholl,* . With their kind assistance, you can execute the following *Recursive Text Filling Mutagenesis* (RTFM, not to be confused with "Read the f***ing manual"). For the zeroeth generation, fill in the gaps randomly, and let the people you located proof-read the full text. Calculate new gap target sets from the few positives you got (if any, that is). Repeat this scheme until you find at least one guy obtaining a high approval rate.
Step 6. ADD THE FOLLOWING SENTENCES:
Special thanks are due to my supervisor Prof. Douglas C. Youvan, a nice guy providing me with a fascinating mind-boggling 1-year roller-coaster trip into a different culture, a different department, a completely new world. His unfortunate leaving MIT stopped my adventure prematurely. Dr Kai Griebenow sat beside me, another German, funny and knowledgeable. Ellen Goldman was my collaborator for Chapter 2, a very friendly one, that is. Simon Delagrave also kindly helped me getting started, and saved me from missing a not-existent deadline. I apologize for any inconvenience if you ended up in Step 5 or if I simply forgot to mention you. Anyway, then you cannot feel characterized inappropriately, like the others.

Step 7. Finally note that you have just generated an acknowledgment page, but there already is one (this page, but OK, it's not a good one). Depending on the weather, this exercise will be either

* regarded as an incredible waste of time and money, or
* as a valuable experience, solid work preparing the way to high benefits in the future. Maybe you will become the inventor of the *"Automatic Acknowledgment Page Generation Tool" (aka CyberAck)*, to the benefit of all graduate students and mankind...

And what the heck, you will not have to write an acknowledgment on your own !

Table of Contents

Chapter 1

Introduction 8

Chapter 2

Estimating Protein Function from Combinatorial Libraries by Decision Algorithms[1] and Neural Networks 14

Chapter 3

Genetic Algorithms and Recursive Ensemble Mutagenesis in Protein Engineering 36

Appendix

How to reproduce results reported in Chapter 1 54

[1] The term "Decision Algorithm" has been coined by Arkin & Youvan . (1992)

Chapter 1

Introduction

Will there ever be an *"Automatic Acknowledgment Page Generation Tool",* like the one mentioned in the acknowledgment ? "Yes, this is quite possible," say the true believers^{TM} in "strong" artificial intelligence (AI). But up to now nobody has come up with a "hopeful monster", an artificial life form performing tasks in a way at least mimicking human originality and creativity. Is the simulation of a human *protein engineer* easier ? Maybe, because it may only require "computational intelligence". (CI, in contrast to AI, is a term coined to distinguish genetic algorithms (GA's), genetic programming, artificial neural networks (ANNs) and the like, from "symbolic" areas of AI, e.g. expert systems. In CI, intelligence stems from the cooperative interaction of units, in a highly parallel manner. Some people call it "connectionism".)

Engineering proteins may be easier than generating acknowledgments or other pieces of literature, because mimicking the "engineer" can be replaced by simulating nature itself. The population of the GA can search the space of possible sequences, like nature has been doing for billions of years. Given good examples, neural networks may guide this search, evolving into protein fitness recognizers, getting closer and closer to nature's own decision process about what works and what doesn't. Alternatively, genetic programming may be used to breed an algorithm for this purpose. Or the population of another GA may work as a classifier by some kind of direct pattern matching with the sequences to be tested, resembling an immune system developing antibodies "against" fit proteins. (Developing the "immune system" itself may be the task of a meta-GA.) Symbolic AI people will try to implement an expert system. For some cases, statistical techniques may be sufficient, as outlined in the next chapter.

In chapter 2, neural networks were applied to estimate functionality of proteins from combinatorial libraries. These libraries were designed with spacers between the mutagenized residues. However, at least for one library, a statistically significant degree of nonlinear behavior was successfully learned by neural networks obtaining functionality prediction accuracies of almost 94%, clearly outperforming the linear (perceptron) model enabling scores at about 88%. A simple decision rule obtained 84%. Since 94% is probably close to the theoretical maximum due to noise in the data, neural networks have fulfilled their promise.

For more general protein functionality prediction problems, and especially if confronted with larger training sets able to exhibit significant second-order features, neural networks can be expected to perform even better. At least theoretically, they are able to approximate almost arbitrary mappings between sequence space and phenotypic estimations.

As shown in the end of Chapter 3, these mappings can also be divided into an information decompressing and an information compressing part. *Decompression* of an amino acid sequence means calculating physical properties, first on a local basis. Full decompression then means calculating the phenotype in its three-dimensional shape and physical manifestation. (Sure, you need to solve the protein folding problem, take care of the cell environment, and more to do this.) Finally, *compression* of the phenotype means the calculation of the (degree of) functionality.

At the heart of Youvan, Arkin, and Yang's "Recursive Ensemble Mutagenesis" (REM, as described in Ch. 3) we will find an exploitation of the fact that a part of the *decompression* of amino acid symbols is homomorphic to the calculation of DNA synthesizer input (so-called "dopes") faithfully expressing them. That is to say that estimating molar volume and hydropathy of a "target set" of amino acids has a rough equivalent in the calculation of the optimal nucleotide mix at the second DNA triplet position. For example, moderate-size hydrophobic amino acids have a T (Thymine) at the second position of their codon, whereas all strongly hydrophilic residues are encoded by a mix of Adenine (A) and Guanine (G) at position 2. If we try to express Methionine (M), Phenylalanine (F), and Valine (V) optimally, we dope with "(A,G,T), T, (G,C)", yielding M, F, V and Leucine (L), Isoleucine (I). Indeed, L and I have hydropathy values similar to the other three. This type of doping has been employed experimentally by Goldman & Youvan for the LH library analyzed in Ch. 2. There, 6 residues were mutagenized simultaneously. The residues had to be selected using expert knowledge, since mutagenizing much more residues in parallel, the observed number of positives becomes zero.

A nucleotide mix calculated from an amino acid target set does not only enable the expression of similar amino acids in parallel, but target sets can be enriched by the unavoidable introduction of amino acids with similar molar volume and/or hydropathy. This similarity makes expression of proteins with related functionality more likely, because disruptive new amino acids are avoided. The scheme

target sets ->
synthesizer dope ->
enriched target sets ->
expression of proteins

is a powerful component of protein engineering by REM, especially if we start with a sequence of phylogenetic target sets, and reiterate. However, the trick has only been applied with respect to hydropathy and molar volume (although it is theoretically possible that the genetic code has a more sophisticated inner structure), and only on a position per position basis, without looking at amino acid interactions. Using the natural genetic code and producing only one DNA synthesizer input one simply cannot take much care of interrelationships.

Getting rid of these restrictions implies moving to a *more* redundant information encoding than DNA. Note that such a new coding can be simulated by *sets of* DNA, and likewise, more complex DNA synthesizer input can be simulated by sets of "dopes". DNA itself is a redundant encoding, 64 different nucleotide triplets encode 20 different amino acids and "stop". We just observed that this makes it easier to have some information (bias) about molar volume and hydropathy in the second nucleotide of a codon. Sets of DNA strands, however, can contain a much more sophisticated bias. Calculating a *single* DNA synthesizer input representing these strands as close as possible would put us back, the sophisticated bias would be lost. But we can calculate sets of dopes. To retain experimental feasibility, we have to keep the number of dopes very small, less than 10 using easily available technology.

In this setting, a more complex "generalized genetic code" maps sets of amino acid sequences into sets of dopes. It can introduce information (bias) with respect to more physical properties, and it can even be faithful to their interactions. For this matter, the generalized genetic code must be sensitive to the context. Therefore, it has to be a mapping from sets of amino acid sequences to sets of DNA synthesizer dopes. However, the number of dopes needed may grow too quickly to be feasible. For instance, we already need two dopes to encode the observation "fit proteins have residue A hydrophilic and residue B large, *or* residue A hydrophobic and residue B small". *How can we provide for complex interactions in a bunch of physical properties, using only a few dopes ?*

Option 1. Let nature reveal the interactions to you, by looking at the feedback it provides in the form of sequenced fit mutants. Starting with a random or phylogenetic "zeroeth generation", one can go through more and more cycles of expression, sequencing, and formulation of one single new dope, which then becomes more and more faithful to at least the most dominating linear interaction. This is the second key idea of REM.

Option 2. Proceed as in option 1, but formulate more than one dope. For that matter, use a Kohonen network, another self-organizing neural network, or a standard clustering algorithm to obtain prototypes from the amino acid sequences. For each of the cluster prototypes, find a dope expressing them as faithfully as possible. Note that for each dope, we are constructing one library.

Option 3. Start with several phylogenetic zeroeth generations, observe throughput (the number of positive mutants), *but do not sequence.* Instead, the fit DNA synthesizer inputs yielding high throughput can be recombined by crossover. Hopefully, these "superdopes" encode for fitter proteins than their "parents", because fit building blocks have been recombined. Delagrave and Youvan's "Exponential Ensemble Mutagenesis" (EEM) uses a similar trick; but in this case, large parts of the sequence are kept constant, while a small part is mutagenized. This is because keeping parts of the gene constant may be the only guarantee for observing any fit mutants at all in the zeroeth generation. Later, results are put together.

Option 4. Let neural networks discover the interactions. You will need some sequenced positives, but after training the network, you can calculate a few stereotypical inputs which activate it maximally by displaying the typical features of a positive mutant. (See also the acknowledgment page...) If the sample is representative, these stereotypes shall contain fit building blocks. Then you do a sensitivity analysis of the stereotypical inputs you found. If an input variable has a high influence on the neural network's prediction about functionality, you have found a presumably important site. Note that for perceptrons, the sensitivity analysis boils down to an inspection of the weight values. The gaps (small sensitivity and/or weight) indicate the sites which are very flexible. Finally, you have to find dopes which express the stereotypical inputs as faithfully as possible, using a random [NN(G,C)] dope at the gap positions.

All methods discussed in this work have one general principle in common: The replacement of an opaque mapping (nature's decision process regarding the functionality of proteins or the fitness of DNA synthesizer dopes) by a computer-implemented one. This replacement is the essence of pattern recognition, as Pao (1989) points out in his book "Adaptive Pattern Recognition and Neural Networks". For protein engineering, this replacement is especially tough, because the natural process of protein functionality determination cannot be formalized in an easy way. However, adaptive computational intelligence systems may be able to come close to nature, avoiding the need for a complete and efficient formalization by approximating it.

Chapter 2

Estimating Protein Function from Combinatorial Libraries by Decision Algorithms[1] and Neural Networks

Collaborator: Ellen R. Goldman, PhD

[1] The term "Decision Algorithm" has been coined by Arkin & Youvan . (1992)

A modified version of the following text has been submitted to the Proceedings of the National Academy of Sciences USA.

Abstract.

Correlations between protein sequences and phenotypes were explored using combinatorial cassette databases of sequence information and the corresponding spectral information derived from the expression of mutagenized pigment-protein complexes for the reaction center (RC) and light harvesting II (LHII) antenna. Heuristically formulated decision algorithms and computer implemented neural networks were compared to determine their accuracy in classification of spectroscopic (i.e., phenotypic) categories. For the databases examined, decision algorithms employing very simple rules were able to separate spectral classes 80-84% of the time, based only on the amino acid sequence of the mutagenized region. Such decision algorithms did not require the formulation of any heuristic rules that involved site-to-site interactions, but rather, performed well based on the stringency of specific critical sites in the protein that accept only a restricted set of amino acids. In some cases, neural networks scored almost 10% higher than these decision algorithms on the same sequence databases. If sites critical for the decision algorithms are omitted, the efficiency of the neural network is still much better than random: 74% of the members of the LHII library and 87% of the RC library are correctly sorted. This shows that there are determining factors in the sequence of these proteins outside the highly stringent sites used by the decision algorithms. Because a linear perceptron scores lower than a more sophisticated three-layer network, some of the factors important for sorting sequences may involve nonlinear (site-to-site) interactions. However, the success of the primitive decision algorithms and perceptrons at sorting sequences into categories suggests that to a first approximation linear effects predominate in the determination of a phenotype.

Parallel construction of large populations of molecules by random mutagenesis or chemical synthesis has become an important approach for biopolymer engineering. The advances in combinatorial biological and chemical techniques make it necessary to develop methods for analyzing large databases of sequence-function correlations. This is particularly crucial when reiterative methods are planned to improve the library.

Combinatorial cassette mutagenesis (CCM) provides molecular biologists with a powerful method of exploring mutations in a protein (Oliphant *et al.*, 1986; Reidhaar-Olson *et al.*, 1991). Entire segments of a gene can be replaced with cassettes of synthetic DNA in which multiple codons have been changed randomly or semi-randomly. Selection or screening criteria are established by the experimenter to classify the CCM mutants as "positives" or "nulls". Positive mutants may be pseudo wild type (phenotypically indistinguishable from wild type) or may express novel phenotypes, while nulls have no functional protein assembly as judged from the selection or screening criterion. Currently, a major challenge lies in interpreting the massive amount of sequence information from a CCM experiment; correlations between the amino acid sequence of a mutant versus its phenotype may not be obvious.

The efficiency of simple decision algorithms (DAs) constructed by the experimenter (as previously described, Arkin & Youvan, 1992; Youvan *et al.*, 1992) and artificial neural networks (ANNs) was compared in the analysis of amino acid sequence data from CCM experiments. The DAs we considered are in the form of simple decision rules based on amino acid positions critical in phenotypic determination. ANNs allow one to model nonlinear, almost arbitrary interactions between the input variables which unlike the DA considers that the determination of protein phenotype is influenced by many possible types of interactions between the amino acids in the chain.

We examined sequence data from a phylogenetically based CCM experiment of the light harvesting II (LHII) antenna (Goldman & Youvan, 1992) and random CCM of the bacterial reaction center (RC) (Robles & Youvan, 1993) from *Rhodobacter capsulatus*. LHII is a peripheral light harvesting protein which binds both a dimer and monomer of bacteriochlorophyll (Bchl) (Zuber, 1986). These pigments can be specifically detected by their near infrared absorption spectra (dimer absorbing at 855 nm; monomer at 800 nm) and provide a colorimetric indicator of protein expression and assembly. Digital imaging spectroscopy (Yang & Youvan, 1988; Arkin *et al.*, 1990; Arkin & Youvan, 1993;

Youvan *et al.*, 1993) was used to screen the LHII CCM library. Two classes of mutants were observed in addition to nulls: pseudo wild type (spectrally similar to wild type), and pseudo light harvesting I (LHI) (spectrally resembling the LHI core antennae). Sixty two of the positives comprising both of these classes were sequenced. The RC is the pigment-protein responsible for charge separation, and without a functional RC, the bacteria cannot grow photosynthetically. Combinatorial RC mutants were selected for photosynthetic growth, and sequences for 25 functional mutants were analyzed.

Materials and Methods

Digital imaging spectroscopy. Digital imaging spectroscopy (DIS) was performed using a *ColonyImager* (Kairos Inc., Mountain View, CA) which facilitates the parallel screening of the ground state visible and NIR spectra from hundreds of colonies directly on a petri dish. Briefly, this new methodology employs a charged coupled device as a detector to image petri plates mounted on the exit port of an integrating sphere. The light source uses an 1/8 meter monochromator to illuminate the dish at different wavelengths and the integrating sphere produces uniform illumination. Spectra are obtained at 5-10 nm resolution; 2 nm band shifts can be detected. For quick analysis of the data, all the spectra from a single petri dish can be presented as a two dimensional color contour map display. Each colony is represented by a horizontal row; absorption is color coded at each wavelength along the row according to a color bar. The spectra can be sorted according to similarity, maximal absorption at various wavelengths, or wavelength of maximum absorption. Different display modes allow the spectra to be scaled relative to the lowest and highest absorption anywhere in the image, or to have each spectra scaled between its own maximum and minimum absorption (as shown in the two panels of Figure 1).

LHII sequence and spectral database. Additional positive mutants were spectrally characterized and sequenced from the previously constructed library described in Goldman & Youvan, 1992. Seven amino acid residues in the β subunit were simultaneously mutagenized using combinatorial cassettes based on phylogenetic target sets. The mutagenized positions were chosen to be on one face of a transmembrane alpha helix, that comprise the binding site for one of the Bchls of the dimer. DIS showed that 6% of the library bound Bchl in two spectroscopic classes: 1) pLH2 mutants have wild type like absorption characteristics with an 855 nm dimer peak and 800 nm monomer peak, 2) pLH1 mutants have the dimer band red shifted to 865 nm and the monomer band reduced or absent. Mutants which showed only absorption characteristic of free pigments in the membrane (760 nm absorption) were classified as nulls. The complete sequence data are shown in Table 1. Each mutant was classified according to its spectral characteristics; Figure 1 shows a color contour map representing the spectra of all the positive mutants that were sequenced. The phenotypes of a few of the mutants appeared to be growth dependent, therefore there is the potential for a 10% experimental misclassification based on the colony's age at the time of imaging. Only unique amino acid sequences were considered for the classification experiments.

Reaction center mutants. We used the sequence information from the nine-site library described by Robles and Youvan, 1993. Nine amino acids in the vicinity of the monomeric Bchl in the active branch of the RC were randomized simultaneously using CCM. These positions were both in the L and M subunits of the RC, and according to the X-ray structure of the *Rhodopseudomonas viridis* RC (Deisenhofer *et al.*, 1985), all are in van der Waals contact with the active branch monomer Bchl. Functional mutants were selected by photosynthetic growth: one in 50,000 colonies plated was found to grow and was therefore considered "positive".

Formulating and evaluating decision algorithms. The simple DAs used in this analysis evaluate each sequence and classify the protein as a positive, or reject it as a null, based on stringent sequence positions. More complex DAs might include sets for protein folding, energy minimization, and overall change in hydropathy and/or molar volume relative to wild type, etc.

For both the LHII and RC sequences it was possible to formulate simple rules for determining positives based on visual inspection of the sequence databases. In the case of LHII, one can generalize that if there is not a Thr in the +7 position, the sequence leads to a null. Additionally, a rule can be formulated to separate the two Bchl binding phenotypes: if the sequence is classified as positive, amino acids with molar volumes larger than Thr (116.1 Å^3) in the -7 position yield a pLH1 phenotype, while amino acids of smaller or equal molar volume yield the pLH2 type spectra. In the case of the RC database, the best criterion for classifying a mutant as positive requires residue L154 to be Leu; otherwise, the mutant is classified as null.

The percent of incorrect phenotypic classification can be evaluated for these simple DAs. The average number of null sequences (see next paragraph) which the DA wrongly labels as positives can be calculated based on the frequency of critical amino acids in the nucleotide mixture used to construct the cassette. The number of nulls considered is set equal to the number of unique positive mutants. The experimentally determined positives that the DA would mistake for nulls are determined by counting from the compiled sequences.

Randomly generated nulls for ANN evaluations. Since only a few nulls were sequenced, we generated sequences randomly obeying the construction scheme of the library in question. The throughput of the library (6% and 0.0002% for the LHII and RC libraries, respectively) gives us an estimate on how many of these "pseudo-nulls" are false negatives. Adding the few true nulls did not change significantly the outcome of our ANN simulations (data not shown). We always averaged at least 16 experiments, each with different random sequences.

Neural network construction and training. We employed standard backpropagation neural networks (Rumelhart *et al.,* 1986; Hertz *et al.,* 1991) consisting of an input layer, a varying number of hidden units, and a binary output layer. In most experiments, the input layer consisted of 12 units for the LHII mutants (2 features for all 6 sequence positions, excluding the His), and 18 units for the RC mutants (9 sites, each with 2 features). We considered two classes of features: (1) molar volume and hydropathy, and (2) artificial letter encoding. For the latter, each one-letter amino acid abbreviation was viewed as a binary number, split into high and low significant bits, and both values were scaled to lie in the interval (0,1). Both physical property values were normalized to lie in the same interval. Regardless of the class of feature selected, the total number of input neurons is not changed, therefore the results of experiments are directly comparable whether using physically meaningful variables or other representations.

The class labels ("0" for nulls, and "1" for positives) were used as target values. To train the network, we used the backpropagation of errors method, employing a sophisticated conjugate gradient descent with line search, implemented as "Rudi Mathon's conjugate gradient with Ray Watrous' line search" by the backpropagation module of Geoffrey Hinton's Xerion Neural Network Simulator. All incoming activations were weighted, summed, and transferred to the units of the next layer using the standard logistic or linear (perceptron) functions. Additionally, we tried the exponential function as a transfer function. Target values and calculated output activations of the network were considered to agree if they deviated by less than 10%. The square of any excess difference was added to the overall error to be minimized.

Evaluation of neural network performance. We were not interested in constructing any specific ANN, but rather in investigating the general usefulness of ANNs for phenotypic estimation of combinatorial libraries. In this case cross-validation is a valid technique for accessing the suitability of various neural network architectures. We did 16 partitions of the data into randomly ordered training and testing sets. For each division of data we trained the network 8 times using different random starting weights and recorded the testing set classification of the best training set classifier (see Fig. 2). Often, *this* classifier has memorized the training set and does not generalize well. However, it would not be fair to record the best testing set classification accuracy. Reporting the average testing set accuracy would have been possible; however, it is common practice to put a neural network in operation which has been trained on all data points, thereby achieving minimum error.

RESULTS

LH versus nulls. In the separation of LH positives from nulls, the DA required positive mutants to have a Thr in the +7 position. Four of the sequenced positives would be wrongly classified according to this rule. Furthermore, 19 nulls would be missclassified as positives (this was calculated based on 57 null sequences, so that there is an equal number of unique positive and null amino acid sequences; the original doping scheme coded for 33% Thr). This leads to an overall rate of 91 correct classifications per 114 mutants (i.e., 80%).

As shown in Fig. 3A, a neural network with one hidden neuron and an exponential transfer function scored better than the DA, at about 86%. Using an exponential transfer function works as a guard against overfitting since the target values are "0" and "1". In this case, large weights memorizing particular aspects of the training set are discouraged because their influence on the activations would be amplified exponentially and the small target values could no longer be met. Indeed, weight values observed in this case were typically smaller than 6. The corresponding "standard" ANN using the logistic transfer function developed weight values in the order of 100, and performed significantly inferior at about 82%. If the number of training cases is small, memorization becomes an important factor, explaining the very bad performance for small training set sizes. The same effect has been observed for networks with more hidden neurons. Using the artificial letter encoding of the amino acids, accuracy is 84% regardless of the transfer function (data not shown). This result indicates that using physically realistic features can aid the learning process, but it can also lead to unwanted memorization.

Fig. 3B gives a more detailed picture of network accuracy for large training set sizes. We observe slightly suboptimal performance at 85% for perceptron-like networks with no hidden neuron and a linear transfer function. This indicates (but does not prove) the presence of nonlinear interrelationships between the amino acids. Furthermore, introducing a cost term associated with each weight enhances performance of the "standard" ANN, but using an exponential transfer function or omitting the hidden neuron seems to be the superior guard against memorization. Altering the cost term may improve performance above the 84% peak we obtained by adding 1% of the sum-of-the-squares of all weights to the network's error, but it is a tedious process. Inferior performance was observed using weight costs other than 1%: 0.01, 0.1, 0.5, 2, 3, 4, and 5%.

Fig. 3C shows the network accuracy if it is not presented the determining +7 position. The 74% accuracy shows that the determination of phenotype is influenced by the remaining positions. Using a direct letter encoding of the 20 amino acids (as discussed above), we observe an accuracy of approximately 73%. Presenting only molar volume or only hydropathy values of the determining +7 position, we observe 79 vs. 85% accuracy, respectively. In these cases, the network's ability to distinguish between Thr and residues with similar physical properties is diminished. Using hydropathy rather than molar volume, Thr is more easily distinguished from other amino acids doped at this position (i.e., Lys, Asp, Arg, Ser).

Three way separation of LH data. Further separation of the positives into pLH1 and pLH2 was based on the rule that if the mutant is positive and has a large (molar volume) amino acid at the -7 position it will be pLH1, otherwise it will be pLH2. Using this simple DA 77% of the nulls, pLH1, and pLH2 mutants are correctly categorized. We observed a very slight advantage for neural networks (80% correct classification), which was again obtained by the network architecture less vulnerable to memorization. The network with an exponential transfer function and no hidden neuron performs significantly better than the network with one hidden neuron (77% correct classification). Using a logistic transfer function, results dropped below 77%.

RC versus nulls. Using a simple DA operating on a data set of combinatorial mutants, separation of functional versus non functional RC mutants was based on the rule that a Leu is required at position L154 for the protein to be positive. This DA yields 6 errors in the 25 functional mutants, and an additional 2 errors out of 25 nulls which are missclassified as positives because of the frequency of Leu in an NN(G/C) dope. This simple DA approach results in 42 out of 50 correct classifications (i.e., 84% accuracy). The low number of positives counted could cause some concern with respect to the square root of N law (i.e., counting errrors for a small sample size), but in our case, the comparisons we have made with ANNs are limited to this specific database. While another sampling of mutants from the LH or RC library will probably lead to values other than 80 and 84%, these values are exact for the database studied herein.

Unlike the LH data, memorization is not an important effect for the RC classification. Panel 4A shows that the network with a logistic transfer function performs better than the one with an exponential function; we obtain up to 91% versus 84% accuracy. There must be significant nonlinear interrelationships between the input features (molar volume and hydropathy of the amino acids), since the perceptron network can only obtain accuracies of up to 88%. No significant change in performance was observed if the number of restarts with new random weights is increased from 8 to 128, or decreased from 8 to 2 (data not shown). This indicates that there are no significant problems with local minima. If the number of training cases is very small (between 14 and 18), chances are high that a linear decision surface exists. Then the perceptron network converges and performs surprisingly well. It is a property of the RC library that with high probability even a few positives contain enough information for a high classification accuracy. Furthermore, the information contributed by separate sites is likely to be redundant in the determination of a phenotype.

ANN accuracy stays slightly above 90% for networks with up to 40 hidden neurons, but it drops to 70-78% using the artificial letter encoding (data not shown). Learning is again diminished by using physically meaningless features. Hinton diagrams of perceptron ANNs reveal that the network is unable to concentrate weights on residue L154, which would enable it to score as good as the simple DA.

Panel 4B shows that after adding a cost term (i.e., 10% of the sum-of-the-squares of all weights) to the error we observe a better prediction of up to 92%. Moreover, networks with 40 hidden neurons obtain almost 94% accuracy if a weight cost of 20% is selected. We could not obtain better results trying a wide variety of combinations between the weight cost term and the number of hidden units.

If the most determining residue (L154) is not known to the network, performance drops slightly to 87%, indicating the presence of strong determining factors in the remaining residues (see Fig. 4C). Dropping other sites, we observe accuracies between 88 and 91%. In particular, leaving out residue L146 does not impair accuracy. At this site we observe a range of amino acids widely scattered in molar volume and hydropathy space, contributing no valuable information to the decision process.

DISCUSSION

There are many types of combinatorial biological and chemical experiments which should be amenable to analysis of sequence-phenotype data by simple DAs and ANNs. Phage display technology can be used to generate libraries of up to 10^9 different proteins (Smith, 1985; Hoogenbaum *et al.*, 1991), that can be screened by affinity for an arbitrary compound. Libraries of synthetic random peptides (Lam *et al.*, 1991; Houghten *et al.*, 1991) have similarly been constructed and screened by binding to acceptors. As combinatorial chemistry becomes more feasible, the databases will become even more complex with a larger repertoire of building blocks which will include thousands of organic chemicals.

We chose to use pigment binding proteins in our experiments because DIS can be used to rapidly assay phenotypes directly from petri dishes. Genomic RC and LHII deletion backgrounds (Youvan *et al.*, 1985) and plasmids that facilitate CCM have been developed for both the LHII (Goldman & Youvan, 1992) and RC (Robles & Youvan, 1993) systems.

The selection of our machine learning method was preceded by a review of comparisons between various symbolic and connectionist paradigms. On the theoretical side, Pao (1989) points out that the essence of pattern recognition is the replacement of an "opaque" mapping from examples to attributes by a similar but "transparent" computer-encoded mapping. In our case, we try to map amino acid sequences to estimates of functionality. Neural networks can learn an exceptionally rich class of mappings (Cybenko, 1989; Hornik, 1989). Exploration of different architectures minimizes our assumptions on the underlying opaque mapping even further.

On the practical side, many publications (for reviews, see Presnell & Cohen, 1993; Hirst & Sternberg, 1992) report encouraging results using ANNs in protein research. Furthermore, an advantage for ANNs was observed in two cases (Shavlik *et al*, 1991): 1) small amounts of training data, and 2) numerical training data. Our experimental data meet both criteria.

Factors affecting decision algorithms. It is possible that DAs could change based on the context of the mutagenesis. A sequence position that is decisive when non continuous amino acid residues are mutagenized might be more flexible when a contiguous stretch is mutagenized. Preliminary results show that the molar volume of the -7 position might not be responsible for pLH1 versus pLH2 phenotype when six amino acids (-10, -9, -8, -7, -6, -5) are randomized simultaneously (S. Delagrave personal communication). However, preliminary sequences in a library where 17 amino acids (-10 through +6) were mutagenized with phylogenetic target sets (Goldman, unpublished results) showed that the phenotypes of the mutants followed the -7 rule.

For both the LH and RC experiments there is some misclassification inherent to the experiment. Up to 10% of the mutants may vary in phenotype depending on their growth time and conditions. The 94% categorization efficiency achieved by the ANN may be close to the highest level that can be expected taking into consideration the inaccuracy of the experimental data. For each library some of the experimentally determined positives will be wrongly classified by the DA. Due to the small sample size, the number of missclassified positives could be slightly different if a second set of positives were isolated.

Decision algorithms and phylogeny. The LH deduced DAs do not recapitulate the phylogenetic data, but rather appear to be specific for *Rb. capsulatus* LHII. The rules fail when applied to the sequences of homologous light harvesting antennae compiled by Zuber, 1990. The +7 sequence position (the determining DA factor between nulls and LH in the library) is a conserved Arg in the β subunits of core antennae (LH I). Among the peripheral antennae (LHII), there is a division among Thr:Asn:Ser of 7:4:4, with a single sequence having a Lys at the +7 position. In the experimental LHII positives the molar volume of the -7 position determines the type of spectral phenotype. However, many of the LHII type antennae from different species (69%) have amino acids with larger molar volume (Val, Leu, Phe) in their -7 position. Although most of the LHI type antennae also have amino acids with large molar volume in their -7 position, there are a few exceptions.

Success of primitive decision algorithms. Originally, simple DAs formulated for computer simulations of recursive ensemble mutagenesis (REM) (Arkin & Youvan, 1992; Youvan *et al.* 1992) were postulated to be too simplistic for actual experimental data. However, LHII antennae and RCs were found to have critical amino acid positions that are basic phenotypic determiners (positive vs. nulls). The success of these simple rules suggests that, as a first approximation, the correlation of sequence and phenotype can be examined on a site-by-site basis. This suggests that CCM can be based on evaluation per site, and justifies the use of phylogenetic target sets in formulating cassettes as well as the construction of target sets per position from positives using random mutagenesis as a basis for REM (Delagrave *et al.* 1993). One should be able to randomize arbitrary regions of these proteins and then combine the sequence information to formulate target sets for larger cassettes (exponential ensemble mutagenesis (EEM) (Delagrave & Youvan, 1993)).

Acknowledgments. Dr. Mary Yang developed the system used in the digital imaging spectroscopy shown in Figure 1. Neural network construction and testing scripts were based on the Xerion Neural Network Simulator developed at the University of Toronto by Drew van Camp, Tony Plate and Geoffrey Hinton. This work was funded by NIH GM42645, DOE 9102-025, DOE DE-FG02ER20019, and by the Human Frontiers Science Program.

References

Arkin, A. P., Youvan, D. C. (1992). *Proc. Natl. Acad. Sci. U.S.A.* **89**:7811-7815.

Arkin, A. P., Youvan, D. C. (1993). In Deisenhofer H. & Norris JR. (eds) *The Photosynthetic Reaction Center, Vol. 1* (pp. 133-155) Academic Press, New York.

Arkin, A., Goldman, E., Robles, S., Coleman, W., Goddard, C., Yang, M., Youvan, D. C. (1990). *Bio/Technology* **8**:746-749.

Cybenko, G. (1989). *Math. Contr. Signals, Syst*. **2,** 303-14.

Deisenhofer, J., Epp, O., Miki, K., Huber, R., & Michel, H. (1985). *Nature* **318**:618-624.

Delagrave, S., Youvan, D. C. (1993) *Bio/Technology.* In press.

Delagrave, S., Goldman, E. R., Youvan, D. C. (1993). *Protein Eng.* **6**:327-331.

Goldman, E. R., Youvan, D. C. (1992). *Bio/Technology* **10**:1557-1561.

Hertz, J., Krogh, A., Palmer, R. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley: Reading, Massachusetts.

Hirst, J.D., Sternberg M.J.E. (1992) *Biochemistry* **31**, 7211-8.

Hoogenboom, H. R., Griffiths, A. D., Johnson, K. S., Chiswell, D. J., Hudson, P., & Winter, G. (1991). *Nucl. Acid. Res.* **19**:4133-4137.

Hornik, K., Stinchcombe, M., White, H. (1989). *Neural Networks* **2,** 359-66.

Houghten, R. A., Pinilla, C., Blondelle, S. E., Appel, J. R., Dooley, C.T. & Cuervo, J. H. (1991). *Nature* **354**:84-86.

Lam, K. S., Salmon, S. E. Hrsh, E. M., Hruby, V. J., Kazmierski, W. M. & Knapp, R. J. (1991). *Nature* **354**:82-84.

Oliphant, A. R., Nussbaum, A. L., Struhl, K. (1986). *Gene* **44**:177-183.

Pao, Y.-H. (1989) *Adaptive Pattern Recognition and Neural Networks,* (Addison-Wesley, Reading, Massachusetts), p.8.

Presnell, S.R., Cohen, F.E. (1993) *Annu. Rev. Biophys. Biomol. Struct.* **22,** 283-98.

Reidhaar-Olson, J. F., Bowie, J. U., Breyer, R. M., Hu, J. C., Knight, K. L., Lim, W. A., Mossing, M. C., Parsell, D. A., Shoemaker, K. R., Sauer, R. T. (1991). *Meth. Enzym.* **208**:564-587.

Robles, S. J., Youvan, D. C. (1993). *J. Mol. Biol.* **232**:242-252.

Rost, B., Sander, C. (1993) *PNAS* **90**, 7558-62.

Rumelhart, D.E., Hinton G.E., Williams R.J. (1986). *Nature* **323,** 533-6.

Shavlik, J.W., Mooney R.J., Towell G.G. (1991) *Machine Learning* **6,** 111-43.

Smith, G. P. 1985 *Science* **228**:1315-1317.

Yang M. M., Youvan, D. C. (1988). *Bio/Technology* **8**:746-749.

Youvan, D. C., Ismail, S., & Bylina, E. J. (1985). *Gene* **38**:19-30.

Youvan, D. C., Arkin, A. P., Yang M. M. (1992). In: R. Maenner, B. Manderick, (ed) Parallel problem solving from Nature, 2 (pp 401-410) Elsevier publishing Co. Amsterdam.

Youvan, D. C., Goldman, E., Delagrave, S., & Yang, M. M. (1993). Meth. Enzym. in press.

Zuber, H. (1986). *TIBS* **11**:414-419.

Zuber, H. (1990). In: Drews G & Dawes EA (Eds.) Molecular biology of membrane-bound complexes in phototropic bacteria. (pp 161-180) Plenum press, New York.

Figure Legends

Table 1 Amino acid sequences of the LHII mutants experimentally classified as positives. The sequence positions are relative to the Bchl-binding His (0). The 'P' mutants show a pLH2 phenotype, while the 'S' mutants are classified as pLH1. The row number indicates the position of the spectra in Figure 1. Although there are some duplications at the amino acid level, each mutant had a unique nucleotide sequence. Rows 27, 28 and 39 are all WT.

Fig. 1 Color contour maps generated by the DIS *ColonyImager* showing the spectra of each sequenced positive. The horizontal axis corresponds to wavelength (710-950 nm) and the vertical axis to colony row number. Each row represents the spectrum of a mutant, encoded by pseudocolor. The color bar shows the range of optical density (OD) from low (black) to high (white). The left panel is in 'absolute mode' (highest OD in the image mapped to white, lowest OD in the image mapped to black); this shows the range of expression levels. The right panel is in 'full deflection mode' (highest OD for each spectrum mapped to white and the lowest OD mapped to black); this representation enables a more direct comparison of the spectral peaks. The row number can be used to find the corresponding amino acid sequence in Table 1. Raw spectral data were sorted according to maximum absorption in the absolute mode, then by wavelength of maximum absorbance in the full deflection mode.

Fig. 2 Exploration and testing of neural networks. We recommend such an extensive evaluation to minimize assumptions on the model, and to maximize confidence in the network's accuracy. In the inner box, entitled: "Test specific neural network", every indentation represents a loop over all possible values or sets as specified one block above, similar to indentation in a C program. The outer box gives an idea of what different neural network architectures shall be explored. The search is by no means complete; one may also change the error measure, the training algorithm, etc.

Fig. 3 Neural network performance for the LH class separation between positives and nulls, as a function of training set size. Panel A shows accuracies of networks with one hidden neuron, and both logistic (filled circles) and

exponential (filled squares) transfer functions. Panel B is a cutout of panel A, adding accuracies of a perceptron network with no hidden neuron and a linear transfer function (filled triangle), and documenting the effect of adding a percentage of the sum-of-the-squares of all weights to the overall network error. These percentages are 0.1% (large circle), 1% (small circle), and 3% (medium-sized circle); these results were obtained using networks with one hidden neuron and a logistic transfer function. Panel C shows the accuracy of networks with one hidden neuron and an exponential transfer function that were confronted with data missing information about the most determining residue used by simple DAs (position +7). Information on this residue is either completely neglected (large box), or molar volume alone (medium-sized box) or hydropathy alone (small box) is considered at this sequence position. All runs involved 16 segregations of the data into training and testing set, each of which includes 16 different sets of randomly generated nulls. All error bars indicate the standard error of the mean for a 95% confidence level. The bold bar indicates the 80% accuracy obtained by the simple DA discussed in the text.

Fig. 4 Neural network performance for the RC class separation between functional and null, as a function of training set size. Panel A shows networks with one hidden neuron, and logistic (filled circles) or exponential (filled squares) transfer function. The filled triangle is used for a perceptron network (no hidden neurons, linear transfer). Panel B is a cutout of panel A, including accuracies of networks (1 hidden neuron, logistic transfer) which add a percentage of the sum-of-the-squares of all weights to their error. This weight cost is 10% (large circle) and 20% (small circle). Additionally, networks with a logistic transfer function and many hidden neurons have been tested, e.g., weight costs of 20% and 20 hidden neurons (small cross) and 40 hidden neurons (large cross) were used. Panel C shows the accuracy of networks with one hidden neuron confronted with data missing about the most determining residue for simple DAs (L154). Performance is plotted for a logistic (filled circle) and a linear transfer (filled triangle). Omitting site L146 instead leads to results plotted by unfilled markers, i.e. circles for a logistic and triangles for a linear transfer, respectively. Some error bars have been omitted for clarity. All runs involved 16 segregations of the data into training and testing set, each for 16 different sets of randomly generated nulls. All error bars indicate the standard error of the mean for a 95% confidence level. The bold bar indicates the 84% accuracy obtained by the simple DA discussed in the text.

	sequence							row
	-7	-4	-3	0	3	4	7	
WT	G	A	L	H	S	A	T	27
S1	L	G	L	H	A	F	T	52
S2	L	G	L	H	A	A	T	62
S3	I	V	T	H	A	A	T	64
S4	V	G	V	H	A	G	T	65
S5	L	A	A	H	A	G	T	61
S6	M	A	L	H	A	A	T	47
S7	L	A	V	H	S	A	T	59
S8	V	A	V	H	S	W	T	53
S9	A	A	L	H	S	T	N	63
S10	L	A	V	H	S	A	T	56
S11	L	V	L	H	S	A	T	48
S12	V	A	L	H	S	G	T	54
S13	V	G	A	H	S	A	T	58
S14	V	A	L	H	A	W	T	51
S15	I	A	T	H	A	T	T	49
S16	M	A	L	H	S	C	T	45
S17	L	A	A	H	A	G	T	55
S18	V	A	A	H	A	Y	T	44
S19	L	V	I	H	A	G	T	57
S20	I	V	T	H	S	A	T	42
S21	V	G	I	H	S	A	T	60
S22	L	A	V	H	A	M	T	46
S23	M	G	V	H	A	M	T	43

	sequence							row
	-7	-4	-3	0	3	4	7	
P1	S	G	I	H	A	G	T	33
P2	C	G	I	H	A	G	T	40
P3	A	G	V	H	S	A	T	36
P4	T	A	I	H	S	M	T	34
P5	A	A	A	H	A	W	T	12
P6	G	A	L	H	A	G	T	3
P7	G	A	T	H	S	F	T	9
P8	G	G	I	H	S	Y	T	35
P9	A	V	L	H	S	A	T	23
P10	G	A	V	H	A	K	T	13
P11	G	G	A	H	A	V	T	15
P12	G	G	A	H	V	A	T	24
P13	G	A	V	H	A	F	S	18
P14	G	A	L	H	S	I	T	32
P15	G	G	I	H	S	Y	T	38
P16	A	A	V	H	S	G	T	30
P17	G	G	L	H	A	V	T	10
P18	A	A	A	H	A	W	T	14
P19	C	G	L	H	A	A	T	50
P20	T	A	A	H	S	A	T	41
P21	G	V	I	H	A	G	T	8
P22	G	G	L	H	A	V	T	22
P23	T	A	V	H	A	Y	T	31
P24	G	A	A	H	S	I	T	2
P25	G	A	A	H	S	A	T	26
P26	A	A	I	H	A	A	T	17
P27	T	A	T	H	A	V	T	37
P28	G	G	I	H	A	V	T	16
P29	A	A	I	H	V	A	S	6
P30	T	A	T	H	V	A	T	5
P31	G	A	A	H	A	K	T	21
P33	F	A	V	H	V	A	T	7
P34	S	G	T	H	A	M	T	1
P35	A	G	A	H	S	A	T	25
P36	G	A	A	H	S	F	S	19
P37	G	G	A	H	A	W	T	11
P38	W	G	V	H	A	F	T	29
P39	G	V	L	H	S	G	T	20
P40	A	V	I	H	A	M	T	4

Table 1

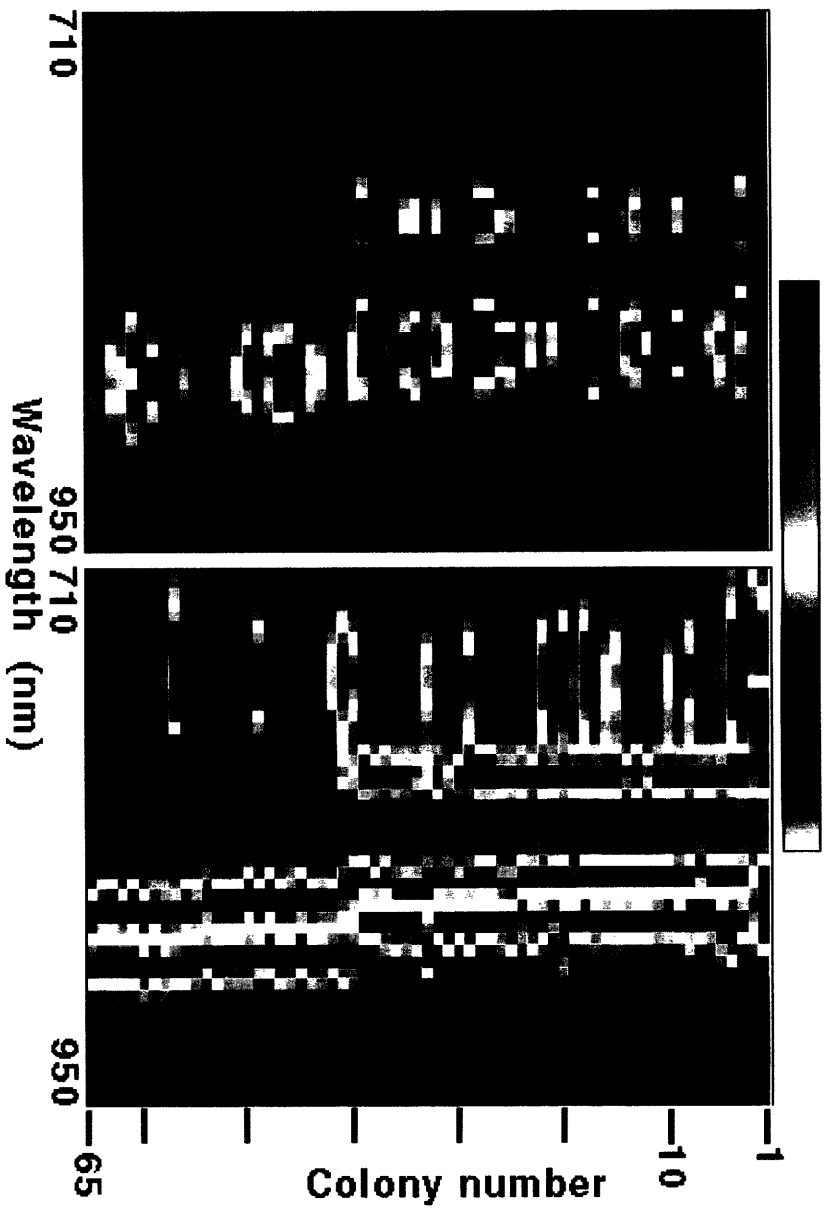

Fig. 1

```
Specify neural network architecture
Specify transfer function, etc.
 Specify number of hidden neurons
   Test specific neural network
   Use various training set sizes
    Randomly segregate data
     Train neural net several times
      Find testing set classification
      of best training set classifier
   Report test results
Report optimal neural net architecture
```

Fig.2

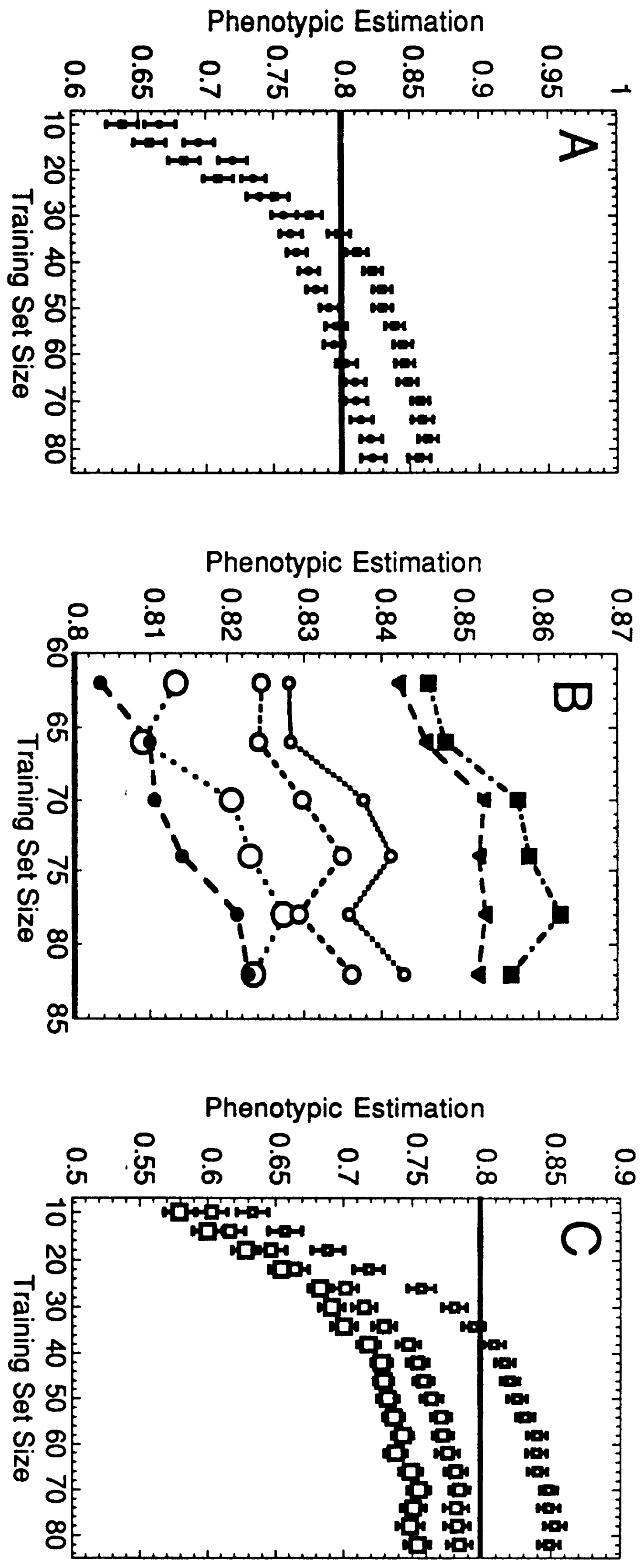
A
Phenotypic Estimation
Training Set Size
B
Phenotypic Estimation
Training Set Size
C
Phenotypic Estimation
Training Set Size

Fig. 3

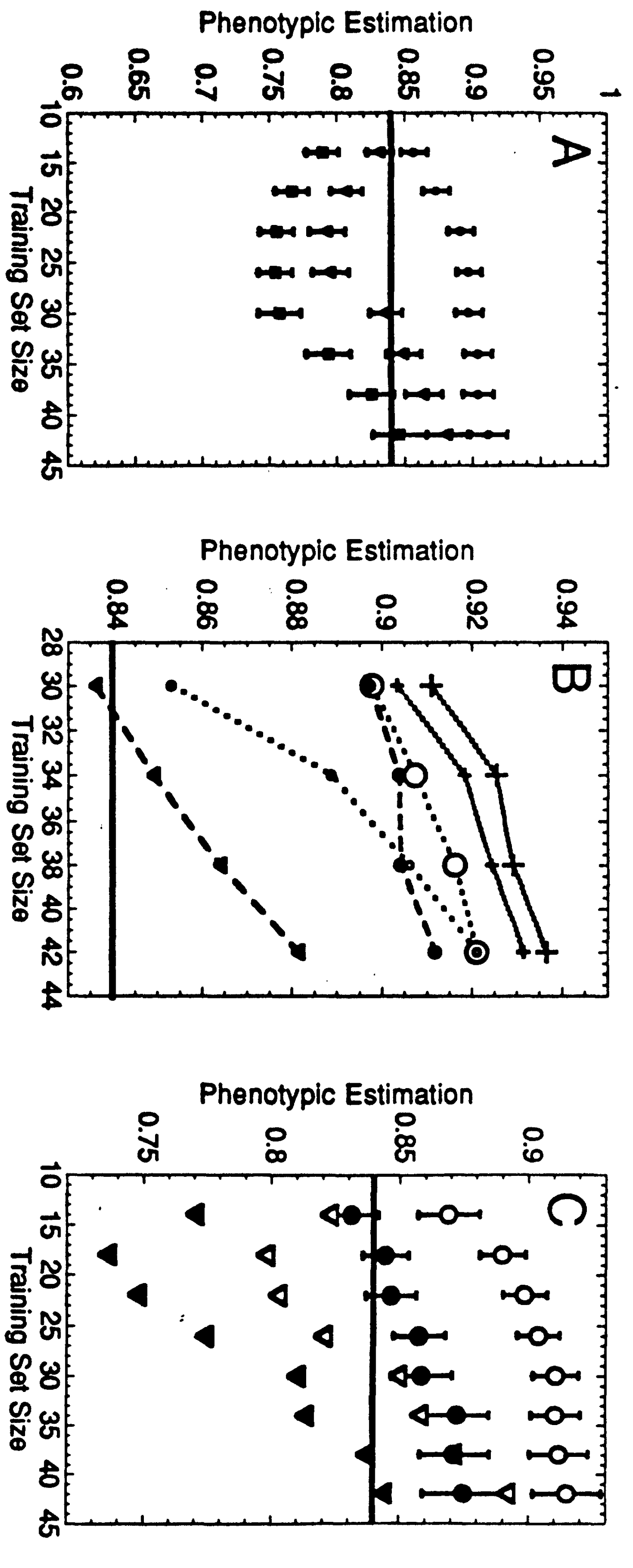
A
Phenotypic Estimation
Training Set Size
B
Phenotypic Estimation
Training Set Size
C
Phenotypic Estimation
Training Set Size

Fig. 4

Chapter 3

Genetic Algorithms and Recursive Ensemble Mutagenesis in Protein Engineering

A hypertext version of the following text shall be submitted to "Complexity International".

Abstract.

We compare Genetic Algorithms (GAs) and an existing protein engineering method called Recursive Ensemble Mutagenesis (REM). This method has proven highly effective both in experimental work and computer simulations. REM tries to find a set of "optimal" amino acids at preselected positions in a given protein so that its functional behavior is enhanced or altered appropriately. REM utilizes the natural distinction between genotype (DNA) and phenotype (protein). After heuristical or experimental identification of fit proteins coded by a random initial DNA population, REM calculates a *new* set of fit DNA sequences, exploiting properties of the genetic code. Moreover, as it is too tedious to resynthesize every single DNA sequence having coded a fit protein, generalized schemata (formae) are calculated that serve as a partially wild-card input for the DNA synthesizer producing the new generation. This procedure has an affect similar to uniform crossover, but it also introduces completely new amino acid residues with related fitness. Although developed independently, REM may be seen as a GA constrained by experimental requirements. However, these constraints give rise to a powerful new optimization and diversification component not found in traditional GAs.

1. Introduction.

One of the most exciting areas in molecular biology is the design of artificial proteins with applications in medicine (Houghton *et al.,* 1991), industrial catalysis (Radledge & Lewis, 1991), and nearly all fields of the life sciences. Originally, computer simulations of REM (Arkin & Youvan, 1992, Youvan *et al.,* 1992) were meant for the design and prediction of laboratory experiments on proteins of the photosynthetic apparatus (Delagrave *et al.,* 1993; Goldman & Youvan, 1992). Here, we have a fast way to detect functional proteins. It is relatively easy (although not trivial) to examine the spectra of photosynthetic bacteria by Digital Imaging Spectroscopy (Yang & Youvan, 1988; Arkin & Youvan, 1993; Youvan *et al.*, 1993). In this way, we can detect whether the artificially mutagenized photosynthetic proteins can bind a bacteriochlorophyll or not. Such new but still functional proteins are studied intensively in photosynthesis research.

In the next section, we will describe the experimental protocol of REM. The computer simulation of these laboratory experiments is difficult because it requires at least a solution to the protein folding problem to evaluate a new protein's fitness exactly. Since protein folding is presumably NP-hard (Unger & Moult, 1993), heuristic decision algorithms were formulated instead. Using them, REM was simulated as described in section 3. A comparison between REM and GAs is given in section 4. REM's utilization of the genetic code in the optimizing calculation of new genotypes (DNA sequences) from the fit phenotypes is a feature which generalizes GAs to a concept called "generalized REM" (GREM) algorithms. A first discussion of this generally applicable idea is given in section 5.

2. The experimental protocol of REM.

In our laboratory, the following procedure is performed to engineer functional proteins performing tasks related to an original protein. Given the original sequenced wildtype, the experimenter chooses positions in its amino acid sequence which should be relevant for the protein's performance. These amino acids can be changed by a method called "combinatorial cassette mutagenesis" (Oliphant *et al.*, 1986; Reidhaar-Olson *et al.*, 1991). Briefly, this method consists of using a DNA synthesizer to produce a gene fragment which then replaces those parts of the original gene that specify the selected amino acids. Synthesizing random DNA leads to an approximately random change at the selected amino acid positions. More exactly, the genetic code's mapping of DNA nucleotide triplets to amino acids determines the amount of each amino acid encoded by random DNA. For example, since Alanine is encoded by 4 of the 64 nucleotide triplets (GCG, GCA, GCC and GCT), a DNA synthesizer producing sequences containing the nucleotides A, C, G, and T with 25% probability at each of the 3 positions gives us about 6% (1/16) Alanine.

Leaving out A and T at every third position, one still encodes every possible amino acid, but they are distributed more evenly. We will denote the first kind of DNA synthesizer input by "NNN", the second one by "NN(G,C)". Here, "N" is a wild-card symbol for equiprobable A, C, G, T and (G,C) is the wild-card symbol for 50% G and 50% C. We have adopted the term "doping scheme" to designate a synthesizer input. Note the similarity of this concept to "schemata" in GA theory. In fact, the "doping schemes" specify "formae", that are generalized schemata (Radcliffe 1991). For example, some commercially available DNA synthesizers are able to produce DNA according to the following doping schemes :

Nucleotide Mixture	Symbol
25% A, 25% G, 25% C, 25% T	N
33% A, 33% G, 33% C	(A,G,C)
33% A, 33% G, 33% T	(A,G,T)
33% A, 33% C, 33% T	(A,C,T)
33% G, 33% C, 33% T	(G,C,T)
50% A, 50% G	(A,G)
50% A, 50% C	(A,C)
50% A, 50% T	(A,T)
50% G, 50% C	(G,C)
50% G, 50% T	(G,T)
50% C, 50% T	(C,T)
100% A	A
100% G	G
100% C	C
100% T	T

Fig.2. DNA Synthesizer Doping Schemes

The REM protocol (Fig. 3) starts with synthesizing random [NN(G,C)] DNA, transforming bacteria to produce the new proteins having approximately random amino acids at the chosen positions and then searching for functional proteins, for example by Digital Imaging Spectroscopy (see section 1). The DNA sequences leading to the functional proteins are determined. To avoid duplicates, REM discards those DNA sequences that encode a protein already found to be functional.

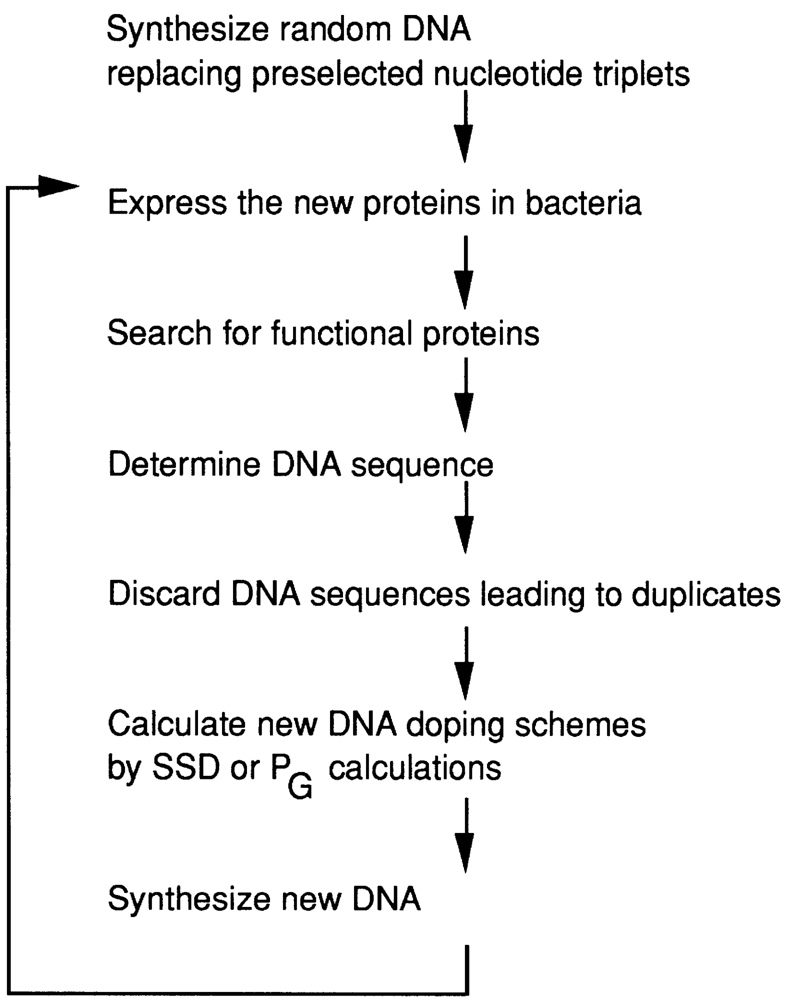
Synthesize random DNA
replacing preselected nucleotide triplets
Express the new proteins in bacteria
Search for functional proteins
Determine DNA sequence
Discard DNA sequences leading to duplicates
Calculate new DNA doping schemes
by SSD or P_G calculations
Synthesize new DNA

Figure 3. The experimental protocol of REM (adapted from Delagrave *et al.*, 1993).

Now, the GA researcher expects that the DNA sequences encoding functional proteins are recombined by cross-over and used to form the new generation. However, in the laboratory this is at least a very tedious if not impossible task. Therefore, we calculate a few or only one new DNA doping scheme as an input for the DNA synthesizer, producing the new generation of DNA sequences in parallel. The new doping scheme is calculated as follows. For each of the chosen amino acid positions a "target set" is formulated as the multiset of all amino acids observed at this position. Which of the admissible DNA doping schemes can generate the target set as closely as possible? Our DNA synthesizer accepts the nucleotide mixtures of Fig. 2 as input. Three of them form a nucleotide triplet which in turn determines a multiset of amino acids, the generated set. The generated set should be "close" to the target set. We are using two notions of "closeness":

(1) The target set is "liberally" closest to the generated set if and only if the sum of squares of the differences

$$\mathrm{SSD} = \sum_{\mathrm{i}} (p_A(\mathrm{i}) - p_T(\mathrm{i}))^2$$

is minimal. Here, i is an index running from 1 to 20, representing the i th amino acid, P_T (i) is its frequency in the target set T, and p_A (i) is the probability of i if doping scheme A is used.

(2) The target set is "conservatively" closest to the generated set, if and only if the group probability

$$P_G = \prod_{\mathrm{i}} p_A(\mathrm{i})$$

is maximal. Here, i takes only the index values of the amino acids in the target set. In this case, all amino acids in the target set must be present in the generated set since P_G is zero otherwise. In the first case, however, amino acids may be discarded. In both cases, new amino acids may be introduced which are unavoidably encoded by the doping scheme designed to encode the target set. These newly introduced amino acids can play an important role, see the next sections.

Having calculated optimal DNA doping schemes encoding the target sets (one per position) as closely as possible, REM uses them as an input for the DNA synthesizer and generates the new population in parallel, starting the next cycle. Using REM, an up to 10^7-fold increase in the observed frequency of functional proteins, compared to random mutagenesis, has been found experimentally

(Goldman & Youvan, 1992, Delagrave & Youvan, 1993). In the latter case, 16 amino acid positions were mutagenized.

3. Computational simulations of REM.

Since some results have been reported elsewhere, we will only give a brief summary. As mentioned in the introduction, the difficulty of the protein folding problem makes an exact simulation of the experiments impossible. However, computer simulations using different heuristic decision algorithms as fitness functions can reveal important aspects of REM (choice of parameters, expected results, etc.). Figure 4, adapted from Youvan *et al.* (1992), shows the efficiency of REM in generating protein populations of a desired functionality. The decision algorithm used evaluates the probability that the mutagenized protein contains a binding site for chlorophyll (see section 1). The population size is set to 10,000, the maximal cardinality of functional proteins used to determine the DNA doping scheme for the next iteration is 50. The plots are average results of REM starting with 10 different initial populations. Plot B shows results using P_G, whereas plot C shows results using SSD. These may be compared with plot A, where a randomly generated population is used for each iteration. Note that in this application of REM our goal is not a single optimum but a diverse population of functional proteins.

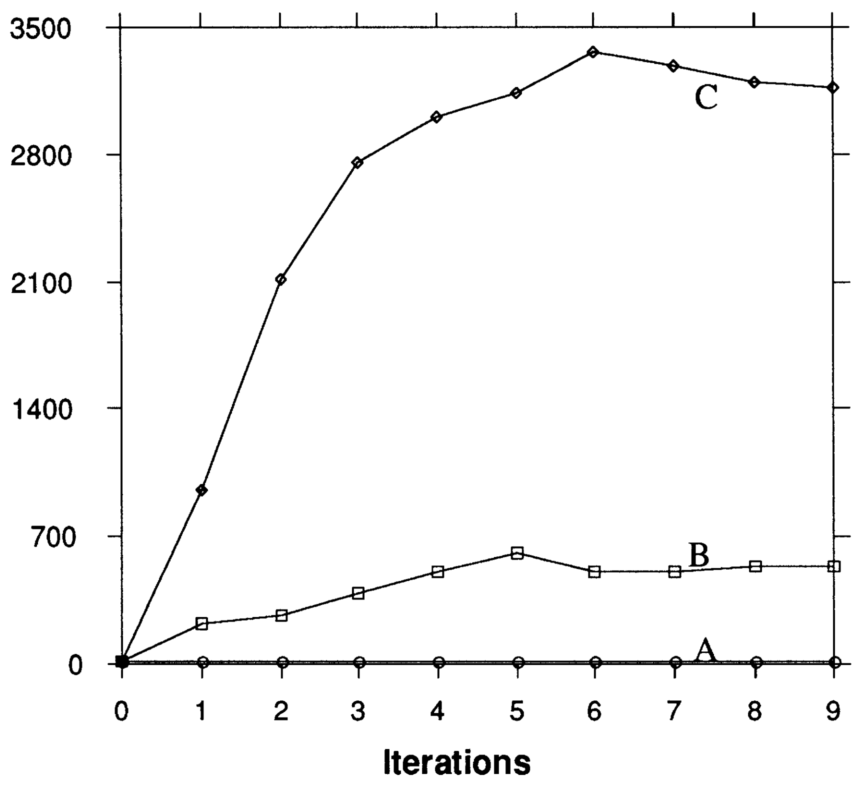

Figure 4. Computer Simulations of REM. (From Youvan *et al.* (1992), adapted)

4. Comparisons between GAs and REM.

Experimental constraints are the main reason for the differences between current REM and canonical GAs (Goldberg 1989). These are:

1. In the currently implemented version, the fitness function evaluates to either 1 (retain) or 0 (discard), just as the protein engineer distinguishes between functional and non-functional proteins. This constraint, however, can be relaxed immediately for those applications of REM which allow a continous measurement of fitness. The probability of the amino acids in the target set used by the SSD or P_G calculation will then be multiplied with the fitness value. Both calculations will return a doping scheme more frequently expressing the amino acids belonging to the fit proteins.

Note, however, that for P_G the effect of the weighted probabilities is multiplicative rather than additive.

2. The recombination procedure of REM is similar to uniform crossover (Syswerda 1991) if one views the DNA synthesizer doping schemes (see section 3) as representations of the populations of DNA they are describing. Every doping scheme containing wildcard symbols can be identified with the set of all DNA strands it represents. Then, ignoring the effects of the SSD or P_G calculations, the fit nucleotide triplets of the old generation of DNA are faithfully represented by the doping scheme, from which the new generation of DNA is sampled randomly. Indeed, REM can be formulated this way, see Arkin & Youvan (1992). In GA terminology, we are calculating "formae", generalized schemata (Radcliffe 1991) in an intermediate step, and we use the formae to construct the new generation. Traditional n-point-crossover between DNA strands cannot be employed experimentally, because it would be quite impractical to implement.

3. A distinction between genotype and phenotype appears occasionally in the GA literature (e.g. Gerrits & Hogeweg 1990, Polani & Uthmann 1992). REM, however, utilizes the natural distinction between DNA and encoded proteins. The genotype-phenotype mapping is given by the genetic code, as discussed in section 2. The genetic code is degenerate, mapping 64 possible nucleotide triplets to 20 possible amino acids (and "stop"). This degeneracy, or redundancy, makes it easy to encode information about the physical properties of amino acids in the second of the three nucleotides, as shown in Figure 5 [adapted from Yang *et al.* 1990]. This structural property of the genetic code is exploited by the SSD or P_G dependent calculation of the next DNA doping scheme from the target set. As mentioned in section 2, some amino acids not included in the target set are "accidentally" encoded by the calculated doping scheme. But there is an enhanced probability that these newly introduced amino acids are similar to the ones in the target set, at least as far as hydropathy and molar volume are

concerned. For example, if we try to express Methionine (M), Phenylalanine (F), and Valine (V) optimally, the amino acid set which is "liberally closest" (i.e., has a minimal SSD value) with respect to this target set is encoded by the dope "(A,G,T), T, (G,C)", yielding M, F, V and Leucine (L), Isoleucine (I). Indeed, L and I have hydropathy values similar to the other three. Since hydropathy and molar volume seem to be the most important parameters for protein folding and function (Robles & Youvan, 1993), the mutagenized protein has a higher probability of being functional; any large change of these values would lead to a high risk of obtaining a non-functional protein. This, together with the general feedback mechanism provided by selection explains the advantages of REM reported in the end of both sections 2 and 3.

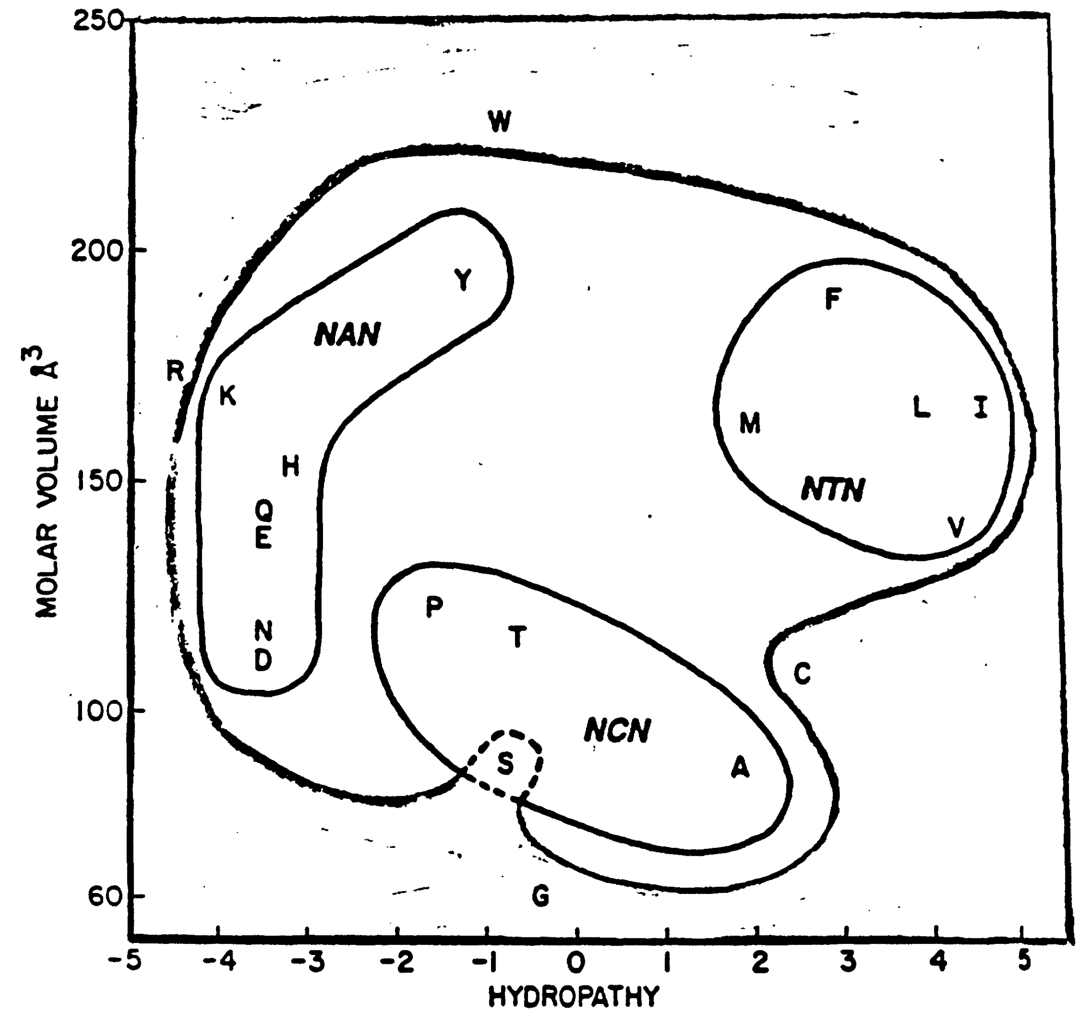

Fig. 5. Hydropathy versus molar volume of amino acids (single-letter code), grouped with respect to the nucleotide at the second codon position. The shadowed region contains the group with Guanine at the second position (termed "NGN"). Here, most of the amino acids have extreme values or special properties (W, R, G, S, C). Serine belongs to 2 groups (NGN and NCN). (Adapted from Yang *et al.* 1990.)

5. Generalized REM algorithms.

In this section, the observations made at the end of the last section are used to motivate an algorithm optimizing not only by selection and recombination or mutation, but also by changing populations while mapping between different representations of increasing redundancy. The action of a GA can be seen as a flipflop process between a representational space and a property space, where recombination and/or mutation occurs in the former and fitness evaluation takes place in the latter. (Therefore, there are two changes of the population in standard GAs: The selection of the fittest, and the recombination. In this section we will investigate the third population change arising from REM.)

At least for engineering problems, the property space can be huge: it may be spanned by all sorts of dependent variables ("features") that can be calculated from the proposed solutions. In the protein engineering case, the proposed amino acid sequence is the starting point for calculating physical properties, for instance, tabulating average hydropathy and molar volume values of each amino acid position. Later, one tries to calculate physical properties for the whole protein and a main goal is the determination of the three-dimensional structure of the protein. These problems are largely unresolved, and in many cases, the cellular environment has to be taken into account, making feature calculation even more complex. In REM, exact feedback about a protein's fitness is given by the experiment.

We have just introduced the concept of a more and more detailed property space, and the fitness evaluation can be seen as an exploration of this space. Furthermore, this exploration is a decompression of the original information (the amino acid sequence), because one needs more bits (units of information) to encode the three-dimensional structure than one needs to encode the amino acid sequence. We will now see that this decompression can be mirrored in the representation space.

Looking at the degree of information compression, one can imagine a more and more detailed (redundant) coding on the left-hand side, and a more and more detailed (evaluated) property space on the right-hand side, with a compact encoding of the phenotype in the center:

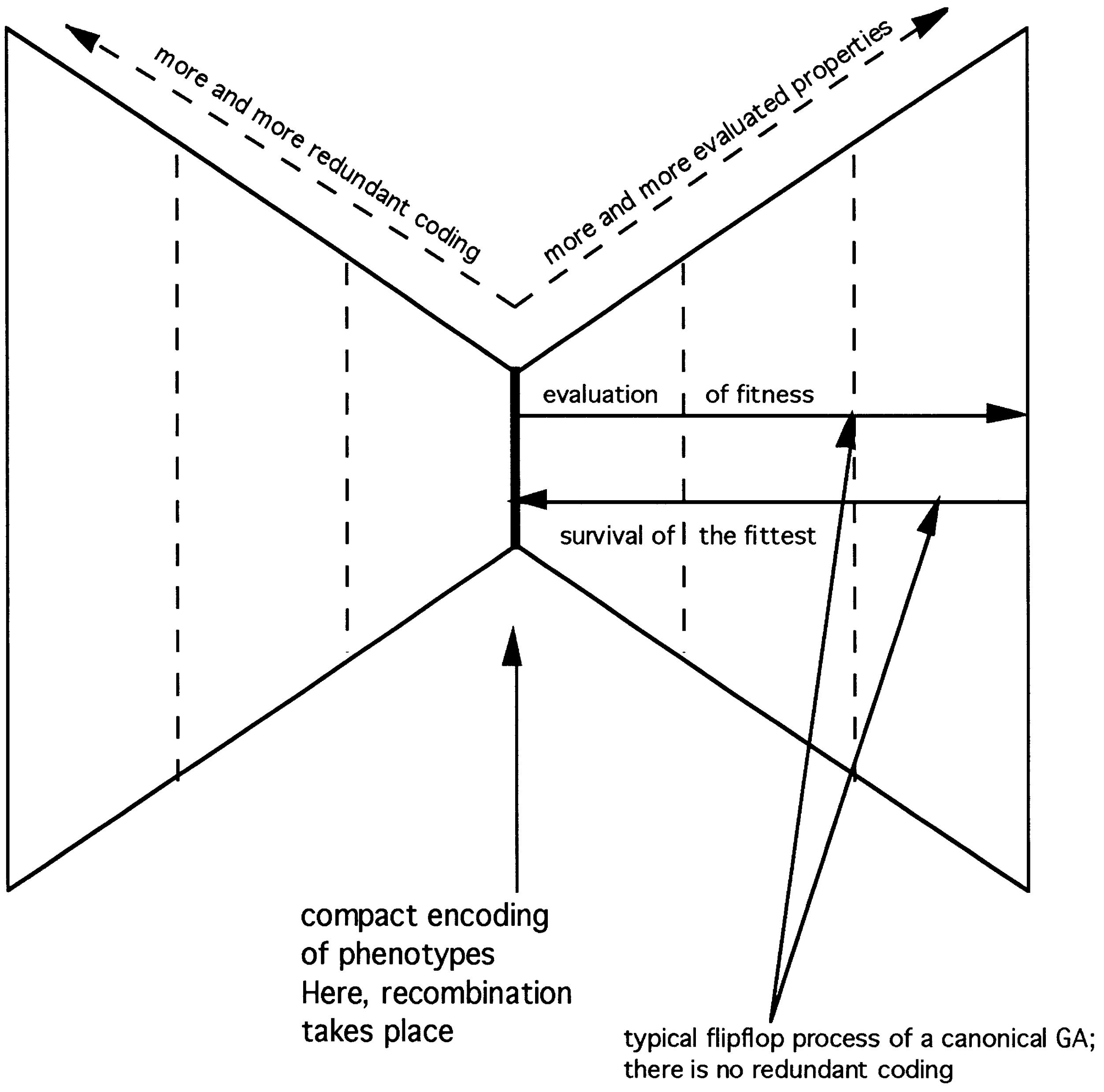
more and more redundant coding
more and more evaluated properties
evaluation of fitness
survival of the fittest
compact encoding
of phenotypes
Here, recombination
takes place
typical flipflop process of a canonical GA;
there is no redundant coding

Fig. 6. Action of a GA in an abstract setting. The representation spaces are on the left, and the physical property spaces are on the right.

On the left, every vertical line represents one possible representation space. The longer the line, the more redundant is the representation. On the right, every vertical line corresponds to one property space, the length of the line indicating its complexity (i.e., number of variables, degree of resolution).

For the protein engineering case, we utilize a redundant genotype (DNA) :

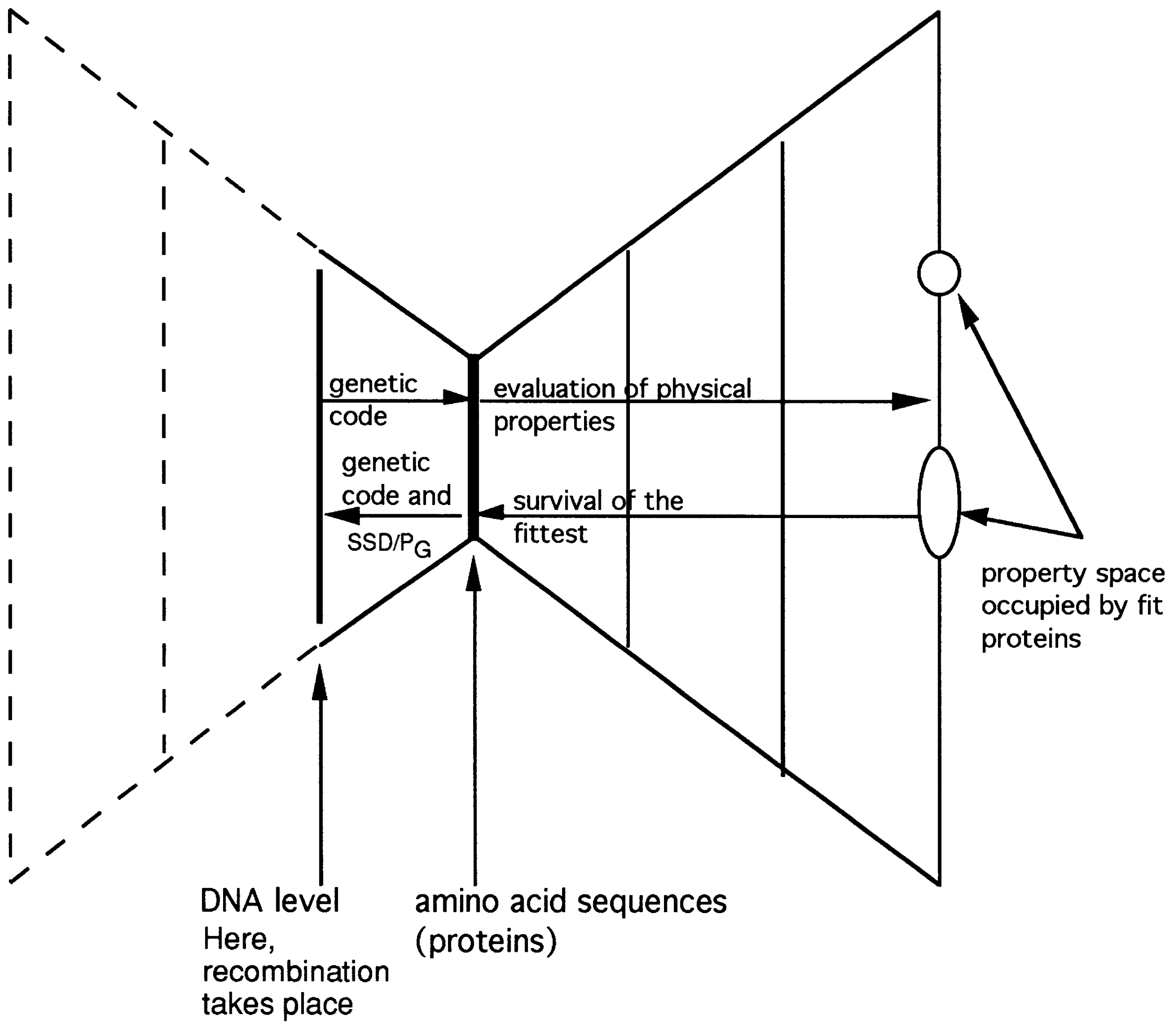
genetic code
evaluation of physical properties
genetic code and SSD/P_G
survival of the fittest
property space occupied by fit proteins
DNA level Here, recombination takes place
amino acid sequences (proteins)

Fig. 6. Action of REM in the abstract setting of Fig. 5

As we observed in section 3 and 4, the genetic code is not a random mapping from the primary phenotype to the genotype; instead the correlation of special physical properties (hydropathy, molar volume) and the second nucleotide position of the triplet helps in the optimization process. By including neighboring DNA nucleotide triplets, one can with high probability include amino acids neighboring in the physical property space. (This is the main side-effect of the SSD or P_G calculations.) In other words, a weak quasi-homomorphism (similar to Holland *et al.* 1986, Riolo 1991) between maps in the representational and the property space is exploited. Clusters in the representational space tend to correspond to clusters in the physical property space.

The last ingredient of our algorithm is the recombination taking place on the redundant encoding level. In the protein engineering case, this is the DNA level, and uniform crossover is simulated by the calculation of a doping scheme (forma) used for the generation of new DNA.

In general, generalized REM algorithms optimize in two directions: moving to the right, fitness is evaluated and used for selection; moving to the left, the mapping between representations of different redundancy is implicitly changing the population. On the left, reproduction can take place, hopefully recombining building blocks that contribute to high fitness on the right, and yielding new building blocks that represent structures of even higher fitness in the property space.

If our goal is the construction of a diverse population of fit individuals, generalized REM algorithms are especially suited because the calculated formae are the start of a new diverse generation of fit phenotypes.

Acknowledgments.

This work was funded by NIH GM42645, DOE 9102-025, DOE DE-FG02ER20019, and by the Human Frontiers Science Program.

References.

Arkin, A. P., E. Goldman, S. J. Robles, W. J. Coleman, C. A. Goddard, M. M. Yang and D. C. Youvan. 1990. Bio/Technology **8**: 746-749.

Arkin, A. P., Youvan, D. C. (1992). *Proc. Natl. Acad. Sci. U.S.A.* **89**:7811-7815.

Arkin, A. P., Youvan, D. C. (1993). In Deisenhofer H. & Norris JR. (eds) *The Photosynthetic Reaction Center, Vol. 1* (pp. 133-155) Academic Press, New York.

Delagrave, S., Goldman, E. R., Youvan, D. C. (1993). *Protein Eng.* **6**:327-331.

Delagrave, S., Youvan, D. C. (1993) *Bio/Technology.* In press.

Gerrits, M., Hoogeweg, P. (1990). In: *Parallel Problem Solving from Nature 1,* H.P. Schwefel, R. Manner, (Ed.) (pp. 70-74) Springer, Berlin.

Goldman, E. R., Youvan, D. C. (1992). *Bio/Technology* **10**:1557-1561.

Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, P.A. (1986). *Induction. Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge MA.

Houghten, R. A., C. Pinilla, S. E. Blondelle, J. R. Appel, C. T. Dooley and J. H. Cuervo. (1991). Nature **354**: 84-86.

Oliphant, A. R., Nussbaum, A. L., Struhl, K. (1986). *Gene* **44**:177-183.

Polani, D., Uthmann, T. (1992). In: *Parallel Problem Solving from Nature 2,* R. Manner, Manderick B (Ed.) (pp. 421-29) Elsevier publishing Co. Amsterdam.

Radcliffe, N.J. (1991). *Complex Systems* **5**: 183-205.

Ratledge, C. and D. Lewis. (1991). J. Chem. Tech. Biotech. **28**: 109-110.

Reidhaar-Olson, J. F., Bowie, J. U., Breyer, R. M., Hu, J. C., Knight, K. L., Lim, W. A., Mossing, M. C., Parsell, D. A., Shoemaker, K. R., Sauer, R. T. (1991). *Meth. Enzym.* **208**:564-587.

Riolo, R. (1991). In: *Proc. third International Conference on Genetic Algorithms.* J.D. Schaffer, (Ed.). (pp. 322-27) Morgan Kaufmann, San Mateo.

Robles, S., Youvan, D.C. (1993). *J. Mol. Biol.* **232**: 242-252.

Syswerda, G. (1991). In: *Proc. third International Conference on Genetic Algorithms,* J.D. Schaffer, (Ed.). (pp. 2-9) Morgan Kaufmann, San Mateo.

Unger, R., Moult, J.. (1993). *J. Mol. Biol.* **231**: 75-81.

Yang M. M., Youvan, D. C. (1988). *Bio/Technology* **8**:746-749.

Yang, M. M., W. J. Coleman and D. C. Youvan. 1990. *In* Reaction Centers of Photosynthetic Bacteria. M.-E. Michel-Beyerle. (Ed.) (Springer-Verlag, Germany) 209-218.

Youvan, D. C., Arkin, A. P., Yang M. M. (1992). In: *Parallel problem solving from Nature, 2* R. Maenner, Manderick B (Ed.) (pp 401-410) Elsevier publishing Co. Amsterdam.

Youvan, D. C., Goldman, E., Delagrave, S., & Yang, M. M. (1993). Meth. Enzym. in press.

Appendix

How to reproduce results from Ch. 2

Welcome to Xerion, and the add-on's for protein functionality estimation with Neural Networks.

This is the README for the directory "$XERIONDIR". $XERIONDIR is the value of a shell variable, currently /usr/people/georg/xerion, as defined in /usr/people/georg/.cshrc.

Relevant files are in the directories $XERIONDIR/nets/bp, $XERIONDIR/bp.sim, espec. $XERIONDIR/bp.sim/Paper, and in $XERIONDIR/doc/Problems.

See the README file and the documentation in $XERIONDIR/doc for more info on Xerion.

In the following, reproduction of results reported in Chapter 2 is discussed, and the programs and files are described.

```
------------------------------------------------
------------------------------------------------
Example: How to reproduce the data from Fig. 4C :
------------------------------------------------
------------------------------------------------
```

0. Disable the graphics, by typing "unsetenv DISPLAY"; otherwise, a memory handling bug in Xerion will cause higher and higher memory consumption.

1. Start bp, in $XERIONDIR/bp.sim, or one of its direct subdirectories.

You see:

```
>bp
Unable to open display, graphics disabled.
  Xerion, V3.1.147, Wed Apr 14 12:19:53 EDT 1993 IRIX System V Release 4.0.1
  Copyright (C) 1990, 1991, 1992, 1993 University of Toronto, Toronto, Canada.
  All rights reserved.
  Written by: Drew van Camp, Tony Plate.

Reading file "/usr/people/georg/xerion/config/xerionrc".
Reading file "/usr/people/georg/xerion/config/bprc".
bp->
```

2. Type explNoHiddenAExp, to generate the accuracy data for the network with exponential transfer, missing site L154 (that's site no. 3 if counting the mutagenized sites starting with 0, therefore the input indices are 6 and 7; there are 2 inputs per site).
   The following dialog will take place:

```
bp-> explNoHiddenAExp
Parameters:
No. of Inputs ? (12)16
Feature ? (M)
Min. No. of Hidden Neurons ? (0)1
Max. No. of Hidden Neurons ? [1 + 1 if only one net shall be explored] (4)2
```

```
No. of Outputs ? (2)
Specials ? (untold/none)Expi1616
Index of first disconnected Input ? (0)6
Index of second disconnected Input ? (0)7
How many rounds [16]? (16)
maximal testing set size [112/40]? (24) 40
minimal testing set size [32/8]? (8)
number of divisions [16]? (16)
number of retries ? [8](8)
Only 2 Outputs reasonable for 18-Nets
Created Net 16M1o2d6d7
Now make changes e.g. type  set currentNet.weightCost = 0.1
Then type in name of continuing script (explNoHiddenBExp)
```

3. Type in explNoHiddenBExp, and the network testing begins:

```
bp-> explNoHiddenBExp
Nets with 1 to 2 hidden neurons will be confronted 16 times with new (random)
data
Dealing with Nets having 1 Hidden Neurons
Round no. 0
Appending random Pseudo-Nulls to 16M1o2d6d7's natural
Training Set of Positives, 18PosData2, saving the whole set
in auto16M1o2d6d7.ex, which is loaded as a Training Set.
The Testing Set is deleted. The random seed is 10.
Appending/ Writing results to
/usr/people/georg/xerion/bp.sim:ferr16M1o2d6d7Expi1616
Current Testing Set Size is : 8
Will divide Data Set 16 times
Doing 8 Retries, writing to
/usr/people/georg/xerion/bp.sim:ferr16M1o2d6d7Expi1616
errorVec = { 0.75 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 }
```

4. If the script has finished, type quit

5. Calculate accuracies for the perceptron network in the same way, replacing explNoHiddenAExp with explNoHiddenALin, 1 Hidden Neuron with 0, and explNoHiddenBExp with explNoHiddenBLin.

6. Calculate accuracies for nets missing site L146 in the same way, the indices for this residue are 0 and 1, since L146 is the zeroeth mutagenized residue.

7. The files ferr16M1o2d6d7Expi1616, ferr16M1o2d6d7Lini1616, ferr16M1o2d0d1Expi1616, and ferr16M1o2d0d1Lini1616 should have been produced by step 0-6. Now produce the corresponding .dt files by extracting the relevant data using $XERIONDIR/PdataExtr. Type the following:

```
>$XERIONDIR/bp.sim/PdataExtr "ls ferr16M1o2d6d7Expi1616 ferr16M1o2d6d7Lini1616
ferr16M1o2d0d1Expi1616 ferr16M1o2d0d1Lini1616" 50
```

(For the data of Fig. 3, the last number must be 114 instead of 50, since there are 114 LC mutants but 50 RC mutants.)

8. Concatenate the .dt - files:

```
>cat ferr16Mlo2d6d7Expi1616.dt ferr16Mlo2d6d7Lini1616.dt
ferr16Mlo2d0d1Expi1616.dt ferr16Mlo2d0d1Lini1616.dt > data.dt
```

9. In termworks on the MAC, login to the UNIX workstation, and type

```
> more data.dt
```

10. copy the text into the clipboard

11. paste the clipboard into MS WORD

12. save the MS WORD file as "Text only", not as "Normal"

13. import the MS WORD "Text only" file using KaleidaGraph's "File -- Open" menu, skipping one line, reading titles, and using delimiter "space, >3" (this should all be default)

14. use copy&paste to transfer the data from ferr16Mlo2d6d7Lini1616, ferr16Mlo2d0d1Expi1616 and ferr16Mlo2d0d1Lini1616 [which appear in appended rows] into appended columns, using rows corresponding to that of the ferr16Mlo2d6d7Expi1616 data

15. Draw the plot according to the Kaleidagraph manual.

How to check results quantitatively via a script:

----

0. Disable the graphics, by typing "unsetenv DISPLAY";
   otherwise, a memory handling bug in Xerion will cause higher and higher memory consumption.

1. Start bp, in $XERIONDIR/bp.sim, or one of its direct subdirectories.
   In sub-subdirectories, you need to type $XERIONDIR/bp.sim/bp.
   If you need example importance (unequal weighting of LH1 vs. LH2 in the 3-class-seperation), follow the same procedure in $XERIONDIR/bp.simImp (you may have to uncompress the executable).

Note: There should also be an alias for 0) and 1): type "bpnG".

2. Call the appropiate expl... script
   depending on whether you need a special transfer function,
   or inclusion of true nulls.
   The script will ask for the relevant parameters.
   Use the variable "Specials" to denote intensity ("i1616"), changes that you do in step 2', the choice of transfer function ("Exp"), etc.

2'. If necessary, set a weight cost by typing "set currentNet.weightCost = X.Y"

3. Everything else is automatic after typing in the name of the continuing script. If the script has finished, type "quit".

```
4. Use "PdataExtr" from $XERIONDIR/bp.sim to produce an extract
   of the data file produced by the "bigTest" script.
   See PdataExtr_README for details.

-------------------------------------------------------------------
-------------------------------------------------------------------
How to check results qualitatively via the Graphical User Interface:
-------------------------------------------------------------------
-------------------------------------------------------------------

1. Start bp, in $XERIONDIR/bp.sim, or one of it's direct subdirectories.
   In sub-subdirectories, you need to type $XERIONDIR/bp.sim/bp.
   If you need example importance (unequal weighting of LH1 vs. LH2 in the
   3-class-seperation), follow the same process in $XERIONDIR/bp.simImp
   (you may have to uncompress the executable).

2. Create a net, by calling createNet inside the bp module.
   The script will ask for the relevant parameters.
   Via the Group types window, you can change the transfer function, etc,
   so that you do not need createNetExp, createNetLin.

2'.If necessary, set a weight cost by typing "set currentNet.weightCost = X.Y".

3. Generate an example set by typing "newRndDat". Use "newRndNulls"
   instead, if you want to include true nulls (LH data only).

4. Permute examples and move examples to the testing set by
   typing "moveTe". The script will ask you for the relevant parameters.

5. Use the "Learning methods" window to randomize the weights
   (via the menu "Network"), to reset the search, and to minimize error
   [i.e. train] (by pressing the "Do It!" button).

6. Use "classTest -t TESTING -stat" to calculate accuracy on the testing set;
   use "classTest -stat" to calculate accuracy on the training set.

7. Use "moveTe" for any new segregation of the data, and "newRndDat"
   to generate new random data.

---------------------------------------------
---------------------------------------------
How to export data from the SGI into a figure:
---------------------------------------------
---------------------------------------------

1) If you would like to plot the data from several data files
   in one plot, concatenate the .dt-files
2) copy them on the MAC into the clipboard via termworks
3) paste them into MS WORD
4) save the MS WORD file as "Text only", not as "Normal"
5) import the MS WORD "Text only" file using KaleidaGraph's "File -- Open"
   menu, skipping one line, reading titles, and using delimiter "space, >3"
   (this should all be default)
```

```
6) use copy&paste (the correspondence in file $XERIONDIR/bp.sim/Paper/
   ProblemsWithKaleidaGraph shows that
   there is no other way) to put data from the bottom to
   the right; append columns and change column names as appropiate.

-----------------------------------------------------------
-----------------------------------------------------------
This is the README for the directory $XERIONDIR/bp.sim/Paper
-----------------------------------------------------------
-----------------------------------------------------------

==========================================================
Abbreviations used for the filenames in this directory tree
==========================================================

ferr..   indicates an output file from the "bigtest" script
ferr..dt tabular data file extracted from the output file; contains
         Testing set size, mean, standard error of the mean,
         [ other values of minor interest ], and no. of reruns.

12       12 inputs ( i.e. LH data )
10       10 inputs ( i.e. LH data, ONE SITE LEFT OUT )
18       18 inputs ( i.e. RC data )
16       16 inputs ( i.e. RC data, ONE SITE LEFT OUT )

M        feature mask for molar volume and hydropathy
L        feature mask for artificial letter encoding

o2       two output neurons (2-class-categorization)
o3       3 output neurons (3-class-categorization)

dXdY     disconnect (omit) input units X and Y, e.g.
d10d11   ... 10 and 11, this is the +7 site for the LH data
d10d10   ... 10 only
d6d7     ... 6 and 7, this is L154 for the RC data

iXY      X segregations of data, Y inclusions of new random data
i1616    the standard intensity of the testing process

newL     using the better artificial letter encoding of nndNewL.c

Exp      using exponential transfer; default is logistic transfer

Lin/Perc using linear transfer; default is logistic transfer

none     using logistic transfer (this abbrev. is omitted later on)

K/Killed output file from the "bigtest" script has been killed before completion
         (e.g. because full evaluation would have taken weeks)

KXY      killed after XY rounds (inclusions of new data)

wtX.Y    using a weightCost of X.Y; default is 0
```

```
rdX.Y    using a zeroErrorRadius of X.Y; default is 0.1

retrX    using X retries (restarts of backprop w/ new random weights;
         default is 8)

Nulls    using true nulls

Imp      using example set importance (implicit in 3-class seperation data)

===================================================
CONTENTS OF DIRECTORY TREE, $XERIONDIR/bp.sim/Paper
===================================================

XYdata            # contains the extracted data in tabular form, for the
                  # respective Figure XY, and also the "data not shown".

FigXY/            # contains the (compressed) outputs
                  # of the "bigTest" script, and the extracted data in
                  # tabular form, for figure XY.
FigXY/            # contains also some of the "data not shown".

FigXY/dns/        # contains most of the "data not shown", sometimes sorted into
                  # subdirectories
FigXY/junk/       # contains junk data which may nevertheless reveal patterns
                  # not yet noticed

Fig5/             # contains the 3-class seperation data for the LH mutants

ProblemsWithKaleidaGraph/
                  # look here how to import data from the SGI into a figure
                  # and to learn about KaleidaGraph's deficiencies

UnpolishedButWorkingDataExtractors/
                  # if you have problems with PdataExtr in $XERIONDIR/bp.sim,
                  # you may use this unpolished one, who did most of the
                  # heavy-duty data extraction work

junkOld8_4data/   # data with 4 new inclusions of random data ("rounds"),
                  # and 8 segregations only

withTrueNulls/    # same as before, but including true nulls

---------------------------------------------------
---------------------------------------------------
This is the README for directory $XERIONDIR/nets/bp
---------------------------------------------------
---------------------------------------------------

=========================================
CONTENTS OF DIRECTORY, $XERIONDIR/nets/bp
=========================================

# Testing Script
```

```
bigTest             # Xerion script for the test process outlined in the
                    # inner box of PNAS draft Fig. 2;
                    # the script will ask for the relevant parameters.
conftable*          # alternative way of calculating accuracies (not used);
conftableReadme     # instead, the relativeError command is used by bigTest

# Network creating Scripts

createNet           # Xerion script creating one net; the script will ask
                    # for the relevant parameters; logistic transfer is used
createNetExp        # same as before; create one net w/ exponential transfer
createNetLin        # same as before; create one net w/ linear transfer

# Network exploration scripts, mainly implementing a loop over different numbers
# of hidden neurons, AND THE LOOP OVER DIFFERENT SETS OF RANDOM DATA

explNoHiddenA       # Start exploration: ask for loop,etc parameters
                    # and create Net (w/ logistic transfer)
explNoHiddenAExp    # same, create net w/ exponential transfer
explNoHiddenALin    # same, create net w/ linear transfer
explNoHiddenB       # Continue exploration: loop over various no's of hidden
                    # neurons, and test with different sets of random data
explNoHiddenBExp    # same, create net (if any) w/ exponential transfer
explNoHiddenBLin    # same, create net (if any) w/ linear transfer
explNoHiddenBNulls  # same, logistic transfer, use newRndNulls for nulls
explNoHiddenBNullsExp # same, exponential transfer, use newRndNulls

# Create new segration of data, to be used especially manually

moveTe              # Move all examples to training set, permute, and move
                    # specified no. of examples back to testing set

# Generate data sets with new random nulls

newRndDat           # Generating data sets w/ feature vectors and labels
                    # of positives and random nulls; no parameters.
newRndNulls         # same, but using true nulls as well.

# Generating feature vectors

nnd*                # Use this program on purpose only; nndNewL.c fixes a
nnd.c               # "bug" producing feature vectors containing zeroes

nndNewL*            # From a set of sequences (in the format defined at the
nndNewL.c           # start of nndNewL.c), generate appropiate feature vectors

# (Raw) data sets and generation of random nulls

12PosData2          # amino-acid sequ. of the LH POSITIVES w/ labels;
12PosData3          # '2' indicates 2-class seperation, '3' indic. 3-class sep.

12rng*              # called by the newRnd... scripts
12rng.c             # used to generate "random" nulls w/ Ellen's doping scheme
```

```
18PosData2          # amino-acid sequ. of the RC POSITIVES w/ labels;

18rng*              # called by the newRnd... scripts
18rng.c             # used to generate random nulls w/ NN(G,C) dope

12NullDataL2        # example sets appended by newRndNulls;
12NullDataL3        # contents are appropiate feature vectors for the TRUE NULLS
12NullDataM2        # '2' indicates 2-class seperation, '3' indic. 3-class sep.
12NullDataM3

# Other

tSzDvRz             # Also known as "bigTest", see above

0HiddenLayer.layout # used by createNet, see Xerion Manual p.13
1HiddenLayer.layout
2HiddenLayer.layout

./Backup/RawData:   # sequence data in raw form
./            120RawData   18RawData
../           12RawData

./Backup/RndExGenAndFeatureGen: # backup for the utilities
./        ../       12rng.c   18rng.c   nnd.c   nndNewL.c

./Devel:            # better, but untested scripts
./            00README      buildNet      datRndNulls
../           bigTest       datRndNew     explNet

./SlightlyOlderUnpolishedButWorking:
                    # if you observe problems with some of the scripts,
                    # use the older scripts from this directory
                    # instead by copying them to
                    # $XERIONDIR/nets/bp; these scripts were used for the main
                    # part of the heavy-duty data generation. The "expl..."
                    # scripts are named tExHi... in this directory

./TestProtocols:    # protocols on debugging the scripts, etc.
./                  protPnets
../                 protPnets.in
prot16L1o2d12d13n   protRz
```