Steffen Schulze-Kremer

Molecular Bioinformatics

Algorithms and Applications

Pointers to Free Internet/WWW Resources

W DE G

de Gruyter

Schulze-Kremer

Molecular Bioinformatics
Algorithms and Applications

Steffen Schulze-Kremer

Molecular Bioinformatics

Algorithms and Applications

Walter de Gruyter · Berlin · New York 1996

Dr. Steffen Schulze-Kremer
Westfälische Str. 56
D-10711 Berlin
Germany

email:
steffen@chemie.fu-berlin.de or
steffen@mycroft.rz-berlin.mpg.de

WWW:
http://www.chemie.fu-berlin.de/user/steffen or
http://mycroft.rz-berlin.mpg.de/~steffen

With 124 figures and 36 tables.
Cover illustration: Wire-frame-model of DNA. Courtesy of Steffen Schulze-Kremer.

Library of Congress Cataloging-in-Publication Data

Schulze-Kremer, Steffen.
Molecular bioinformatics: algorithms and applications / Steffen Schulze-Kremer.
Includes index.
ISBN 3-11-014113-2 (alk. paper)
1. Molecular biology – Computer simulation. I. Title.
OH506.D346 1995 95-40471
574.8'8'0113–dc20 CIP

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Schulze-Kremer, Steffen:
Molecular bioinformatics: algorithms and applications /
Steffen Schulze-Kremer. – Berlin; New York; de Gruyter, 1995
ISBN 3-11-014113-2

∞ Printed on acid-free paper which falls within the guidelines
of the ANSI to ensure permanence and durability.

© Copyright 1995 by Walter de Gruyter & Co., D-10785 Berlin
All rights reserved, includung those of translation into foreign languages. No part of this book may be reproduced or transmitted in any form or by any means, electronic of mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher. – Printed in Germany.
Converted by: Knipp Medien und Kommunikation, Dortmund. – Printing: Gerike GmbH, Berlin. Binding: Dieter Mikolai, Berlin. – Cover Design: Hansbernd Lindemann, Berlin.

To my friends,
especially Otto B., Jochen, Kathrin and Grit.

Preface

Molecular bioinformatics is a newly emerging interdisciplinary research area. It comprises the development and application of computational algorithms for the purpose of analysis, interpretation and prediction of data and for the design of experiments in biosciences. Background from computer science includes but is not limited to classical von Neumann computing, statistics and probability calculus, artificial intelligence, expert systems, machine learning, artificial neural nets, genetic algorithms, evolutionary computation, simulated annealing, genetic programming and artificial life. On the application side, focus is primarily on molecular biology, especially DNA sequence analysis and protein structure prediction. These two issues are also central to this book. Other application areas covered here are: interpretation of spectroscopic data and discovery of structure-function relationships in DNA and proteins. Figure 1 depicts the interdependence of computer science, molecular biology and molecular bioinformatics.

The justification for introducing a new label for a range of rather diverse research activities is motivated by the following four observations.

1) Exponential growth of data requires new ways of information processing.

A vast amount of genetic material has been sequenced to date. Many laboratories continue to output sequences of new genes world wide. There is an exponential growth[1] of known DNA sequences and protein structures. In March 1995, about 42.000 known protein sequences were present in the SwissProt[2] database and roughly 2900 three-dimensional structures of proteins, enzymes and viruses in the Brookhaven Protein Database[3]. The international Human Genome Organisation (HUGO) is currently attempting to sequence a complete human genome[4]. By March 1995, 3748 genes were mapped[5] in a total of approximately 25 million base pairs, which is still only about 0.8% of the total size in nucleotides of one entire human genome. Such quantities of data cannot be looked at and analysed without

1 L. Philipson (EMBL), *The Human Genome and Bioinformatics*, Proceedings of the Symposium **Bioinformatics in the 90's**, ASFRA Edam, Maastricht, 20. November, 1991.

2 R. D. Appel, A. Bairoch, D. F. Hochstrasser, *A new generation of information retrieval tools for biologists: the example of the ExPASy WWW server*, **Trends Biochem. Sci.**, vol 19, pp. 258-260, 1994. Internet World Wide Web Access at http://expasy.hcuge.ch or http://129.195.254.61. See Chapter 8.1 or ask your local computer specialist for details on WWW.

3 **PDB Newsletter**, Brookhaven National Laboratories, no 68, April 1994, Internet World Wide Web Server at http://www.pdb.bnl.gov.

4 T. Caskey, President of the Human Genome Organisation, HHMI Baylor College of Medicine I, Baylor Place, Houston, 77030 Texas, World Wide Web Server at http://www.bcm.tmc.edu.

5 HUGO Report Card March 1995, Internet World Wide Web Server of the Genome Data Bank at http://gdbwww.gdb.org.

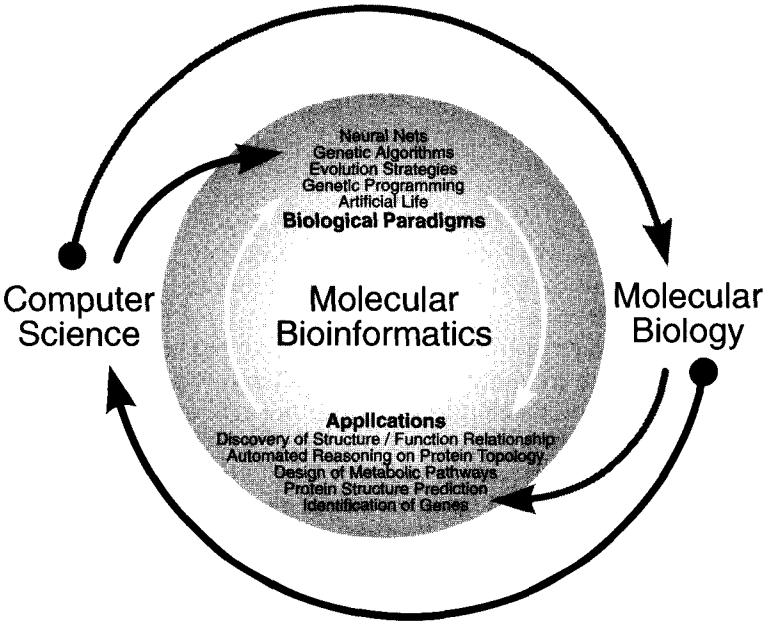


Figure 1: Scope of Molecular Bioinformatics.
Relation of molecular bioinformatics to computer science and biology. Computer science looks at problems in molecular biology and offers computational algorithms. Molecular biology looks at computer science and suggests new paradigms for information processing. These new approaches can be used to develop new applications which in turn may reveal new features in biology.

the help of computers. Specialised software obviously becomes an essential prerequisite for data storage and retrieval. This is a new challenge for traditionally empirically oriented bioscientists. It requires expertise in algorithmic theory and experience in programming computers *together* with a profound understanding of the underlying biological principles. Instead of being considered an appendix to either bioscience or computer science, such concentrated, interdisciplinary effort wants and deserves recognition on its own.

2) Better algorithms are needed to fully explore current biochemical databases.

In order to understand the purpose of a genetic sequence, it is in most cases not sufficient to perform a statistical analysis on the distribution of oligonucleotides. This is because patterns in DNA or protein tolerate slight alterations in some positions without losing their biochemical function. Furthermore, even for a simple, straightforward statistical analysis, e.g. on the frequency distribution of amino acid residue triplets to predict secondary structure, there is not (yet) enough data in our

databases[6]. More sophisticated methods are needed to explore the treasure of existing databases. *Data mining* becomes a key issue. A number of biocomputing related national and international research programs confirm this notion[7]. Computer scientists and mathematicians have been developing a variety of algorithms for quite some time, many of which bioscientists are not yet aware of. The establishment of molecular bioinformatics as a recognised, self-reliant discipline will be beneficial in promoting the dialogue between the two disciplines and to attract researchers from both areas.

3) Computer science learns from nature.

Molecular bioinformatics is not a one-way street. Biology has some interesting ideas to offer computer scientists, as can be seen in the case of artificial neural nets[8], genetic algorithms[9], evolution strategies[10], genetic programming[11], artificial life[12] and (from physics) simulated annealing[13]. To gather inspirations, to learn more from nature and to fully exploit existing approaches, a closer interaction between both disciplines is required. The professional endeavour to continue along these lines is covered neither by computer science nor biology. To fill this gap, molecular bioinformatics as a self-standing discipline can become the common platform for exchange and research in the aforementioned areas.

4) Creating a real-world application oriented forum.

Computer scientists sometimes are able to solve a particular type of problem efficiently and would like to apply their algorithms in a real-world situation to evaluate their performance and to gain new insights. If a computer scientist has not yet decided which area to turn to, he or she can profit from molecular bioinformatics. Here, they will find people they can communicate with in their technical language and who can introduce them to some of the most challenging and potentially rewarding problems science nowadays has to offer. One forum for such interaction has become the Internet course on Biocomputing at the Globewide Network

6 M. J. Rooman, S. J. Wodak, *Identification of predictive motives limited by protein structure data base size*, **Nature**, no 335, pp. 45-49, 1988.

7 Commission of the European Communities: Biomolecular Engineering (BEP 1982-1984), Biotechnology Action Program (BAP 1985-1989), Biotechnology Research for Innovation, Development and Growth in Europe (BRIDGE 1990-1993), Biotechnology (BIOTECH2 1994-1998). German Minister for Research and Technology: Molecular Bioinformatics (1993-1996).

8 M. Minsky, S. Papert, **Perceptrons – Expanded Edition**, MIT Press, Cambridge MA, 1988.

9 J. H. Holland, **Adaptation in natural and artificial systems**, University of Michigan Press, Ann Arbor, 1975.

10 I. Rechenberg, **Evolutionsstrategie**, Frommann-Holzboog, Stuttgart, 1973, 1994.

11 J. Koza, **Genetic Programming (I, II)**, MIT Press, Cambridge MA, 1993, 1994.

12 C. G. Langton C. Taylor, J. D. Farmer, S. Rasmussen (Eds.), **Artificial Life II**, Addison-Wesley, 1992.

13 S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, *Optimisation by Simulated Annealing*, **Science**, vol 220, no 4598, pp. 671-680, 1983.

Academy, Virtual School of Natural Sciences[14] that enables students and instructors from all over the world to participate in a university course.

This book attempts to provide an overview on advanced computer applications derived from and used in biosciences and in doing so to help define and promote the newly emerging focus on molecular bioinformatics. It is intented to serve as a guidebook for both biologists who are about to turn to computers and for computer scientists who are looking for challenging application areas. The reader will find a variety of modern methodological approaches applied to a number of quite divers topics. This collection gives a contemporary overview of what, in principle, can be achieved by computers in bioscience nowadays and what is yet difficult to grasp. In many places detailed information on the availability of software is included, preferably through Internet addresses of freely accessible World Wide Web Servers or by email. In the appendix, a list of Internet entry points to biocomputing facilities provides orientation to the novice in molecular bioinformatics and assists the specialist in staying abreast of the latest scientific developments.

The book is also intended to inspire further research. Researchers already familiar with molecular bioinformatics may still gather new ideas from the material presented as sometimes related methodologies are applied to unrelated problems and related problems are treated by completely different algorithms. The comparison in such cases may provide valuable insights.

In order to best serve an interdisciplinary audience, each chapter starts with an introduction into the mathematical basis of the algorithm used. Then, results of one or more original research papers applying that approach to different biochemical problems are presented and discussed. Finally, limitations and open questions are established and suggestions are offered along which lines to continue research. As in a manual, the material presented in this book should be sufficient to enable the reader to rebuild an application or to modify or extend it. Whether or not he or she intends to do so, the presentation of each topic should in any case be adequately detailed to allow the reader to decide if the proposed approach looks promising to be utilised for his or her own project.

Naturally, this book has its limits. Not all relevant work can be presented here. Notably, molecular mechanics, molecular dynamics and classic molecular modelling are important topics in molecular bioinformatics. However, these disciplines have matured and grown to such an extent that describing them thoroughly would make up a whole book by itself. In fact, such books have already been written[15,16]. The development of biological systems to act as data storage and com-

14 The author is one of the authors of the GNA-VSNS Biocomputing course. For more information, first send an email to the author at steffen@chemie.fu-berlin.de, steffen@mycroft.rz-berlin.mpg.de or contact vsns-bcd-faculty@bioinformatics.weizmann.ac.il.

15 C. L. Brooks III, M. Karplus, B. M. Pettitt, **Proteins: A theoretical perspective of dynamics, structures and thermodynamics**, Wiley, 1988.

16 A. M. Lesk, **Protein Architecture – A practical approach**, IRL Press, Oxford, 1991.

puting devices[17,18,19] is also not covered in this book. This more empirically oriented work contrasts the computational aspect of molecular bioinformatics as emphasised here.

A choice was made to include some early projects in molecular bioinformatics as these comprise the roots of this discipline. A number of important conclusions can be drawn from these works which are still valid and which will need to be considered in future applications. The larger part of this book, however, is devoted to fairly recent work. To aid the reader continue his or her way through molecular bioinformatics, the first chapter also contains references to a number of approaches that had to be omitted.

I hope this book will bring inspiration and information to its readers.

17 B. H. Robinson, N. C. Seeman, *The design of a biochip: a self-assembling molecular-scale memory device*, **Protein Engineering**, vol 1, no 4, pp. 295-300, 1987.

18 B.C. Crandall, J. Lewis (Eds.), **Nanotechnology: Research and Perspectives**, ISBN 0-262-03195-7, 1992.

19 **Proceedings of the International Conference on Molecular Electronics and Biocomputing**, September 1994, Goa, India, Tata Institute of Fundamental Research, Ratna S. Phadke, email ratna@tifrvax.tifr.res.in.

Table of Contents

Preface VII

1. Introduction 1

1.1. Methodologies 3
1.2. Application Areas 8

2. Artificial Intelligence & Expert Systems 13

2.1. Methodology 13
2.1.1. Symbolic Computation 14
2.1.2. Knowledge Representation 18
2.1.3. Knowledge Processing 29
2.2. Applications 34
2.2.1. Computer Aided Reasoning in Molecular Biology 34
2.2.2. Knowledge based Representation of Materials and Methods 40
2.2.3. Knowledge based Exploration in Molecular Pathology 44
2.2.4. Planning Cloning Experiments with Molgen 46
2.2.5. Expert Systems for Protein Purification 62
2.2.6. Knowledge based Prediction of Gene Structure 87
2.2.7. Artificial Intelligence for Interpretation of NMR Spectra 94
2.2.7.1. Nuclear Magnetic Resonance 94
2.2.7.2. Protean 104
2.2.7.3. Protein NMR Assistant 108

3. Predicate Logic, Prolog & Protein Structure 111

3.1. Methodology 111
3.1.1. Syntax 111
3.1.2. Connectives 112
3.1.3. Quantification and Inference 113
3.1.4. Unification 114
3.1.5. Resolution 116
3.1.6. Reasoning by Analogy 117
3.2. Applications 122
3.2.1. Example: Molecular Regulation of λ-Virus in Prolog 122
3.2.2. Knowledge based Encoding of Protein Topology in Prolog 126
3.2.3. Protein Topology Prediction through Constraint Satisfaction 133
3.2.4. Inductive Logic Programming in Molecular Bioinformatics 142
3.2.4.1. Trimethoprim Analogues 147

3.2.4.2. Drug Design of Thermolysin Inhibitors 149
3.2.4.3. α-Helix Prediction 150

4. Machine Learning of Concepts in Molecular Biology 152

4.1. Methodology 152
4.1.1. Learning Hierarchical Classifications: Cobweb/3 152
4.1.2. Learning Partitional Classifications: AutoClass III 159
4.2. Applications 162
4.2.1. Inductive Analysis of Protein Super-Secondary Structure 162
4.2.1.1. Properties of Secondary Structures 164
4.2.1.2. PRL Database 168
4.2.1.3. α-Helix/α-Helix Pairs 175
4.2.1.4. Helix/β-Strand Pairs 187
4.2.2. Symbolic Induction on Protein and DNA Sequences 197
4.2.2.1. Decision Trees over Regular Patterns 200
4.2.2.2. Searching Signal Peptide Patterns in Predicate Logic Hypothesis Space 205

5. Evolutionary Computation 211

5.1. Methodology 212
5.1.1. Genetic Algorithms 212
5.1.2. Evolution Strategy 219
5.1.3. Genetic Programming 220
5.1.4. Simulated Annealing 230
5.2. Applications 236
5.2.1. 2-D Protein Model for Conformation Search 236
5.2.2. Protein Folding Simulation by Force Field Optimisation 246
5.2.2.1. Representation Formalism 247
5.2.2.2. Fitness Function 249
5.2.2.3. Conformational Energy 250
5.2.2.4. Genetic Operators 251
5.2.2.5. Ab initio Prediction Results 252
5.2.2.6. Side Chain Placement 256
5.2.3. Multi-Criteria Optimisation of Protein Conformations 256
5.2.3.1. Vector Fitness Function 258
5.2.3.2. Specialised Genetic Operators 260
5.2.3.3. Results 262
5.2.4. Protein – Substrate Docking 267
5.2.4.1. Distance Constraint Docking 268
5.2.4.2. Energy driven Docking 270

6. Artificial Neural Networks 272

6.1. Methodology 272
6.1.1. Perceptron and Backpropagation Network 273
6.1.2. Kohonen Network 276

6.2. Applications 279
6.2.1. Exon-Intron Boundary Recognition 280
6.2.2. Secondary Structure Prediction 284
6.2.3. ϕ / ψ Torsion Angle Prediction 286
6.2.4. Super-Secondary Structure Detection 288

7. Summary, Conclusion & Prospects 291

8. Appendix 293

8.1. Internet Entry Points for Biocomputing 293
8.1.1. Information, Literature, References 293
8.1.2. Institutions Dealing with Molecular Bioinformatics 294
8.1.3. Databases 294
8.1.4. Search Tools for World Wide Web and FTP Sites 295
8.1.5. Software Resources 296

Index 297

1. Introduction

Although molecular bioinformatics has only recently become a self-reliant, recognised discipline[1,2,3], first work in the spirit of molecular bioinformatics dates back to the late 50's[4,5,6,7,8,9]. Since then the idea spread that computer scientists and biologists can profit from one another. Growing interest in interdisciplinary research connecting computer science and molecular biology came primarily from an exponential growth of known biological sequence[10] and structure[11] data, from the need for more sophisticated methods of data analysis and interpretation in biosciences, and from the discovery of nature as a source of models for efficient computation.

Computer scientists have discovered this trend and included workshops on molecular bioinformatics into most of their important international conferences. The proceedings and workshop notes of the Hawaii International Conference on System Sciences (HICSS)[12], the American Association for Artificial Intelligence (AAAI)[13], the conference on Parallel Problem Solving from Nature (PPSN)[14], the European Conference on Machine Learning (ECML)[15], the International Con-

---

1 Commission of the European Communities: Biotechnology Action Program (BAP 1985-1989), Biotechnology Research for Innovation, Development and Growth in Europe (BRIDGE 1990-1993), BIOTECH2 (1994-1998). German Minister for Research and Technology: Molecular Bioinformatics (1993-1996).

2 L. Hunter (Ed.), **Artificial Intelligence and Molecular Biology**, MIT Press, Cambridge MA, 1993.

3 S. Schulze-Kremer (Ed.), **Advances in Molecular Bioinformatics**, IOS Press, 1994.

4 F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, **Psychological Review**, vol 65, pp. 386-408, 1958.

5 M. Minsky, S. Papert, **Perceptrons**, MIT Press, Cambridge MA, 1969.

6 I. Rechenberg, **Evolutionsstrategie**, Frommann-Holzboog, Stuttgart, 1973, 1994.

7 J. H. Holland, **Adaptation in natural and artificial systems**, University of Michigan Press, Ann Arbor, 1975.

8 M. Stefik, *Planning with Constraints*, **Artificial Intelligence**, no 16, pp. 111-169, 1981.

9 B. G. Buchanan, E. H. Shortliffe, **Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project**, Addison-Wesely, 1984.

10 A. Bairoch, B. Boeckmann, *The Swiss-prot protein sequence data bank*, **Nucl. Acids Res.**, vol 20, pp. 2019-2022, 1992. Internet World Wide Web Access at http://expasy.hcuge.ch.

11 **PDB Newsletter**, Brookhaven National Laboratories, no 68, April 1994. Internet World Wide Web Server at http://www.pdb.bnl.gov.

12 **Proceedings of the 26th and 27th Hawaii International Conference on System Sciences**, IEEE Computer Society Press, Los Alamitos CA, 1993, 1994.

13 **Proceedings of the 9th Conference on Artificial Intelligence**, AAAI Press, Menlo Park CA, 1991.

14 **Proceedings of the 2nd Conference on Parallel Problem Solving from Nature**, (R. Männer, B. Manderick Eds.), North Holland, 1992.

15 **Proceedings of the European Conference on Machine Learning**, (P. B. Brazdil Ed.), Springer, 1993.

ference on Genetic Algorithms (ICGA)[16], the International Joint Conference on Artificial Intelligence (IJCAI)[17], and especially the International Conference on Intelligent Systems for Molecular Biology (ISMB)[18] are all valuable recourses of information. A freely accessible database of international researchers working in molecular bioinformatics is available[19].

Molecular bioinformatics comprises three fundamental domains: the development of algorithms and computer programs for applications in biosciences; the abstraction of biological principles for new ways of information processing; and the design and use of biochemical systems for data storage and as computing devices[20,21]. This book concentrates on the first two issues. The design of an artificial molecular computer is expected to become a feasible and profitable endeavour in the next decade but will not be treated here. Molecular mechanics, molecular dynamics[22] and classic molecular modelling[23] are also underrepresented here. This is because these topics have already been around for some time and an exhaustive treatment of their issues would fill several books on their own. Nevertheless, they contribute significantly to molecular bioinformatics.

Although computer science and biology can be treated separately, and, in fact, were so most of the time, there are four levels of interaction between these two disciplines.

1) Computer programs can help analyse biological data.
2) Computer models and simulation can help explain biological behaviour.
3) Atomic interactions can be used to build a molecular data processor.
4) Biology can serve as a source of models for computational algorithms.

Note the analogy between 1 and 3 and between 2 and 4. Topics 1 and 3 both use one discipline to process / enact the fundamental principle of the other. Topics 2 and 4 use one discipline to model / simulate the other's emergent behaviour. There is also a possibly recursive interaction. Contributions from biology can be used to create computer programs (or computers) that in turn can be used to operate on biological data. This happens, for example, in the application of genetic algorithms to the problem of protein folding. In this case, the original biological structure, a gene, and the biological algorithm, evolution, have been abstracted to give a general computational model. The artificial "genes" of that model may then be mapped

16 **Proceedings of the 4th International Conference on Genetic Algorithms**, (R. K. Belew, L. B. Booker Eds.), Morgan Kaufmann, Los Altos CA, 1991.
17 **Proceedings of the Artificial Intelligence and Genome Workshop 26 at the IJCAI-93**, (J.-G. Ganascia Ed.), Institut Blaise Pascal, Laforia, Universite Paris VI, CNRS, 1993.
18 **Proceedings of the 1st and 2nd International Conference on Intelligent Systems for Molecular Biology**, (Eds. I: L. Hunter, D. Searls, J. Shavlik, II: R. Altman, D. Brutlag, P. Karp, R. Lathrop, D. Searls), AAAI Press, Menlo Park CA, 1993, 1994.
19 L. Hunter, **AIMB database**, anonymous ftp at lhc.nlm.nih.gov (130.14.1.128), directory /pub/aimb-db, 1993. Internet World Wide Web Server at http://www.nlm.nih.gov.
20 B. H. Robinson, N. C. Seeman, *The design of a biochip: a self-assembling molecular-scale memory device*, **Protein Engineering**, vol 1, no 4, pp. 295-300, 1987.
21 B. C. Crandall, J. Lewis (Eds.), **Nanotechnology: Research and Perspectives**, ISBN 0-262-03195-7, 1992.
22 C. L. Brooks III, M. Karplus, B. M. Pettitt, **Proteins: A theoretical perspective of dynamics, structures and thermodynamics**, Wiley, 1988.
23 A. M. Lesk, **Protein Architecture – A Practical Approach**, IRL Press, Oxford, 1991.

onto a linear transformation (i.e. proteins) of the very objects from which they originally had been derived. – A schematic diagram on the interplay of computer science, molecular biology and molecular bioinformatics is shown in Figure 1 (in the PREFACE).

The prior example illustrates the possibility of raising confusion by the use of technical terms from different contexts. A gene in biology has one meaning, but a different one in the context of a genetic algorithm. The same goes for neural networks or evolution strategies. In this book, definitions from both biology and computer science will be provided and care is taken not to confuse them.

This book contains a collection of methodological approaches applied to a variety of information processing tasks in molecular biology. Emphasis is put on methods that explore algorithmic features found in nature. Preferred applications are protein structure analysis and interpretation of genomic sequences.

The main text of this book is divided into the following five chapters. Each chapter covers a group of algorithmically related methodologies of data processing, e.g. artificial neural networks or genetic algorithms. First in each chapter, general features of the computational methodologies used are described. Then a number of sections follow each containing an application of that method to a different biological problem. Each section starts with an introduction to the biochemical background of the problem and on any application-dependent modifications of the method. Then, specific details of the implementation for that application are described. Original results from one or more research groups are presented to illustrate and evaluate the performance of that approach.

This book can be used as a guidebook for novices in computer science and molecular biology to explore the scope of molecular bioinformatics. For specialists, it can serve as a manual to rebuild the methods and applications presented here or to extend and adopt them to fit one's own subject of interest. In this case, it is valuable to compare the results of different methodologies on the same problem, e.g. on prediction of protein secondary structure.

The final part of this book is for reference. The reader will find a list of Internet entry points that can be used by anybody with a computer connected to the Internet to navigate in the World Wide Web and retrieve information on people, institutions, research projects and software in molecular bioinformatics.

1.1. Methodologies

Table 1 gives an overview of the algorithmic techniques used for molecular bioinformatics. The main methodologies are briefly described in the following paragraphs.

Artificial intelligence, symbolic computation and expert systems. The shift from numeric to symbolic computation and the focus on information processing procedures analogous to human mental processes distinguishes artificial intelligence programming from classic, procedural programming. Although artificial intelligence programs are also finally translated into a series of sequential instructions for one or more central procession units of a computer, they often al-

Methodology	Technique	Paradigm
Artificial Intelligence		human
	Expert Systems	
	Inference	
	Imagery /Vision	
	Knowledge Representation	
	Linguistics	
	Pattern Matching	
	Planning	
	Qualitative Theory	
	Heuristic Search	
Artificial Life		evolution
Artificial Neural Nets		neurons
Classifier Systems		economy
Evolutionary Computation		nature
	Evolution Strategies	evolution
	Genetic Algorithms	genetics
Practical Experimental Work		none
	Circular Dichroism	
	Molecular Biology	
	Nuclear Magnetic Resonance	
	X-ray Crystallography	
Genetic Programming		genetics
Logic		none
	Predicate Calculus	
	Prolog	
Machine Learning		human
	Conceptual Clustering	
	Decision Trees	
	Inference	
Mathematics		none
	Dynamic Programming	
	Graph Theory	
	Information Theory	
	Probability Calculus	
	Statistics (Monte Carlo)	
Simulated Annealing		physics

Table 1: Methodologies and Algorithms used in Molecular Bioinformatics.
This is an overview on some techniques used in molecular bioinformatics. "Paradigm" tells if the technique has a model in nature. Mathematics and logic are thought of as *a priori* concepts.

low a complex task to be represented in a more natural, efficient manner. Semantic use of symbols other than numerical and character variables provide the means

for coding highly structured applications. Lisp[24] and Prolog[25] are special symbol manipulating programming languages which are often used for coding artificial intelligence programs. They allow easy implementation of self-modifying and recursive programs. In Lisp, data and program can alter each other during run time, which permits flexible flow control[26]. Artificial intelligence research has produced various sophisticated applications, three of which shall be briefly mentioned here. AM[27] discovered concepts in elementary mathematics and set theory on the basis of knowledge on *mathematical aesthetics*. AM combines the features of frame representation, production systems and best-first search. Eliza[28] simulates a psychiatrist talking to a patient. Several patients actually believed there was a human responding. Macsyma[29] was the first program to actually carry out symbolic differentiation and integration of algebraic expressions. Since then, it has been extended to become a versatile and flexible mathematical package not only on Lisp machines.

Artificial life. This rather new research area focuses on the realisation of life-like behaviour in man-made systems consisting of populations of semi-autonomous entities whose local interactions with one another are governed by a set of simple rules[30]. Key concepts in artificial life research are local definition of components, parallelism, self-organisation, adaptability, development, evolution, and emergent behaviour. Apart from being a philosophically fascinating area, artificial life in the future could help provide simulation models for interacting biological systems (e.g. cells, organisms) and support the design of *software robots*[31].

Artificial neural networks. The intention to abstract the behaviour of biological neural networks from their natural environment and to implement their function on a computer produced a large variety of so-called artificial neural networks[4,32]. These programs can often discriminate or cluster data better than statistics or probability calculus. Artificial neural networks are non-deterministic in the sense that their response depends on the order in which the training examples are presented to the net. The goal of simulating the behaviour of actual biological neurons and neural networks is not part of the mainstream in artificial neural network research. It turned out that an accurate, detailed simulation of the behaviour of only one biological neuron requires much more computational effort than is ben-

24 P. H. Winston, B. K. P. Horn, **LISP**, Addison-Wesely, 1984.
25 W. F. Clocksin, C. S. Mellish, **Programming in Prolog**, Springer, 1984.
26 E. Charniak, D. McDermott, **Introduction to Artificial Intelligence**, Addison-Wesley, 1985.
27 D. B. Lenat, *AM: Discovery in mathematics as heuristic search*, in **Knowledge-Based Systems in Artificial Intelligence** (R. Davis, D. B. Lenat Eds.), pp. 1-225, McGraw-Hill, New York, 1982.
28 J. Weizenbaum, *ELIZA – A computer program for the study of natural language communication between man and machine*, **Communications of the ACM**, vol 9, no 1, 1965.
29 C. Engleman, W. Martin, J. Moses, M. R. Genesereth, Macsyma Reference Manual, Technical Report, Massachusetts Institute of Technology, Cambridge MA and Symbolics Inc., 1977.
30 C. Langton (Ed.), **Artificial Life**, Addison-Wesley, p. xxii, 1989. Internet World Wide Web Server at http://alife.santafe.edu.
31 Softbods are autonomous agents that interact with real-world software environments such as operating systems or databases. Get more information from the Internet World Wide Web server at http://www.cs.washington.edu/research/projects/softbots/www/softbots.html.
32 J. A. Freeman, D. M. Skapura, **Neural Networks**, Addison-Wesley, 1991.

eficial for training artificial neural networks to perform data analysis tasks. Using more biologically detailed models has not yet significantly improved prediction performance of artificial neural networks.

Genetic algorithms and evolutionary computation. The idea to let a computer develop a solution for a problem like nature produces fit individuals by evolution led to the invention of evolution strategies[6] and genetic algorithms[7]. These approaches use "genetic" operators to manipulate numeric or string representations of potential solutions ("individuals"). A fitness function ranks all "individuals" of one "generation", the best of which are allowed to "survive" and to "reproduce". This is repeated for a fixed number of cycles or until a particular fitness criterion is fulfilled. Genetic algorithms operate on the "genotype" of a potential solution, evolution strategies on its "phenotype". Both methods have been shown to be superior to e.g. Monte Carlo search in extremely difficult search spaces. They are also non-deterministic because genetic operators are performed probabilistically.

Knowledge representation formalisms. Knowledge representation is a central issue in artificial intelligence research. As with humans[33], language also determines the limits of a computer's operation. The objective is therefore to find a representation formalism that can capture all essential details of an application, one that is compatible with the algorithm used and which can be coded efficiently. It should also allow intuitive use and easy maintenance of a knowledge base[34]. Prominent knowledge representation formalisms are objects[26], frames[35], scripts[36], production rules[9], predicates[37], decision trees[38], semantic nets[39], λ-calculus and functions[40], and classic, procedural subroutines. Currently, object-oriented programming style is becoming very popular, as can be seen in the wide-spread use of the object-oriented programming language C++ with its consequent use of classes[41].

33 L. Wittgenstein, **Tractatus Logico-Philosophicus**, Suhrkamp Edition, 1982.

34 The notion of a database traditionally means keeping data, possibly sorted, in one place. A knowledge base emphasises the data to be stored in a higher structured format, e.g. as rules or objects. Knowledge bases tend to be used interactively, as e.g. in expert systems. Also, a knowledge base is automatically updated and sometimes even generated automatically. There may be a mechanism to guarantee truth maintenance. In this book, the concepts of database and knowledge base are used almost interchangeably, as the difference between them is more one of emphasising their use and context rather than their contents. Almost any database can be used as a knowledge base, if properly accessed by sophisticated algorithms. In this book, the concept of a data bank is used to refer to (commercially) available collections of "raw" data, as e.g. a sequence data bank or a protein structure data bank.

35 M. Minsky, *A Framework for Representing Knowledge*, in **The Psychology of Computer Vision**, (P. H. Winston Ed.), MacGraw-Hill, New York, 1975.

36 R. C. Schank, R. P. Abelson, **Scripts, Plans, Goals and Understanding**, Erlbaum Hillsdale, New Jersey, 1977.

37 W. F. Clocksin, C. S. Mellish, **Programming in Prolog**, Springer, 1984.

38 J. R. Quinlan, *Discovering Rules by Induction from Large Collections of Examples*, in **Introductory Readings in Expert Systems**, (D. Michie Ed.), pp. 33-46, Gordon and Breach, London, 1979.

39 R. Quillian, *Semantic Memory*, in **Semantic Information Processing**, (M. Minsky Ed.), MIT Press, Cambridge MA, 1968.

40 J. McCarthy, *Recursive Functions of Symbolic Expressions and their Computation by Machine*, **Communications of the ACM**, vol 3, no 4, pp. 185-196, 1960.

41 M. A. Ellis, B. Stroustrup, **The Annotated C++ Reference Manual**, Addison-Wesley, 1990.

Machine learning. The field of machine learning studies computational methods for acquiring new knowledge, new skills, and new ways to organise existing knowledge[42]. This includes the modelling of human learning mechanisms. For example, Checkers[43] is a program that actually learned to play checkers better than expert human players. It did so by repetitive training, selecting relevant features from the board and weighting them properly. There are a number of ways to learn: rote learning, learning from instruction, learning by analogy, learning by deduction, learning by induction, learning by abduction, supervised learning and unsupervised learning. Computers can already be programmed to exhibit these types of learning on restricted domains. Machine learning algorithms can sometimes find hidden regularities and patterns in a biological database, which are invisible to purely statistical methods.

Logic and predicate calculus. The application of basic logic operators (not $\neg$, and $\wedge$, or $\vee$, follows $\rightarrow$, equivalent $\equiv$, there exists $\exists$, for all $\forall$) on simple predicates defines first order logic. Using first order logic as the basic principle for a computer programming language led to the origin of Prolog[37]. In contrast to procedural programming languages like Basic or Fortran, in Prolog emphasis is on how to *describe a problem*, not *the solution*. A built-in deductive inference machine then performs inverse resolution to derive valid solutions. The power of first order logic and Prolog comes from the ability to easily express non-numerical constraints, as is fundamental e.g. for the description of protein structure topologies.

Simulated annealing. The process of crystallisation at gradually decreasing temperatures can be abstracted to give a general optimisation procedure. Simulated annealing[44] performs a search in a multi-modal search space by exploring the valleys (if a global minimum is desired) in a random manner. As the (simulated) temperature decreases, less (fictitious, simulated) energy is becoming available to overcome the hills between neighbouring valleys. Typically, at the end of the run the probe arrives at a rather deep valley, which indicates a good solution. However, the algorithm does not guarantee that the global optimum be found. Simulated annealing is therefore preferably used on analytically intractable problems where one has no choice but to explore a large search space. The *ab initio* prediction of energetically favourable protein conformations is one such problem.

Statistics and probability calculus. Classic statistics and probability calculus provide well founded methods to analyse numerical databases. These methods still serve as the standard against which other approaches are measured. Any new methodology has to prove its merit by producing at least comparable results to statistics and probability calculus or by presenting qualitatively new statements that could not be derived by either of them.

---

42 **Encyclopedia of Artificial Intelligence**, (S. C. Shapiro Ed.), Wiley-Interscience, pp. 464, 1987.

43 A. L. Samuel, *Some Studies in Machine Learning using the Game of Checkers*, in **Computers and Thought**, (E. A. Feigenbaum, J. Feldman Eds.), pp. 71-105, McGraw-Hill, New Yorck, 1963.

44 S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, *Optimization by Simulated Annealing*, **Science**, vol 220, no 4598, pp. 671-680, 1983.

Classifier systems. One type of message passing, rule-based production systems in which many rules can be active simultaneously is called classifier system[45]. In such systems each rule can be interpreted as a tentative hypothesis on some factors relevant to the task to be solved, competing against other plausible hypotheses that are carried along and evaluated in parallel. Classifier systems got their name from the fact that they classify incoming messages from the environment into general sets. The condition parts of the production rules are matched against incoming messages from the environment and the action parts effect changes in the environment. Rules are assigned a strength value on the basis of their observed usefulness to the system. New rules are generated by genetic operators as in genetic algorithms. Classifier systems have been shown to learn strategies for successful, optimised behaviour in an adapting environment (e.g. poker play[46]).

Genetic programming. Derived from genetic algorithms is the concept of genetic programming[47]. Here, not a single solution to a particular problem is processed in an evolutionary manner but whole computer programs instead. Here again, Lisp is the programming language of choice. Similar to artificial neural networks, a fitness function measures how well the generated program reproduces known input / output values. When the training is finished, the program may be inspected, simplified and tried on a new set of input values with prior unknown results. One particular advantage of this approach is the ability to discover a symbolic expression of a mathematical function, not only a numerical approximation. By defining a proper set of primitives as basic building blocks (e.g. SIN, COS, EXP, LN for mathematical problems, or other, utterly application-specific user-defined operators) one can predetermine the appearance of the final solution.

1.2. Application Areas

The main application areas covered in this book are proteins and genes, but there are also a few other tasks not directly linked to either of them. Table 2 gives an overview of application areas in molecular bioinformatics. There is certainly more work that would fit under the label of molecular bioinformatics. However, this book is intended to give a representative selection, not a complete account of all work done. Owing to space limitations, a number of applications cannot be described here in detail. Some of them are mentioned and referenced below or in the respective chapters. The following paragraphs briefly describe the key issues in the main application areas.

Biotechnology. Major objectives in biotechnology are the industrial exploitation of micro-organisms, the use of biomolecules in technical applications and the

45 J. H. Holland, *Escaping Brittleness, The possibilities of general-purpose learning algorithms applied to parallel rule-based systems*, in **Machine Learning II**, (R. S. Michalsky, J. G. Carbonell, T. M. Mitchell Eds.), Morgan Kaufmann, Los Altos CA, pp. 593-623, 1986.

46 S. Smith, *A Learning System based on Genetic Algorithms*, **Ph.D. Dissertation**, Department of Computer Science, University of Pittsburg, PA, 1980.

47 J. Koza, **Genetic Programming (I, II)**, MIT Press, 1993, 1994.

invention of tools to automate laboratory work. Biochips[20,21,48] are mixtures of biological macromolecules that spontaneously assemble into an ordered, three-dimensional arrangement. Self-assembly of these complexes can be guided by stretches of complementary RNA. Such an association of macromolecules may then be used as a memory device. In that context, excitation of chemical groups by light or electrical stimulation can make molecules temporarily change their conformation or composition. Later, a different stimulation can be used to read the stored information or to clear it. Advantages of biochips are their ability to self-assemble and the use of inexpensive, organic components which can be produced by gene technology and micro-organisms.

Biosensors[49] consist of immobilised enzymes catalysing a reaction which consumes the compound to be measured while producing ions. A small, semipermeable tube containing the enzyme and an electrode is introduced into the probe. The reaction of enzyme and compound produces a concentration change of permeable ions. This results in a electrical signal that can be measured and amplified. The amplitude of that signal is correlated to the concentration of the compound. Biosensors are useful for monitoring concentration changes of critical metabolites in medicine. They are available for a wide range of reactions and are highly specific.

Databases. The recent increase in genetic sequence and protein structure data, and also information about micro-organisms, enzymes and chemical reactions required the design and development of specialised databases. Some established concepts for database design are: object-oriented, relational, predicate-based, or combinations of these. *Object-oriented* means that the basic items stored in the database can be accessed as abstract, semantic objects (in contrast to simple variables or strings). They can be related to each other within a class hierarchy (e.g. the class of *Escherichia coli* is a subset of the class of bacteria). Individual objects (e.g. a single *Escherichia coli* bacterium under the microscope) can inherit all characteristic class properties and methods from its class object. These properties are stored only once in the database but can be accessed through all related instances. Object-oriented databases are useful for storing knowledge about hierarchically structured domains (e.g. a taxonomy of enzymes or micro-organisms). *Predicate-based databases* store information in the form of facts (e.g. "Residues 7-17 of Crambin form a helix.") or rules (e.g. "A helix is defined by a certain number of consecutive turns."). This type of representation is convenient if an inference engine is linked to the database because then facts and rules can immediately be used for deductive inference. *Relational databases* can be visualised as a set of cross-indexed tables. They are easily maintained and allow fast access to a single item when information in the tables is kept sorted. However, if a complicated request needs to follow the links through several tables, this can slow down retrieval speed significantly. Also, expressing complex search constraints in terms of a relational database query language can sometimes become difficult or even impossible.

---

48 **Proceedings of the International Conference on Molecular Electronics and Biocomputing**, September 1994, Goa, India, Tata Institute of Fundamental Research, Ratna S. Phadke, email ratna@tifrvax.tifr.res.in.

49 F. Scheller, R. Schmid, *Biosensors: Fundamentals, Technologies and Applications*, in **GBF Monograph 17**, Verlag Chemie, Weinheim, 1992.

Biotechnological Applications
Biochips
Biosensors
Gel Reading Automata
Database
Object-Oriented Representation
Predicate Representation
Relational Paradigm
Processed / Selected Data
Medical Diagnosis Systems
DNA / RNA
Prediction of Gene Structure (Exon, Intron, Splice Site, Promoter, Enhancer)
Hydration and Environment
Genome Mapping
Secondary Structure Prediction
Sequence Analysis (Alignment, Search for Patterns)
Proteins
Classification according to Structure, Sequence or Function
De novo Design
Discovery of Structure / Function Relationships
Docking and Enzyme – Substrate Binding
Evolutionary Relationships
Folding Process and Motion
Force Fields and Energetics
Homology-based 3D Modelling
Inverse Folding Problem
Localising, Targeting and Signal Sequences of Membrane Proteins
Packing, Accessibility and Hydrophobicity
Prediction of Structure / Function Motifs by Multi-Criteria Comparison
Reconstruction of Backbone from Cα-Atoms
Reconstruction of Tertiary Structures from Backbone
Secondary Structure Prediction
Sequence Analysis (Alignment, Pattern Search)
Side Chains and Rotamers
Simulation of Metabolism
Solvent Interactions
Super-Secondary Structure and Hierarchy of Protein Structures
Tertiary Structure Prediction and Refinement
Toxicology
Spectrum Interpretation
Mass Spectra
Nuclear Magnetic Resonance Spectra
Planning of Experiments
Cloning
Protein Purification

Table 2: Selection of Applications in Molecular Bioinformatics.
This is a (certainly non-exhaustive) list of applications relevant in molecular bioinformatics.

DNA and RNA. Genomic information is stored in RNA and DNA. Although the basic nature of the genetic code was solved some 40 years ago[50,51], details of the organisation of chromosomes and genes are still not completely understood. Genes are known to have certain functional regions, e.g. exons, introns, splice sites, promoter sites and enhancer regions. *Exons* are DNA sequence segments that supply the information for protein formation. In eucaryotes, exons are often interrupted by non-coding portions which are called *introns* (from "intervening sequences"). Introns have to be excised to get a proper RNA copy of a gene. The boundaries of exons and introns are called *splice sites*. Reliable prediction of splice sites is desirable since this allows determination of the uninterrupted gene and the amino acid sequence of the corresponding protein (i.e. its *primary structure*). *Promoter sites* are regions in DNA that determine the start of transcription for a gene. *Enhancer elements* control the extent and speed of transcription. Important features of DNA are its *secondary structure* and interactions with a solvent. The mechanisms of the mentioned genetic elements can be described on a molecular level but these models are not yet accurate enough to allow confident prediction of gene structure in large quantities of unannotated sequences. *Sequence alignment* of DNA sequences is important for identifying homologous sequences and for searching related patterns. An immediate medical application is the development of antisense drugs that can specifically inhibit expression of malfunctioning genes. One problem with aligning sequences is the occurrence of insertions and deletions[52] which makes it difficult to determine the corresponding positions in either sequence; another problem is the exponential increase in complexity when comparing multiple sequences. *Genome mapping* is the localisation of DNA fragments, restriction sites and genes on a chromosome. This usually involves deriving a set of linear arrangements that satisfy a number of neighbourhood constraints. In practice, this is difficult because the information available is often incomplete, noisy and / or redundant. Complete exploration of all possible arrangements requires traversing a search space that increases exponentially with fragment number.

Proteins. The chromosomes of eucaryotes carry information on how to synthesise tens of thousands of different proteins. Proteins are multi-functional macromolecules, each a string made of 20 different amino acid residue types which are assembled by the ribosome and which spontaneously fold up into complicated conformations. They are used for a wide range of purposes: enzymatic catalysis of chemical reactions, transport and storage of chemical compounds, motion, mechanical support, immune protection, generation and transmission of nerve impulses, growth control and differentiation. The key to the function of a protein is its three-dimensional structure. The spatial arrangement of the atoms in the *active site* of an enzyme is made so that a substrate of roughly complementary geometry can bind to it and be subsequently modified in a chemical reaction. As was shown

50 J. D. Watson, F. H. C. Crick, *Genetic implications of the structure of deoxyribonucleic acid*, **Nature**, vol 171, pp. 964-967, 1953.

51 F. H. C. Crick, L. Barnett, S. Brenner, R. J. Watts-Tobin, *General nature of the genetic code for proteins*, **Nature**, vol 192, pp. 1227-1232, 1961.

52 Insertions and deletions are subsumed under the term "indels", as it is often impossible to decide whether an insertion in one sequence or a deletion in the other occurred.

in the Nobel-prize winning experiments by C. Anfinsen[53], the three-dimensional structure of a protein can be completely determined by its amino acid sequence. Although additional, non-spontaneous mechanisms of folding have been observed since then[54], the general view still holds that *sequence implies structure.*

The analysis of protein architecture includes comparison on the level of *primary structure* (i.e. the order of amino acids along the polypeptide chain), *secondary structure* (i.e. short stretches of amino acid residues in particularly regular conformation), and *tertiary structure* (i.e. the exact conformation of a whole protein). The so-called *protein folding problem* is to establish detailed rules defining the relationship between primary and tertiary structure. These rules could help predict conformations for the many known protein sequences of unknown structure and also for some newly invented ones. Biochemists could then reason about the function of those proteins on the basis of their predicted conformations. A general solution to this problem is not yet known but for a number of special cases algorithms for predicting secondary and tertiary structure have been developed. The reliable and accurate *prediction of secondary structure* is of interest as it allows the assembly of a whole protein in terms of almost rigid building blocks. This reduces the number of potential conformations by several orders of magnitude. Unfortunately, secondary structure prediction is rather difficult due to the effect of *long range interactions.* This means that identical short sequences of amino acids can adopt different secondary structures in different contexts, i.e. secondary structures are not purely locally defined by their sequence.

Other applications in molecular bioinformatics are the interpretation of NMR and mass spectra, and the planning of cloning and protein purification experiments.

53 C. B. Anfinsen, C. B. Haber, M. Sela, F. H. White, *The kinetics of the formation of native ribonuclease during oxidation of the reduced polypeptide chain,* **Proc. Natl. Acad. Sci. USA**, vol 47, no 9, pp. 1309-1314, 1961.

54 P. J. Kang, J. Ostermann, J. Shilling, W. Neupart, E. A. Craig, N. Pfanner, *Requirement for hsp70 in the mitochondrial matrix for translocation and folding of precursor proteins*, **Nature**, vol 348, pp. 137-143, 1990.

2. Artificial Intelligence & Expert Systems

This chapter concerns the application of expert systems and other techniques from artificial intelligence to various problems in bioscience. The methodologies described here clarify fundamental issues in symbolic information processing and illustrate their use. Section 2.1 elaborates on some key principles and algorithms in artificial intelligence programming. Section 2.2 then presents a number of applications using these techniques.

2.1. Methodology

One important feature of artificial intelligence programming and expert systems is *symbolic computation* as opposed to *numerical programming* and *string processing*. Symbolic computation is a prerequisite for implementing and manipulating knowledge bases. Typical representatives of symbolic programming languages are Lisp and Prolog. Here, we will take a brief look at programming techniques used with Lisp while programming in Prolog will be dealt with in a later chapter.

Lisp[1] is the second oldest programming language and emerged at about the same time as Fortran. During the 1970's and 80's, Lisp was run on dedicated hardware which were then rather expensive computers like the Symbolics™ Lisp machine. Today, fast Lisp interpreters are available for most common platforms at a reasonable price. However, there are two problems with Lisp. First, although there is a specification of Common Lisp[2], porting between different Lisp dialects tends to remain difficult[3]. Second, Lisp was designed to facilitate symbolic computation and not primarily numerical processing. Applications that require many floating point operations can be rather slow in Lisp. This problem can largely be overcome by interfacing Lisp to other languages, e.g. C or Assembler. Unlike C, Assembler or Pascal, Lisp allows rapid prototyping of rather complex systems through symbolic computation.

Lisp is traditionally an interpreted language which means that the user types in a command, e.g. an atom to be evaluated or a function to be called, and the result

1 J. McCarthy, Recursive Functions of Symbolic Expressions and their Computation by Machine, **Communications of the ACM**, vol 3, no 4, pp. 185-196, 1960.
2 G. L. Steele Jr., **Common LISP – The Language**, Digital Press, 1984.
3 S. Schulze-Kremer, *Common Lisp – ein geeigneter Lisp Standard?*, **Praxis der Informationsverarbeitung und Kommunikation**, vol 11, pp. 181-184, Carl Hanser Verlag, München, 1988.

appears on the screen without an explicit call of a compiler. This interactive way of programming encourages the developer to first build small components which are then used as building blocks to create more complex functions. Of course, all current Lisp systems also have compilers which are used to translate the whole system into a faster running application once it is debugged and tested. The use of an interpreted language may be helpful in keeping track of the incremental growth when developing an experimental system. Lisp is an abbreviation of list processing, or, misconstrued by those failing to appreciate its flavour, "lot's of insidious, silly parentheses".

2.1.1. Symbolic Computation

The basic data type in Lisp is a symbol, called an *atom*. The basic computational unit is a *function*. Symbols are e.g. 123.45, PROTEIN, DNA or RNA. The power of symbolic computation is based on the ease of processing semantic concepts of an underlying model instead of mere variables. Access and reference to symbols can reflect their properties, internal structure or their behaviour in response to external stimuli. Ideally, calling or evaluating a symbol displays the result of its interaction with other concepts. Properties and features of a concept can be stored in so-called *property lists*, *associative lists* or in functions. Atoms are kept in lists and a number of primitive functions are provided to process those lists. They can, for example, retrieve the first element (CAR) or the rest of a list (CDR) or they can define a new function (DEFUN). The user does not have to care about integer, double or character data types as in other programming languages. If the first atom in a list is a function name, the remaining atoms or lists that follow are treated as arguments. Atoms and lists are summarised in the term *symbolic expression*. A Lisp interpreter is the top-level Lisp function which receives input from the keyboard and then tries to evaluate a symbolic expression. If that is an alphanumeric atom, its global value is returned. If it is a numerical atom (i.e. an ordinary number), the number itself is returned. If it is a list, the function denoted by the first atom is called with all following symbolic expressions as its arguments. The arguments are themselves evaluated before being passed over to the calling function. Example 1 illustrates some basic Lisp functions.

The most intriguing fact about Lisp (and also Prolog) is that *data and program are indistinguishable*. Lists hold atoms and other lists for storage of data. At the same time, any list can be interpreted as the definition of a function or as a function call. From this follows that in Lisp *new programs* can be automatically generated and immediately executed during run time. Those programs may then, in turn, generate other programs or modify their parent programs, and so on. This capability allows the implementation of flexible, self-modifying programs that evolve and potentially learn or improve during run time. This capability of Lisp was recently used in the context of genetic programming[4]. An example for the run-time construction of a (simple arithmetic) program and its execution is given in Example 2 where a function call to add 4 numbers is created during run-time and evaluated.

4 J. Koza, **Genetic Programming (I, II)**, MIT Press, Cambridge MA, 1993, 1994.

```
Input                                  Result
-------------------------------------------------------
⇒ (+ 1 2 3)                            6
⇒ (* (+ (- (/ 55 11) 10) 9) 8)         32
⇒ (CAR '(PROTEIN DNA))                 PROTEIN
⇒ (CDR '(PROTEIN DNA RNA))             (DNA RNA)
⇒ (CADADR '(PROTEIN (DNA RNA)))        RNA
⇒ (CONS 'PROTEIN '(DNA RNA))           (PROTEIN DNA RNA)
⇒ (LIST 'PROTEIN 'DNA 'RNA)            (PROTEIN DNA RNA)
```

Example 1: Basic Lisp Functions.
Input to the Lisp interpreter on the left side produces output on the right. The first line is a function call to add three numbers. The second line is a more complicated arithmetic expression. PROTEIN, DNA and RNA are alphanumeric symbols. The function CAR returns the first element of a list, the function CDR returns the rest. CAR and CDR can be nested up to four levels in one word, where the sequence of A's and D's *read from the end of the word* identifies the order of CAR and CDR evaluation. CONS pushes a symbolic expression into a list at the first position. LIST creates a list of an arbitrary number of arguments. Most Lisp interpreters make no distinction between upper and lower case letters except when they are enclosed in double quotes.

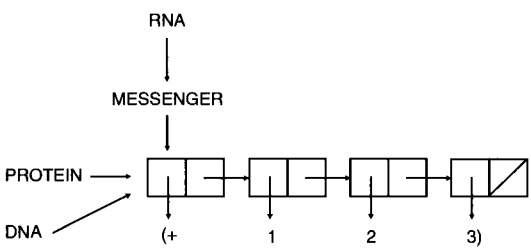


Figure 1: Lisp Pointer Structure.
This figure explains the pointer structure underlying the code of Example 2. RNA points to MESSENGER (both global symbols). A CONS cell has two pointers, one to the first element of a list, another to the beginning of the rest of the list. MESSENGER, PROTEIN and DNA point to a CONS cell, that points to the "+" sign and yet another CONS cell. Three more CONS cells define the rest of the list (+ 1 2 3).

In Example 2, the first line binds the list (+ 1 2 3) to the symbol PROTEIN. This is the standard way of defining a global variable. The second line binds *the value* of PROTEIN (that's why there is no quotation mark before PROTEIN) to the symbol DNA. The CAR (first element) of *the value* of DNA is the symbol "+". Now, the symbol RNA gets as its new value the symbol MESSENGER, which is duly echoed by the Lisp interpreter. Then, *the value* of RNA, which at this time is the symbol MESSENGER, is given *the value* of DNA. By evaluating MESSENGER in the next line we see the Lisp interpreter confirm that operation. The function EVAL does one extra evaluation. First, all arguments are evaluated before being passed to EVAL, then EVAL evaluates them once more. (EVAL 'RNA) returns MESSENGER, *as this is the value of*

```
Input                                  Result
-------------------------------------------------
⇒ (SET 'PROTEIN '(+ 1 2 3))             (+ 1 2 3)
⇒ (SET 'DNA PROTEIN)                    (+ 1 2 3)
⇒ (CAR DNA)                             +
⇒ (SET 'RNA 'MESSENGER)                 MESSENGER
⇒ (SET RNA DNA)                         (+ 1 2 3)
⇒  MESSENGER                            (+ 1 2 3)
⇒ (EVAL 'RNA)                           MESSENGER
⇒  RNA                                  MESSENGER
⇒ (EVAL RNA)                            (+ 1 2 3)
⇒ (EVAL 'MESSENGER)                     (+ 1 2 3)
⇒ (EVAL MESSENGER)                      6
⇒ (EVAL
    (CONS (CAR PROTEIN)
     (LIST (CADR PROTEIN)
           (* 4 9)
           (CADDDR PROTEIN)
           (CADDR PROTEIN))))           42
```

Example 2: Simple Automatic Program Generation in Lisp.
Input to the Lisp interpreter on the left side produces output on the right. See main text for explanation.

RNA. The quotation mark in front of RNA neutralises the first evaluation of EVAL. This is the same as if we gave the symbol RNA to the Lisp interpreter. EVALuating RNA without a quotation mark produces the list (+ 1 2 3). EVALuating MESSENGER quoted gives the same list, but for MESSENGER unquoted the list itself is evaluated once more thereby adding 1+2+3. Finally, instead of directly typing in a short list for adding four numbers, we use EVAL to assemble the function call (+ 1 36 3 2) via CONS, LIST, the CAR/CDR functions and the value of PROTEIN. After construction of that mini-program it is evaluated by EVAL and the result resturned by the Lisp interpreter. Similarly, more complex programs can be written if nested function calls with different functions are assembled (see also Example 5).

Recursive programming is easily done within Lisp. Many search algorithms can be formulated in a recursive manner. The general approach to define a recursive function is as follows. First, specify the terminating clause. This is where recursions end and a default value is returned. Then, the case with only one element left in the argument list is handled. That element is processed and its result returned. Finally, the recursive clause appears. If a composite structure is seen, it is decomposed into the head and the rest of that structure. Each part is then passed to the original function. The results of those recursive calls are then joined taking care to maintain their order as in the original call. The definition of a recursive function FLATTEN is given in Example 3. FLATTEN extracts all atoms from within an arbitrarily nested list. More on recursion in Lisp is charmingly presented elsewhere[5].

The function FLATTEN has one argument, locally named *arg*. Uppercase words denote predefined Lisp functions. COND is a conditional function which resembles

5 D. P. Friedman, M. Felleisen, **The Little LISPer**, MIT Press, 1987.

```
(DEFUN flatten (arg)
   (COND ((NULL arg) NIL)
         ((ATOM arg) (LIST arg))
         (T (APPEND (flatten (CAR arg))
                    (flatten (CDR arg))))))
```

Example 3: Recursion in Lisp.
This is the definition of FLATTEN. See main text for details.

a series of IF-THEN pairs. Each COND clause starts with a condition, here whether the argument is an empty list: (NULL *arg*). If this is the case, the following value is returned as the value of COND, in this case the empty list NIL. The last COND clause starts with T (for TRUE). This is the default clause to be evaluated, if none of the prior two clauses had been selected. FLATTEN returns NIL (equivalent to an empty list) if its argument is an empty list. If the argument is an atom, a list containing that atom is returned. Finally, first the head, then the rest of the argument is processed in another call to FLATTEN and the results then merged into one list (via APPEND).

A flavour of symbolic manipulation of biological cellular components is given in Example 4. A cell is simplified as a list containing the Lisp atom PROTEIN and a sublist (NUCLEUS (DNA)) which itself contains the Lisp atom NUCLEUS and the list (DNA). Symbolically, all objects are to be extracted from their cellular compartments into one list using FLATTEN. This crude model can be easily refined by using more nested compartments to represent intracellular organisation more realistically. Also, more elaborate versions of FLATTEN can be thought of which take into account the principle and applicability of different separation methods. Such a program could then be used to simulate a sequence of purification steps on a particular specimen. The result would show how well the desired component was separated from other constituents and whether it could be preserved in a functional state.

The final example here for symbolic computation is algebraic differentiation[6]. There is surprisingly little code required to implement differentiation of basic algebraic expressions. Example 5 shows the complete Lisp program. Again, the problem can be solved recursively. The function DIFFERENTIATE simplifies an expression by dividing one complex term into a number of simpler expressions. Only the most basic ones are explicitly differentiated. DIFFERENTIATE gets two arguments: first, the expression to be differentiated (argument E), and second, the reference variable *dx* (argument X). At the beginning, if the expression is a constant or a single instance of the variable to be differentiated (i.e. E is a Lisp atomic expression, not a list, in which case ATOM returns NON-NIL), the value 0 or 1 is returned, respectively. The next COND clause examines additive terms. If the operator is either "+" or "-", a list is returned the first element of which is the original top-level operator. The following two expressions recursively differentiate the two terms to be summed (or subtracted) and substitute their differentials into the overall result. LIST does two things here. First, it collects three items to be returned in a list, and second,

6 P. H. Winston, B. K. P. Horn, **LISP**, Addison-Wesley, 1984.

```
⇒ (SET 'CELL '(PROTEIN (NUCLEUS (DNA))))
⇐ (PROTEIN (NUCLEUS (DNA))))
⇒ (FLATTEN CELL)
  ENTERING: FLATTEN, ARGUMENT LIST: ((PROTEIN (NUCLEUS (DNA))))
   ENTERING: FLATTEN, ARGUMENT LIST: (PROTEIN)
   EXITING: FLATTEN, VALUE: (PROTEIN)
   ENTERING: FLATTEN, ARGUMENT LIST: (((NUCLEUS (DNA))))
    ENTERING: FLATTEN, ARGUMENT LIST: ((NUCLEUS (DNA)))
     ENTERING: FLATTEN, ARGUMENT LIST: (NUCLEUS)
     EXITING: FLATTEN, VALUE: (NUCLEUS)
     ENTERING: FLATTEN, ARGUMENT LIST: (((DNA)))
      ENTERING: FLATTEN, ARGUMENT LIST: ((DNA))
       ENTERING: FLATTEN, ARGUMENT LIST: (DNA)
       EXITING: FLATTEN, VALUE: (DNA)
      EXITING: FLATTEN, VALUE: (DNA)
     EXITING: FLATTEN, VALUE: (DNA)
    EXITING: FLATTEN, VALUE: (NUCLEUS DNA)
   EXITING: FLATTEN, VALUE: (NUCLEUS DNA)
  EXITING: FLATTEN, VALUE: (PROTEIN NUCLEUS DNA)
⇐ (PROTEIN NUCLEUS DNA)
```

Example 4: Symbolic Extraction of Cellular Components.
The function FLATTEN is called with the nested list (PROTEIN (NUCLEUS (DNA))). Recursive calls are traced, showing input and output. Calls processed by the first COND clause of FLATTEN return NIL and have been omitted for brevity. The signs "⇒" and "⇐" denote input and output of the Lisp interpreter.

it initiates the differentiation of the two terms. Remember, every argument is evaluated *before* being included by LIST. For this purpose, (OPERATOR E) returns the original operator. The next COND clause handles the case of multiplication. The two terms being multiplied are recursively treated according to the differentiation rule. Finally, if found in E division and exponential expressions are processed. This version of a symbolic differentiator does not simplify its result. Example 6 shows some applications of DIFFERENTIATE.

Recommended reading on symbolic programming in Lisp is P. Winston's and B. Horn's book LISP[6].

2.1.2. Knowledge Representation

When thinking about problem solving in general and with the human model in mind in particular, the first two questions are:

1) What is the problem?
2) How to get to a solution?

The first question involves the description of the problem or goal; the second refers to the algorithm to be applied. This section addresses the first question, i.e. ways to represent a problem in computer tractable format. The following section then

$$
\begin{aligned}
\frac{d}{dx}c &= 0, \quad \frac{d}{dx}x = 1 \\
\frac{d}{dx}(u \pm v) &= \frac{d}{dx}u \pm \frac{d}{dx}v \\
\frac{d}{dx}(u \cdot v) &= u\frac{d}{dx}v + v\frac{d}{dx}u \\
\frac{d}{dx}\frac{d}{v} &= \frac{d}{dx}uv^{-1} \\
\frac{d}{dx}u^n &= nu^{n-1}\frac{d}{dx}u
\end{aligned}
$$

```
(DEFUN DIFFERENTIATE (E X)
  (COND
    ((ATOM E) (COND ((EQUAL E X ) 1)
                    (T 0)))
    ((OR (EQUAL (OPERATOR E) '+)
         (EQUAL (OPERATOR E) '-))
     (LIST (OPERATOR E) (DIFFERENTIATE (ARG1 E) X)
                        (DIFFERENTIATE (ARG2 E) X)))
    ((EQUAL (OPERATOR E) '*)
     (LIST '+ (LIST '* (ARG1 E) (DIFFERENTIATE (ARG2 E) X))
              (LIST '* (ARG2 E) (DIFFERENTIATE (ARG1 E) X))))
    ((EQUAL (OPERATOR E) '/)
     (DIFFERENTIATE (LIST '* (ARG1 E)
                             (LIST 'EXPT (ARG2 E) -1)) X))
    ((EQUAL (OPERATOR E) 'EXPT)
     (LIST '* (ARG2 E)
              (LIST '* (LIST 'EXPT (ARG1 E) (- (ARG2 E) 1))
              (DIFFERENTIATE (ARG1 E) X))))))

(DEFUN ARG1 (E) (CADR E))
(DEFUN ARG2 (E) (CADDR E))
(DEFUN OPERATOR (E) (CAR E))
```

Example 5: Symbolic Differentiation.
The formulas at the top are the standard differentiation rules. Below, a program in Lisp incorporating these rules is shown. DEFUN is used to define new Lisp functions. The functions ARG1, ARG2 and OPERATOR return the first and second arguments and the operator of a prefix algebraic expression, respectively. They are identical to the predefined Lisp functions CADR, CADDR and CAR and have been defined here only for improved readability of the program. In Lisp, algebraic expressions are written in prefix notation, e.g. 7 / (4 - X) as (/ 7 (- 4 X)). See text for an explanation of the program.

describes some fundamental algorithms how to process (symbolically represented) knowledge.

The reason why knowledge representation is so important is obvious. A scientific article from, say, the Journal of Molecular Biology, can be stored in several ways in a computer. The simplest way is to type in the text using an ordinary word processor and redrawing the diagrams and figures. If a scanner and good character recogni-

```
⇒ (DIFFERENTIATE '(+ X (* X X )) 'X))
⇐ (+ 1 (+ (* X 1)
          (* X 1)))

⇒ (DIFFERENTIATE '(EXPT X 4) 'X)
⇐ (* 4 (* (EXPT X 3) 1))

⇒ (DIFFERENTIATE '(* X (EXPT X 4)) 'X)
⇐ (+ (* X (* 4 (* (EXPT X 3) 1)))
     (* (EXPT X 4) 1))

⇒ (DIFFERENTIATE '(/ X (EXPT X 4)) 'X)
⇐ (+ (* X (* -1 (* (EXPT (EXPT X 4)
                         -2)
                   (* 4 (* (EXPT X 3) 1)))))
     (* (EXPT (EXPT X 4) -1) 1))
```

Example 6: Symbolic Differentiation.
Calls of the function DIFFERENTIATE. The algebraic expressions to be differentiated are $x+x^2$, x^4, $x{\cdot}x^4$, and x/x^4. The two later ones could be simplified but were kept in this form to illustrate the combined use of several COND clauses in DIFFERENTIATE. The lines preceded by "⇒" are input to the Lisp interpreter, results of DIFFERENTIATE are marked "⇐". Output has been indented for clarity.

tion software are available, one can even avoid the typing. Scanned diagrams can be saved as pixel files. Thus, the complete information from the article is in the computer in form of character strings and pixel graphics. The capabilities to operate on information stored this way are rather limited. One can perform text search for keywords or phrases and one can display the diagrams on the screen or printer. One can not, for example, ask the computer "What is the connection between drug X and the symptoms Y?" even if those items were discussed in the paper. Or one cannot ask "What is the standard deviation of the data points in diagram Z?" if only diagram Z is given but no explicit calculation of that standard deviation.

In order to answer these and similar questions, it is necessary to represent the *meaning* of the text and the diagrams in the computer[7,8]. Humans use language to represent meaning and so must computers. This is where knowledge representation enters the scene. A number of knowledge representation formalisms have already been developed and the most important ones are listed in Table 1. They define basic "chunks" of knowledge and how these can be manipulated by an algorithm. A human expert and/or special software is needed to transform raw data, i.e. text and diagrams of the article into objects, rules, predicates or other knowledge representation formalisms. This process is called *knowledge engineering* and it

7 R. P. Futrelle, *Steps toward a scientist's assistant – reading the literature*, **Working notes Symposium Artificial Intelligence and Molecular Biology**, AAAI, Stanford University CA, pp. 46-50, 1990.

8 K. Baclawski, R. P. Futrelle, N. Fridman, M. J. Pescitelli, *Database techniques for biological materials and methods*, **Proceedings First International Conference on Intelligent Systems for Molecular Biology**, AAAI Press, Menlo Park CA, pp. 21-28, 1993.

Representation Formalism	Unit
functional	functions, procedures
semantic nets, graphs, trees	nodes, edges
object-oriented	objects, frames
predicate calculus	predicates
rules	conditions, conclusions, actions
scripts	events, tasks, processes

Table 1: Knowledge Representation Formalisms.
Some fundamental ways of how to organise and implement knowledge. "Unit" refers to the basic entity used to build more complex models.

requires both textual analysis and pattern recognition in pictures. Scientific articles are relatively easy to transform into computer tractable knowledge representations as their language is restricted and scientists try to be factual and to reason logically. Mathematical diagrams come only in a limited number of variations. The difficulty with knowledge engineering is to grasp all meaning from the data presented and transform it into one of the mentioned representation formalisms in an as much as possible automated way. If this can be done inexpensively, efficiently and accurately, the next generation of information systems will become available that will not only have the enormous capacity of whole libraries but that can also use this data for automatic reasoning and inference.

Functional Representation. Functions can be used to describe a computational model[9]. A mathematical function defines a correspondence between one variable (the result) and some others (the arguments). Functions provide a natural way to describe transformation or modification of data. Some of the most popular programming languages are function-oriented (C, Pascal). As was demonstrated in the previous section, Lisp is also a functional language. The advantage of functional programming is the modularisation of a problem into smaller pieces which then can be called from other functions to carry out more complex operations. Individual functions can be traced in isolation which facilitates "debugging" and verification of large programs. However, purely functional programming has its limits. The declaration and definition of variables, input/output to files or to the screen, memory management, program flow control and a non-volatile memory between successive calls of a function are not (mathematical) functions themselves. These actions are rather considered to be *side effects* of a function. Functions whose main purposes are such side effects are more precisely termed *procedures*. Relations between objects, as e.g. *super-class*, *type-of* or *instance-of*, have no functional equivalent and are therefore difficult to represent in a uniform and concise way in a functional language.

One example for the application of a functional representation is reasoning about biological theories (Figure 2). D. Moberg and L. Darden implemented part of the Mendelian genetic theory into a compiler that can transform a functional repre-

9 C. Reade, **Elements of Functional Programming,** Addison-Wesley, 1986.

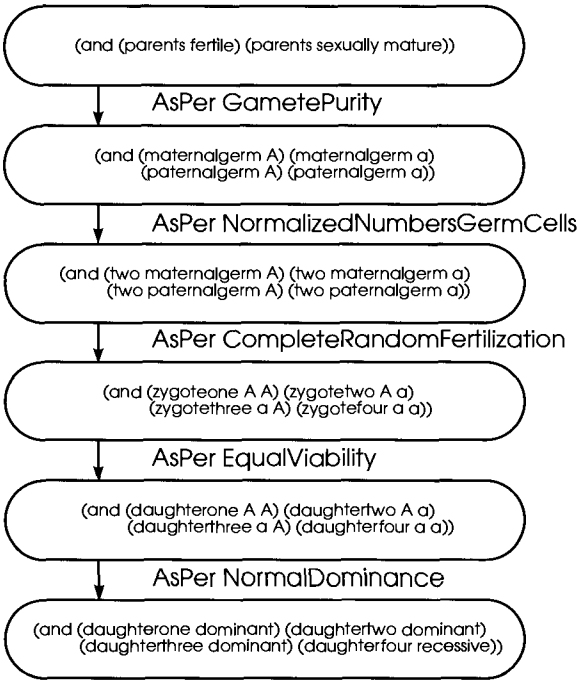


Figure 2: Mendelian Theory as Functional System.
Input to functions is printed inside boxes. Each argument is a conjunction of facts in Lisp syntax. A function below a box receives the argument from above and subsequently returns its value below. Five theoretical components of Mendelian theory have been specified this way.

sentation of biological concepts into a diagnostic classification hierarchy which can then, based on the observation of an anomalous inheritance pattern, suggest likely inconsistencies in the original assumptions[10].

Semantic nets. Originally, semantic nets were designed to represent the meaning of words and phrases in English texts[11]. In a semantic net, information is represented as a set of nodes connected to each other by a set of labelled arcs which indicate relationships among nodes. Nodes can be objects, abstract concepts or qualities. In most programming languages graphs of semantic nets can be coded as object-attribute-value triplets. Figure 3 shows a semantic net representing the central dogma of molecular biology in connection with *E. coli* bacteria. Example 7 shows the corresponding object-attribute-value representation of Figure 3 in Lisp syntax.

10 D. Moberg, L. Darden, *Representing and reasoning about biological theories using the functional representation system*, **Working notes Symposium Artificial Intelligence and Molecular Biology**, AAAI, Stanford University CA, pp. 95-96, 1990.

11 R. Quillian, *Semantic Memory*, in **Semantic Information Processing**, (M. Minsky Ed.), MIT Press, Cambridge MA, 1968.

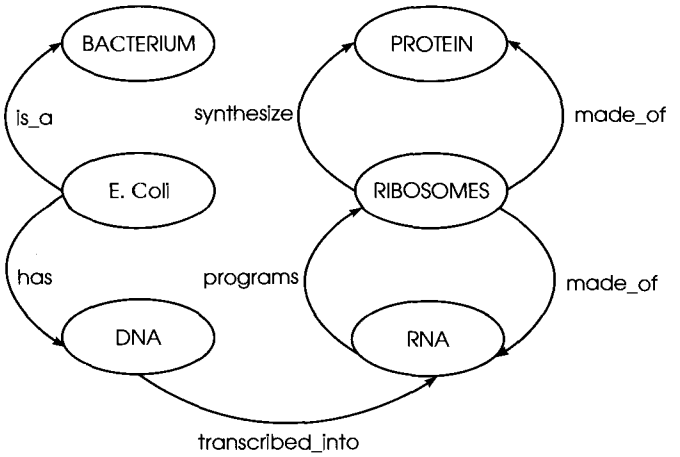


Figure 3: Semantic Net for the Central Dogma of Molecular Biology.
The central dogma of molecular biology states that DNA is transcribed into RNA which in turn serves as a template for protein synthesis. Emphasis is on the fact that information flows *from* DNA *to* protein and never the other way around.

```
Atom        Property list
-----------------------------------
E.COLI      ((IS_A BACTERIUM)
             (HAS DNA))
DNA         ((TRANSCRIBED_INTO RNA))
RNA         ((PROGRAMS RIBOSOMES))
RIBOSOMES   ((MADE_OF RNA)
             (MADE_OF PROTEIN)
             (SYNTHESIZE PROTEIN))
```

Example 7: Lisp Representation of a Semantic Net.
Central dogma of molecular biology in object-attribute-value representation. All objects that have at least one arc leaving their node in Figure 3 have a property list which contains sublists with a property (the relationship defined by an arc) and the node to which it is connected. Nodes without leaving arcs have no properties recorded. Note that arcs in semantic nets are unidirectional.

Arcs in semantic nets are unidirectional. If in the previous example one wants to know what other objects are made of RNA another arc would have to be drawn from RNA to RIBOSOMES and labelled "OCCURS_IN". This would then show up in the Lisp representation as an additional entry "RNA ((OCCURS_IN RIBOSOMES))".

There is a difference between concepts in general and a particular instance of a concept. In Figure 3 all nodes are general concepts. One particular *E. coli* cell observed under the microscope is an instance of the class of *E. coli* bacteria and can be given a unique name, e.g. MY_E.COLI or E.COLI3548. This can be represented by an arc INSTANCE_OF from a node representing the individual object MY_E.COLI to the class concept E.COLI. An instance inherits all properties of its class concept and also of all superclasses of that class. The distinction between class concepts

and individual concepts allows contradicting statements to be made about objects of the same type, e.g. about the DNA molecule in a particular cell and another one suspended in buffer in a test tube. Both instances of DNA share the general features of the class concept DNA but may actually differ in sequence, base ratio or other features. *Partitioned semantic nets*, where parts of a graph are encapsulated to define the scope of variables, can be used to represent statements like "Every *E. coli* bacterium in this culture carries a tetracycline resistance gene on vector pBR322". Semantic nets are useful for reasoning about the connection between two concepts within a model. All one has to do is find the way or ways to traverse from one node to another. The connecting path can be translated into simple object-attribute-value triplets to describe the relationship between two concepts. For small models, this can conveniently be visualised in node/arc diagrams as shown above. Trees are restricted graphs where each node can have only one incoming arc. Mathematical graph theory provides the basis for syntactically correct transformations of graphs.

Object-Oriented Representation. The basic unit in object-oriented representation is, as the name suggests, an object. As with semantic nets, class objects and instances of class objects are distinguished. A class can be derived from one or more superclasses and it can have a number of subclasses. Inheritance follows down the class hierarchy. For example, Crambin (class) is a protein (superclass) that is composed of 46 amino acid residues (other classes), which is amphiphilic (class property) and which has the secondary structures α-helix, β-strand and turn (subclasses). Each subclass in turn can be described through properties and/or other constituent objects. Other than semantic nets, objects can also have procedural attachments, so-called *methods*. The basic mechanism of interaction between objects is *message passing*. If an object receives a message that contains the selector to one of that object's (possibly inherited) methods, it evaluates that method which may induce a change in the object's internal state and/or produce messages to other objects. There are four main advantages of using an object-oriented representation. First, the constraint of having to distinguish between class objects and individual instances helps to develop a properly structured model. Second, local definition of an object and inheritance of class variables is both safe[12] and economic. Third, object-oriented representation is highly modular and transparent. This allows easy modification or substitution of single components of a model. Implementation details are hidden in order to not obscure the view of the model. Fourth, the mechanism of message passing allows natural simulation of interaction between independent or interdependent objects. A complex model can be described by specifying its basic building blocks and their specific patterns of interaction. C++ is a modern programming language[13] that was designed for object-oriented programming. Original implementations of an object-oriented programming environment were written in Lisp and called flavour systems, because one could mix the "flavour"

12 One has to think of how to resolve circular inheritance and multiple inheritance, where different properties with identical names have to be distinguished. Object-oriented programming languages (e.g. C++) provide mechanisms to preserve consistency in such cases.

13 M. A. Ellis, B. Stroustrup, **The Annotated C++ Reference Manual**, Addison-Wesley, 1990.

of different objects together[14]. Example 8 illustrates the object-oriented representation formalism for some facts on proteins. The slot-filler structure for an object description was originally developed for frames[15]. A number of applications have been using object-oriented representation formalisms recently: modelling genetics and expression of *E. coli*[16]; processing of fluorescence-activated cell sorting data[17] and molecular pathology[18].

Reasoning about the physical world involves reasoning about the kinds of changes that occur and their consequences. Reasoning about the dynamics of a system using scripts and object-oriented representation encounters the so-called *frame-problem*[19]. How can one tell which facts change from one instant to the next and which don't? In situational calculus, situations are used to model the world at different instants of time. Scripts[20] are frames that contain a (stereotyped) sequence of events to describe a process. Using situational calculus to represent the changing states of the world requires writing explicit axioms that describe which things change and which things remain the same. The number of axioms rises as the product of the number of predicates and actions. Thus, adding a new action potentially requires adding a large number of new axioms. By contrast, histories[21] are descriptions of objects that are extended through time but always spatially bounded. Histories help to solve the frame-problem because objects can interact only when their histories intersect. This creates two new problems. First, "How are histories generated?", and second, "Which intersections between histories actually correspond to interactions between the objects?" To answer these questions a formal theory of (physical) processes is needed. The foundations of the so-called *qualitative process theory*[22] provide a framework for representing interactions of objects over time. Using this formalism the following types of reasoning on the state of processes can be performed: checking the activity of a single process; predicting future events; determining which events led to a particular situation; checking consistency of a physical situation; interpreting measurements; planning experiments to gain more

14 D. Moon, D. Weinreb, **Flavors: Message Passing in the Lisp Machine**, MIT-AIM-602, Cambridge, MA, 1980.

15 M. Minsky, *A Framework for Representing Knowledge*, in **The Psychology of Computer Vision**, (P. Winston Ed.), MacGraw-Hill, New York, 1975.

16 G. Perriere, F. Dorkeld, F. Rechenmann, C. Gautier, *Object-oriented knowledge bases for the analysis of prokaryotic and eukaryotic genomes*, **Proceedings First International Conference on Intelligent Systems for Molecular Biology**, AAAI Press, Menlo Park CA, pp. 319-327, 1993.

17 T. Matsushima, *Constructing a distributed object-oriented system with logical constraints for fluorescence-activated cell sorting*, **Proceedings First International Conference on Intelligent Systems for Molecular Biology**, AAAI Press, Menlo Park CA, pp. 266-274, 1993.

18 S. S. Ball, V. H. Mah, *Senex: a CLOS / CLIM application for molecular pathology*, **Proceedings First International Conference on Intelligent Systems for Molecular Biology**, AAAI Press, Menlo Park CA, pp. 39-46, 1993.

19 J. McCarthy, P. J. Hayes, *Some philosophical problems from the standpoint of artificial intelligence*, **Machine Intelligence**, (B. Metzner, D. Michie Eds.), vol 4, Edinburgh University Press, Edinburgh, 1969.

20 R. C. Schank, R. P. Abelson, **Scripts, Plans, Goals and Understanding**, Erlbaum Hillsdale, New Jersey, 1977.

21 P. J. Hayes, *The naive physics manifesto*, in **Expert Systems in the Microelectronic Age**, (D. Michie Ed.), Edinburgh University Press, Edinburgh, 1979.

22 K. D. Forbus, *Qualitative Process Theory*, **Artificial Intelligence**, vol 24, pp. 85-168, 1984.

CLASS:	Protein	INSTANCE:	Crambin
SUPERCLASS:	Peptide	CLASS:	Protein
SUBCLASS:	Enzyme	VARIABLES:	Size = 46 residues
VARIABLES:	Size		Temp-Range = 45°
	Temp-Range		Sequence = (TTCC ...)
	Sequence		
METHODS:	Net-Stoichiometry ()		
	Active-at (Temp)		
	Digest (Protease)		
CLASS:	Amino-Acid	INSTANCE:	Glycin
SUPERCLASS:	Organic-Compound	CLASS:	Amino-Acid
SUBCLASS:	Ala, Arg, ..., Val.	VARIABLES:	Structure = CH_2NH_2COOH
VARIABLES:	Structure		Side-Chain-pK = N.A.
	Side-Chain-pK		
METHODS:	Mol-Weight ()		
	Form-Peptide-Bond (AA)		

Example 8: Object-Oriented Representation of Crambin and Glycin.
Four object definitions are given, two class objects (PROTEIN, AMINO-ACID) and two individual objects (CRAMBIN, GLYCIN). Strictly speaking, CRAMBIN and GLYCIN refer only to one single molecule of each species. Class variables are once declared in a class and may in each derived instance locally be assigned an individual value. Glycin has no side chain pK since it has no side chain. Methods are declared within classes and hold general procedures that can be used by related instances. NET-STOICHIOMETRY( ) and MOL-WEIGHT( ) could be functions to sum the atomic weights of a compound based on its sequence and structure information, respectively. Empty brackets after a function indicate that it has no arguments. It may access class variables, though. ACTIVE-AT (TEMP) could be a function which returns whether that protein is still functional at a certain temperature TEMP (based on the instance's value of TEMP-RANGE). DIGEST (PROTEASE) could return the fragments for this protein upon digestion with a particular protease of known sequence specificity.

knowledge on a situation; and, finally, causal reasoning (in contrast to reasoning based on constraints, e.g. conservation laws in physics).

Rules. Rules were the first representation formalism used for so-called *expert systems*. Originally, expert systems in artificial intelligence were meant to refer to computer programs that are "experts" on a particular domain. However, the definition of expert and that of domain can be extended to label almost any program an expert system, e.g. including basic accountancy software. This may be counter-intuitive as one connotation of (human) expertise is that he or she is clearly above average among professionals in that domain. In contrast, for becoming an expert system a program had only to outperform a human in one specialised task that would take more time to be done manually than by using that program. Comparison of an expert system to the whole range of mental capabilities of human experts was never much an issue. Unfortunately, this discrepancy in the conception of expert systems gave raise to exaggerated hopes in expert system technology and later produced much frustration. Nevertheless, in a number of cases expert systems have proved

to be valuable and economical tools for expert amplification, not expert replacement. In this sense only, the notion of expert systems is advocated here.

The use of an expert system is recommended if all or most of the following selection criteria are meet. 1) There is a rather narrow problem domain in which only few recognised experts master the essentials of that domain. 2) The problem solving task should be combinatorial and primarily of a cognitive type. 3) The task may take hours to days to be completed by an expert. 4) The task should be decomposable into subgoals which can be treated more or less independently. 5) The knowledge involved should be transmissible in a formalised representation. 6) There should be experts willing to share their expertise based on case studies and other data. – As an expert system takes some time to be up and running it is important to incrementally work one's way up from relatively easy to hard problems. If successful, this creates a positive atmosphere about the expert system project in contrast to the risk of failure if one cannot fulfil expectations that were set too high at the beginning. Finally, the desired expert system should have a reasonable payoff.

A rule in its simplest form is "IF *condition* THEN *action*". The expression for *condition* can be any statement or combination of statements that if seen to occur at run time initiate *action*. The expression *action* can perform a transformation of the current state of the problem. This is called a production system[23]. One problem with production systems is conflict resolution: What shall be done if more than one rule is applicable? This can not be answered in general but has to be decided in view of the respective application. In another version, *action* can induce a change in the environment or can be simply a logical conclusion. The latter type is used for theorem provers[24], where the goal is to deductively prove a fact or an implication based on a number of hypotheses (rules and background knowledge). One extension to simple rules are *probabilities* and *certainty factors*. These numbers can be used to measure the confidence in or certainty of a precondition and the strength of a conclusion. This is difficult first, because the certainty rating of a possibly very large number of rules (which may have been crafted by different individuals) has to be consistent. Second, there is often not enough data to calculate proper probabilities for attribute-class relationships[23]. This is the case, for example, in medical diagnosis systems. Ideally, one would like to know the conditional probability of a disease depending on a symptom. This kind of data is difficult to measure directly because there are no statistics listing patients by symptoms rather than by disease. Fortunately, in this case Bayes's theorem can be used to calculate the missing probabilities:

$$P(d \mid s) = \frac{P(d) \cdot P(s \mid d)}{P(s)},$$

where $P(d \mid s)$ is the conditional probability of a disease d given symptom s. $P(s \mid d)$ is the conditional probability of a symptom s given a disease d. The latter numbers

23 E. Charniak, D. McDermott, **Introduction to Artificial Intelligence**, Addison-Wesley, 1985.

24 H. Gelernter, *Realization of a geometry theorem-proving machine*, in **Information Processing: Proceedings of the International Conference on Information Processing**, UNESCO, 1959.

are available. $P(s)$ is the general probability of the symptom s, $P(d)$ the general probability of disease d. For m diseases and n symptoms approximately $m{\cdot}n$ numbers are needed:

$$\begin{aligned}(m\cdot n \quad &\text{conditional probabilities})+\\ (m \quad &\text{disease probabilities})+\\ (n \quad &\text{symptom probabilites}) \quad = \quad m\cdot n+m+n\approx m\cdot n\end{aligned}$$

In the simplest case of multiple symptoms, the equation is extended to include the conjunction of two symptoms:

$$P(d \mid s_i \wedge s_j) = \frac{P(d)\cdot P(s_i \wedge s_j \mid d)}{P(s_i \wedge s_j)}.$$

Now, for n symptoms the probabilities $P(si \wedge sj \mid d)$ and $P(si \wedge sj)$ must be known. There are $2\sum_{i\neq j}^{n} pair_{ij} = 2\cdot\frac{n(n-1)}{2} = n(n-1)\approx n^2$ pairs of symptoms which increases the number of probabilities to approximately $m{\cdot}n^2$. In applications, where n and m are in the range of several hundreds, this product gets prohibitively large (this for the case of only *two* symptoms considered simultaneously). The combination of, say, 40 symptoms makes it even unlikely to find a statistically significant number of patients that exhibit exactly those combinations of symptoms. To improve this situation, one can make the assumption that two symptoms are statistically independent in general and therefore also especially among people suffering from a particular disease. This assumption is invalid because many symptoms are caused by the same disease and therefore do not indicate independent, different events. However, the assumption of independent symptoms simplifies the previous equation to give:

$$P(d \mid s_i \wedge s_j) = \frac{P(d)\cdot P(s_i \mid d)\cdot P(s_j \mid d)}{P(s_i)\cdot P(s_j)}.$$

For the two symptom-case are now only approximately $m{\cdot}n$ numbers required, the same as in the case with only one symptom. It is convenient to use the logarithm of that probability because then one can tabulate normalised I factors and has only to add the I factors of other symptoms to get the overall probability of a disease. A list of those diseases with the highest probability is then presented to the physician for further examination. Note that this calculation also ignores multiple diseases.

$$\log P(d \mid s_i \wedge s_j) = \log P(d) + \log I(d \mid s_i) + \log I(d \mid s_j) \text{ with } I(d \mid s) \equiv \frac{P(s \mid d)}{P(s)}$$

One key problem with expert systems is *knowledge acquisition*. However thoroughly a human expert tries to recapitulate his principles of reasoning, it became clear that in practice not all relevant knowledge can be formulated explicitly. Some rules of thumb and "gut feeling" that can not further be explained inevitably escape the knowledge engineer. This is called the *knowledge engineering bottleneck*. One way to improve this situation is to use machine learning algorithms to directly learn from raw empirical data. Later chapters will examine some algorithms for that purpose.

Truth maintenance can be another problem. How can one guarantee that new rules do not conflict with present knowledge and that no contrary conclusions will be

produced? This is especially a problem with non-monotonic reasoning where it is generally assumed that adding new rules requires modification of existing ones. One way to guarantee conflict resolution and truth maintenance is by using predicate calculus and a conflict resolution algorithm as already implemented in Prolog. Then each new rule can be tested whether it contradicts any other rules at the time of assertion. Should that actually happen, however, there is no standard way to proceed, whether to exclude the new rule, modify it, or change some of the old rules. In general, conflict resolution strategies are domain dependent.

Rule-based systems were first used for medical diagnosis[25]. These early systems have matured and some of them are now in practical use, e.g. as consultant systems for antibiotic therapy[26] and internal medicine[27,28], as intelligent monitoring system in oncology[29] or as knowledge-based interpretation assistance of pulmonary function test data[30] and hepatitis serological data[31].

Predicates are discussed in a later chapter on predicate calculus.

2.1.3. Knowledge Processing

Knowledge processing refers to the manipulation of data that is stored using a knowledge representation formalism. This section explains some basic concepts of manipulating a knowledge base with a computer.

Rules of Inference. Reasoning on data encoded in one of the previously mentioned knowledge representation formalisms can be done by *induction*, *deduction* or by *abduction*.

Inductive inference means generalisation from individual examples to general rules. In mathematics, a theorem can be logically proven by inductive inference. For that purpose one has to prove that the theorem is correct for the simplest example i. Then, assuming the theorem is correct for example i, one tries to mathematically derive the validity of example $i+1$. Induction in every day life and also in machine learning has a different meaning. There, it refers to the process of cre-

25 B. G. Buchanan, E. H. Shortliffe (Eds.), **Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project**, Addison-Wesley, 1984.

26 T. A. Pryor, R. M. Gardner, P. D. Clayton, H. R. Warner, *The HELP System*, **Journal of Medical Systems**, 7, pp. 87-102, 1983.

27 R. A. Bankowitz, M. A. McNeil, S. M. Challinor, R. C. Parker, W. N. Karpoor, R. N. Miller, *A computer assisted medical diagnostic consultation system service*, **Annals of Internal Medicine**, 110, pp. 824-832, 1989.

28 K. P. Adlassnig, M. Akhavan-Heidari, *CADIAG-2 / GALL: An experimental diagnostic expert system for gallbladder and biliary tract diseases*, **Artificial Intelligence in Medicine**, 1, pp. 71-77, 1989.

29 E. H. Shortliffe, A. C. Scott, M. B. Bischoff, A. B. Campell, W. van Melle, C. D. Jacobs, *An expert system for oncology protocol management*, in **Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project**, (B. G. Buchanan, E. H. Shortliffe Eds.), Addison-Wesley, 1984.

30 J. S. Aikins, J. C. Kuntz, E. H. Shortliffe, R. J. Fallat, *PUFF: An expert system for interpretation of pulmonary function data*, **Computers and Biomedical Research**, 16, pp. 199-208, 1983.

31 K. P. Adlassnig, W. Horak, C. Chizzali-Bonfadin, *Integrated medical database and expert system HEPAXPERT-II: Automatic interpretation of tests for hepatitis A and B*, in **Proceedings MIE'91**, Springer-Verlag, Berlin, 1991.

ating a classification over a previously unstructured set of examples (unsupervised learning) or of finding common features in members of one class that can be used to predict the class on the basis of its properties (supervised learning). Both supervised and unsupervised learning are error prone, e.g. noisy data may give rise to rather unspecific rules.

The algorithm ID3 will be described here as an example for supervised learning. Later we will see its application to membrane protein sequence prediction. ID3[32] is a descendant of the concept learning system[33]. The algorithm learns rules which are represented as a decision tree. A decision tree is similar to a flow chart in which a node of the tree represents a test on an attribute and each outgoing branch corresponds to one conceivable result of this test. Each terminal leaf corresponds to one class. To classify a new example, a path from the root of the decision tree to one of the leaf nodes is followed. At each node the branch with the corresponding value of the example is chosen. In prediction mode, the class of the leaf node represents the class prediction for the current example.

To build a decision tree one starts with a single node containing all training examples. If all examples at that node are of the same class, ID3 has accomplished its task and terminates. If not all examples on that node are in the same class they are divided into different branches depending on their value in one of the attributes. That attribute is chosen where most examples with similar values belong to the same class. An entropy function is used to measure the increase in order that is introduced by each attribute. In the following formula i is the index of new branches under the node to be expanded and j is an index over all possible classes. p_{ij} is the probability of class j estimated from the data in branch i; w_i is the occupancy of branch i (number of examples in branch i divided by number of examples in parent node). The entropy increase is then calculated as follows:

$$\text{Entropy} = -\sum_i w_i \sum_j p_{ij} \log_2 p_{ij}.$$

The stepwise growth of the decision tree is repeated until each node contains only examples in one class. Finally, the tree can be translated into rules. A condition is the conjunction of attribute-value pairs along the path from the root to a leaf. The class assignment of the leaf becomes the conclusion of that rule. Table 2 contains some sample data of a toy problem about properties of tigers. Eight instances are described by three attributes SIZE, AGE and whether the object has claws (CLAWS?). The AGE and CLAWS? attributes have only two possible values, SIZE has three. The task is to find the minimal decision tree to discriminate tigers from non-tigers. Let us calculate the total entropy change for separating the examples according their CLAWS? property.

$$\begin{aligned}
\text{CLAWS} &= \text{yes} \quad (3 \text{ tigers}, \quad 2 \text{ non-tigers}) \\
\text{CLAWS} &= \text{no} \quad (0 \text{ tigers}, \quad 3 \text{ non-tigers}) \\
\text{Entropy}_{\text{left}} &= \frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5} = -0.971
\end{aligned}$$

32 J. R. Quinlan, *Induction of Decision Trees*, **Machine Learning**, vol 1, pp. 81-106, 1986.
33 E. B. Hunt, J. Marin, P. T. Stone, **Experiments in Induction**, Academic Press, NY, 1966.

	Attributes		**Class**
CLAWS?	AGE	SIZE	
claws	old	large	tiger
no claws	young	large	not tiger
claws	young	medium	tiger
claws	old	small	not tiger
claws	young	small	not tiger
claws	young	large	tiger
no claws	young	small	not tiger
no claws	old	large	not tiger

Table 2: Sample Data for ID3.
Three attributes are used to describe two classes of objects: tigers and non-tigers.

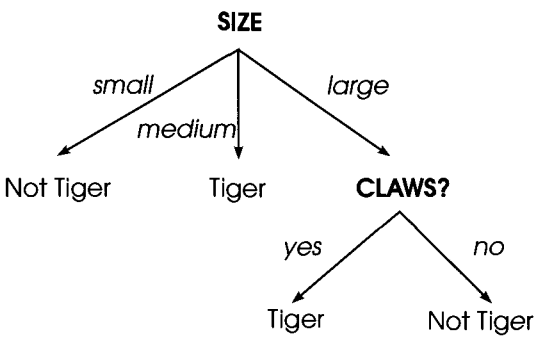


Figure 4: Decision Tree Output of ID3.
This decision tree is produced by ID3 for the toy data in Table 2.

$$\text{Entropy}_{\text{right}} = \frac{3}{3}\log_2\frac{3}{3} + 0 = 0$$

$$\text{Entropy}_{\text{total}} = -\left(\frac{5}{8}\cdot\text{Entropy}_{\text{left}} + \frac{3}{8}\text{Entropy}_{\text{right}}\right) = 0.607$$

The respective entropy values for AGE and SIZE are 0.955 and 0.500. SIZE is chosen as the top node because it brings the lowest entropy increase and therefore introduces as much as possible ordering in the tree. In the next step, the second best attribute CLAWS? is chosen. After that, the attribute AGE is no longer required to discriminate the given data into tigers and non-tigers. If, as a new example, a young tiger of small size is presented, the current set of rules corresponding to the decision tree in Figure 4 would be inaccurate since it would classify any small individual as non-tiger. – This example illustrates how essential features can be automatically extracted from raw data. The same procedure could be used for interpretation of other class-attribute-value information as e.g. spectroscopic data or properties of chemical compounds that are relevant for a particular reaction.

One problem of ID3 is its sensitivity to noise. If the sample data contains inaccurate or false measurements no good rules are generated. Instead, very long decision trees with many leafs are created which are not easily comprehensible. This is a result of the bottom-up approach of ID3, where feature selection starts from individual examples. Sometimes tree pruning[34] can increase rule specificity and coverage by halting the classification algorithm before examples in all nodes are completely resolved. Top-down classification algorithms, as e.g. implemented in CN2[35], start with a most general property description which is then successively refined to become more and more specific to discriminate between different classes. This also reduces the effect of noise in data.

Deduction is far less controversial than induction. Assuming an observed fact p and a rule "IF p THEN q" it is then correct and valid to infer the existence of q. This type of inference is called *modus ponens*. Three preconditions for using *modus ponens* are to determine whether p is given in a particular situation, to verify whether the rule is correct and whether it is applicable in that context.

Abduction is similar to deduction, only that not p but q is given together with the rule "IF p THEN q". In that situation, one can be tempted to explain the occurrence of q through p because q would then follow from p. However, there may be other ways for q to occur independently of p which is why abduction used in this way is not guaranteed to deliver definite results. Nevertheless, this is the inference method most often used in empirical science. If one substitutes q with $\neg v$[36] and observes v then p can be excluded if the rule "IF p THEN $\neg v$" was correct. Thus, either p or the rule "IF p THEN $\neg v$" can be tested.

Forward Chaining. *Forward chaining* is a generalisation of *modus ponens*. Let us assume we have knowledge represented as rules in implicit quantifier form with variables. Then presenting a new fact that can be unified with the condition part of some of these rules has the effect of asserting their conclusions to the current knowledge base. For example, if we have a rule "IF (x IS PROTEIN) THEN (x HAS AMINO ACID RESIDUES)" with the one variable x and the fact "CRAMBIN IS PROTEIN" is presented, then this fact can be unified with the condition part by substituting x with CRAMBIN. This would then result in the insertion of the fact "CRAMBIN HAS AMINO ACID RESIDUES" into the knowledge base. *Unification* is the process of substituting variables in one or two expressions such that they can be made equal[23]. The combination of unification and *modus ponens* allows inference to be concentrated on "interesting" conclusions. Variables express generality in facts and rules and unification ensures that only properly matching combinations are used for deduction. Forward chaining is also called *assertion-time inference* as it is performed when building a knowledge base.

Backward Chaining. In contrast, *backward chaining* is done at query time i.e. when a knowledge base has already been built and now one wants to know whether a particular fact or rule is compatible with that background knowledge or can be

34 B. Cestnik, I. Kononnenko, I. Bratko, *Assistant 86: a knowledge-elicitation tool for sophisticated users*, in **Progress in Machine Learning** (Proceedings of the 2nd European Working Session on Learning, I. Bratko, N. Lavrac Eds.), Sigma, Wilmslow, UK, pp. 31.45, 1987.

35 P. Clark, T. Niblett, *The CN2 induction algorithm*, **Machine Learning**, vol 3, pp. 261-283, 1989.

36 $\neg v$ is pronounced "non v" and means the negation of the fact v.

Inference	From	And	Deduce	plus Unification
Deduction	p	$p \rightarrow q$	q	Forward Chaining
Abduction	$\neg q$	$p \rightarrow q$	$\neg p$	Backward Chaining

Table 3: Modus of Inference for Forward and Backward Chaining.
Summary of resolution principles. Both modes of inference can be generalised to "($\neg a$ and $a \vee b$) infers b". Use ($a = \neg p$ and $b = q$) or ($a = q$ and $b = \neg p$) for substitution.

derived from it. The goal to be proven is compared with the conclusion parts of the rules in the knowledge base. If a conclusion and the goal can be unified the unifying substitution is recorded and an attempt is made to prove the conditions of the matching rule(s). If the conditions can be matched by other appropriate facts in the knowledge base the problem is solved. If not, the conclusion parts of other rules are used to try to resolve the new subgoal(s). The main problem with backward chaining is to limit the search space that grows exponentially with the number of rules. In general, it can not be predicted which rules are most promising to try, so one has to look for application specific heuristics when brute force search exceeds the computational resources at hand. Table 3 summarises the dependencies between deduction, abduction, forward and backward chaining.

Problem Solving as Search. Three well known fundamental search techniques are depth-first, breadth-first and best-first search, the latter also known as A* algorithm[37]. *Depth-first search* is done by exploring the search space in a vertical direction. For example, searching the tree of all possible chess games by depth-first strategy would require all games to be systematically played to the end, one after the other.

In contrast, *breadth-first search* would extend the search tree horizontally. For the chess example, in the first step, all games with only one white move would be examined; then all possible continuations with one white and one black move; then all games with one white, one black and another white move; and so on. While depth-first search examines one completed game after the other, breadth-first search examines all games with the same number of moves simultaneously. In the case of chess, both approaches would fail in practice. With depth-first search a computer has to examine too large a number of games for a mortal, human programmer to wait for the results. Breadth-first search would soon use up all available memory by filling it with uncompleted games and abort. Depth-first search is advisable for applications with an easily calculated objective function using a fast computer while breadth-first search is better suited for computationally expensive tasks that can be decomposed into compact subgoals. It is also possible to merge both strategies and examine all proposed solutions that are completed by a certain node level.

A more sophisticated combination of depth-first and breadth-first search is *best-first search*[23,38]. At each step only the *most promising* partial solution generated so far

37 pronounced "A-star".
38 P. E. Hart, N. J. Nilsson, B. Raphael, *Correction to: A formal basis of the heuristic determination of minimum cost paths*, **SIGART Newsletter**, vol 37, 1972.

is extended next. Therefore one needs a function to estimate the quality of a partial solution. This can be calculated as the sum of the path length from the root to the current node and an estimate the distance from that node to the final goal. If the cost of traversing the path to the node is neglected, the node estimated closest to the goal is extended first. If a solution with the fewest number of steps is required, the cost function per step can be made a constant. The cost function determines the convergence of a search. If the cost to reach the goal state is perfectly estimated a direct path between start node and goal will be found. If the cost function does not overestimate the actual cost of getting from that node to the goal the optimal path will be found, otherwise nothing is guaranteed.

2.2. Applications

This section contains some examples for the application of expert system techniques to a range of knowledge processing tasks in molecular biology.

2.2.1. Computer Aided Reasoning in Molecular Biology

Research in molecular biology can be conceptionally divided into three phases, two theoretical ones and one empirical[39]. The first theoretical phase includes:

1) searching literature,
2) summarising, updating and developing biochemical models, and
3) planning experiments to gather new data.

Then an empirical phase follows with:

4) carrying out experiments and
5) gathering data and measurements.

Finally, another block of theoretical activities completes the work:

6) interpreting the results and
7) documenting them.

Iterating over these seven steps a researcher builds evidence for improved and more detailed models. Computers can assist on all theoretical aspects and by controlling robots they can also contribute to the empirical parts. The latter aspect is not further considered here. A major intellectual challenge is intelligent support of the human researcher in steps 1, 2, 3, 6, and 7.

Step 1) Literature search can already be done conveniently by computer. A number of data banks offer their services for retrieving references, abstracts or

39 S. Schulze-Kremer, *CAROBT – Computer Aided Reasoning On Biochemical Topics*, **Project Proposal for the CAROBT Project** to the German Minister of Research and Technology, Grant No. BEO 0317199A, 1988.

whole articles in electronic form[40,41,42]. Although these bibliographic data bases contain a huge number of references, access and retrieval in most of them is still done by basic string search. One can search for an author, one or more keywords or for subject-headings (used to tag related articles on one subject). The latter allows, for example, searching for AIDS without retrieving articles about hearing aids which would get selected on the basis of pure (case insensitive) string matching. Using knowledge representation formalisms these data bases can be converted into knowledge bases. A dialogue between researcher and knowledge base could then actually *focus* on issues on the level of biochemical objects or experimental methods. The knowledge base server would have an "understanding" of the objects discussed in the sense that the knowledge base provides a context for each object which can be made explicit. An example of such a system at work will be presented later in this chapter.

Step 2) Summarising, updating and developing molecular biological models requires more abilities in addition to knowledge retrieval[43,44]. Summarising a scientific article is difficult if no abstract or title with keywords is provided. In that case, an understanding of what is the central issue must be gained. This cannot yet be done automatically by a computer program, but the process of knowledge engineering can be used to transform a scientific article, sentence by sentence, rule by rule, object by object, into the format of a knowledge representation formalism. *Encoding* knowledge in a knowledge representation formalism does not produce a summary of the original document but it has the effect of creating a condensed copy of all its information in the form of facts, hypotheses and conclusions. Forward chaining (insertion time inference; see deduction, above) can help identifying inconsistent facts and conclusions. *Updating* a model is done by replacing old statements of the knowledge base with new ones. *Building* a model is implicitly done when adding new facts or rules. Formation of hypotheses can be initiated by selecting focus, scope and level of detail for the extraction and presentation of a particular subject in the knowledge base. All knowledge relevant to one subject is part of its model (possibly yet incomplete or inconsistent). To get an explicit model description the knowledge base can be asked to print out all statements that are connected to a particular subject and which are not more than a certain number of nodes, objects, rules (etc.) apart from the central subject. The output could automatically be translated into simple English phrases for better readability.

Step 3) Planning experiments is at least as difficult as building a model. It requires the definition of a goal, understanding of the objects and of experimen-

40 **Medline**, U.S. National Library of Medicine (NLM), MEDLARS Management Section National Library of Medicine, 8600 Rockville Pike, Bethesda, MD 20894, Tel. (US) 301-496-6193, 1993. Internet World Wide Web Server at http://www.nlm.nih.gov.

41 **Computer Database**, Information Access Company, 362 Lakeside Drive, Foster City, CA 94404, Tel. (US) 415-378-5200, 1993.

42 **PaperChase**, Centre for Clinical Computing, 350 Longwood Avenue, Boston, MA 02115, US, 1993.

43 P. D. Karp, *Hypothesis Formation and Qualitative Reasoning in Molecular Biology*, **Ph.D. Thesis**, Computer Science Department, Stanford University, CA, 1989.

44 D. S. Weld, *Qualitative Reasoning and Automated Modelling*, **Working notes Symposium Artificial Intelligence and Molecular Biology**, AAAI, Stanford University CA, pp. 144-148, 1990.

tal methods available and a strategy for combining single steps of an experimental procedure. To test a new hypothesis one has to devise an experiment that could falsify that hypothesis. The experimental set-up follows from the functionality of the objects interacting in the hypothesis. This and some general constraints are in principle enough to deduce and propose an experiment. M. Stefik's system Molgen is an example for a knowledge based system capable of planning simple genetic experiments. It will be discussed in detail later in this chapter.

Steps 4) and **5)** require physical actions and therefore would have to be done by a computer controlled robot. There is a large potential for automating experimental procedures in molecular biology but as this book focuses primarily on algorithmic aspects, robot automation of experimental procedures will not be considered here.

Step 6) Interpretation of experimental results is difficult because of inaccuracies in measurements and because there is an inherent bias in the way numbers are compared. Nevertheless, knowledge based systems can help here, too. First, numerical procedures and statistical algorithms can be used to analyse a series of data points, e.g. approximate a line or calculate a standard deviation. Alternatively, genetic programming[4] can generate new models of a correlation function. These numerical findings can be translated into statements like "THERE IS A LINEAR CORRELATION BETWEEN x AND y UNDER THE ASSUMPTION z", written in mathematical syntax $X \wedge Z \rightarrow Y$. These statements and other qualitative findings can then be asserted into a knowledge base where either contradictions become apparent or consequences of that new rule can be inferred and considered for further experimentation.

Step 7) Documentation of experiments may at first sight seem not too intellectually challenging a task for automation by computer. Yet, everyone who had to summarise his or her findings in a manuscript to be submitted for publication in a scientific journal knows how much thinking goes into the organisation of the text before one can start writing. What is the innovative aspect of the work? How does it compare to related work? What are the critical changes in the material and methods used? What conclusions can be drawn from the results? What new problems arise? These are some of the questions that have to be asked. Knowledge based systems can be of help in answering some of them. The central subject will still have to be identified by the researcher. When compiling the description of materials and methods a knowledge base could retrieve those parts of previously described methods that are used in the current work. As with hypertext[45] documents where pointing to single words or expressions on the screen causes a new window to appear that contains an explanation of the selected item (which itself may be inspected further etc.), a knowledge base can be used to describe a known method in full detail. Complex entries are defined by simpler objects, rules (etc.), and in addition to hypertext, a knowledge base can use its procedural attachments to *simulate* (parts of) an experimental procedure. For practical purposes one would specify the components of the method which were already used in an earlier experiment together with the desired level of detail and the knowledge base would generate a description of these relevant components with their references. Only changes and new devel-

45 J. J. Cimino, P. L. Elkin, G. O. Barnett, *As we may think: the concept space and medical hypertext*, **Computational Biomedical Research**, vol 25, no 3, pp. 238-63, 1992.

opments would have to be specified explicitly by the researcher. Documentation of results can also be supported. By forward chaining, a knowledge base can be made to produce a (potentially very large) list of logically correct conclusions which can be drawn from the new results together with existing background knowledge. The researcher checks this list to see whether he or she has included all implications and major ramifications of the work. A knowledge base can also be used to point out contradictions to other work which will then have to be resolved in future experiments. Thus, a realistic objective is not a program that writes complete manuscripts to be submitted for publication but one that supports the researcher on time consuming and laborious tasks which can be supported by expert system technology.

The benefits of knowledge based systems in molecular biology are plentiful. In addition to retrieval of static knowledge the system can use inference to derive facts, rules, hypotheses and plans. A carefully implemented knowledge base can provide fast access to much more detailed information than human experts will ever care to memorise. A knowledge based system can be used as an interactive tutor to learn about new domains by browsing through the data and by asking questions. Documentation of experiments can be facilitated as explained above. Knowledge bases can grow with increasing empirical data and machine learning algorithms can be used to help organise new raw data.

Example. To give a flavour of the kind of conversation on molecular biological topics between a researcher and a knowledge based inference system to come, a fictitious dialogue on aspects of phenylalanine synthesis in *E. coli* is given below. Before that, the main objects and mechanisms[46] involved are briefly reviewed here for non-biologists. *E. coli* is a bacterium species. Phenylalanine (abbreviated *phe*) is one of the basic 20 naturally occurring *amino acids* that are used as building blocks for protein synthesis. A variety of proteins can perform a wide range of functions in a cell, e.g. catalysing the synthesis of those amino acids that cannot be accumulated in sufficient quantities from the environment. This is what the phenylalanine synthesising protein does in the case of low phenylalanine concentration. If no phenylalanine is found or can be synthesised cell growth stops in *E. coli* until conditions improve. Information for protein formation is encoded in *genes*. DNA, a phosphate-deoxyribose polymer, is the carrier of genetic information which is encoded in sequences made from an alphabet of four nucleotides. Those sections on DNA which contain the structural information for proteins are copied onto a messenger molecule called mRNA. mRNA is subsequently scanned by cellular protein synthesis machines (*ribosomes*) and directs the synthesis of the protein it codes for. This process is called *translation*. Expression of a gene comprises both transcription and translation and therefore implies net synthesis of the coded-for protein. An *operon* includes genetic control regions and structural genes for the expression of a complete enzymatic pathway (e.g. for ß-galactosidase synthesis).

The Jakob-Monod model of gene regulation states that transcription, the step in protein synthesis in which a mRNA copy is built from a DNA template, begins when a special protein called RNA polymerase binds to a specific location in front of the structural gene, called the *promoter*. Another molecule, called *repressor*, which is made elsewhere in the cell and is activated by an excess of the end product of the

46 L. Stryer, **Biochemistry**, 3rd edition, Freeman, New York, 1988.

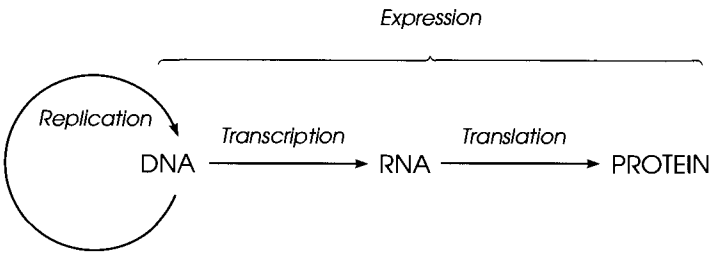


Figure 5: Gene Expression and Protein Synthesis.
Transcription copies information from DNA to mRNA molecules which program ribosomes for protein synthesis. See also Figure 3.

gene (e.g. phenylalanine), can bind to a nearby site on the DNA, called *operator*. If the repressor binds it prevents mRNA polymerase from moving on to the promoter, thereby halting the protein-producing machinery at the very first step. Mutations in DNA (chemical changes of single nucleotides) can destroy the functionality at either the promoter or operator location. Not every mutation has a damaging effect, but the majority of them do.

The synthesis of phenylalanine is also controlled by another mechanism, *attenuation*. Here, the cell uses the fact that translation of mRNA may start before its transcription is fully completed. In phenylalanine synthesis the phenylalanine synthesising protein has near its N-terminal end seven phenylalanine residues. If phenylalanine concentration is low, translation across that portion of mRNA will be slow. Pausing in this position a little longer than at high phenylalanine concentration (where phenylalanine is readily available for synthesis and hence synthesis proceeds faster) the ribosome prevents a structural change in mRNA that would otherwise cause abnormal termination of transcription and thus failure of expression of phenylalanine synthesising protein. Several other operators for the biosynthesis of amino acids in *E. coli* are known to have such *attenuator sites* (threonine, tryptophan, histidine). The following dialogue is clearly speculative[47] but illustrates in a translated-to-English form the kinds of reasoning a knowledge based inference system should be able to perform. The markers "⇒" and "⇐" denote input to and output from the system. Some comments are given in between in brackets.

⇒ Assume E. coli is growing in a medium of low *phe* concentration!
⇐ Situation: Cell is E. coli. *Phe*-operon is active. Biosynthetic enzymes for *phe* synthesis are produced. Cell grows.
[Using a functional model of the *phe*-operon, the system simulates the described experiment and displays the result.]
⇒ Affirmative. Now, assume a similar situation but with a mutation in the *phe*-operon and no cell growth!
⇐ Situation: Transcription of *phe* gene is disabled. Cell growth is disabled because of *phe* starvation.
[The system knows that a mutation in a DNA promoter region is likely to have a destructive effect and deduces from disabled *phe* synthesis as a result of lack of *phe* synthesising enzyme together with a low *phe* content in the medium that there is a lack of an essential amino acid which prevents growth.]

47 P. Friedland, L. H. Kedes, *Discovering the secrets of DNA*, **Communication of the ACM**, vol 28, no 11, pp. 1164-1185, 1985.

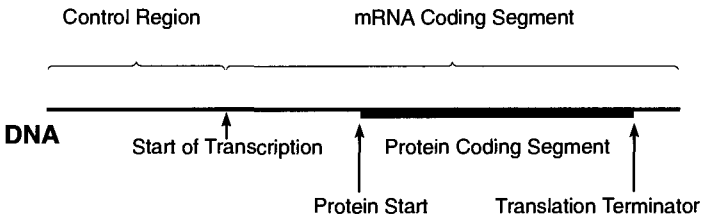


Figure 6: Basic Structure of a Gene.
A gene consists of a control region that has various control sites, a binding site for transcription and a structural region coding for a protein.

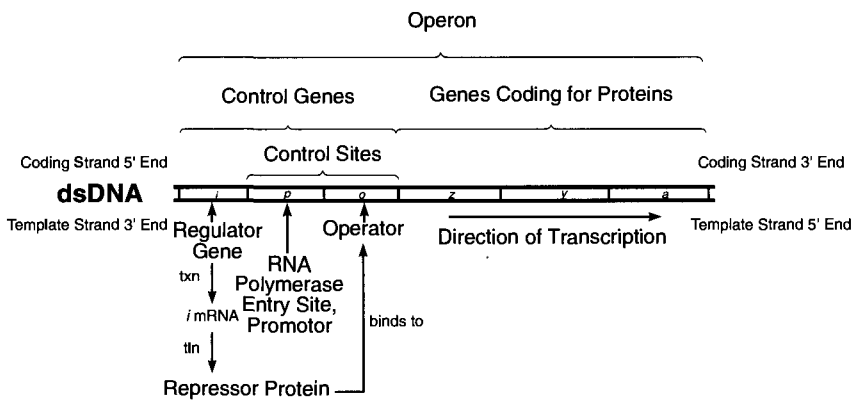


Figure 7: *Lac*-Operon of *E. coli*.
E. coli can use lactose as its sole source of carbon. An essential enzyme in the metabolism of this disaccharide is ß-galactosidase, which hydrolyses lactose to galactose and glucose. The *lac*-operon is a co-ordinated unit of ß-galactosidase gene expression. The gene *i* codes for a repressor protein that binds to the operator site thereby blocking transcription of the ß-galactosidase structural genes. Transcription of gene *i* is labelled "txn", translation "tln". If lactose is present some of it is transformed into allolactose which binds to the gene *i* product tighter than DNA. Now, the gene *i* product is no longer free to bind to the operator and expression of ß-galactosidase can proceed. Thus, the substrate lactose itself induces expression of the enzymes necessary for its degradation. There are some short consensus sequences known to identify a promoter. Eucaryotes can have other elements controlling transcription at virtually any location on DNA. Also only in eucaryotes, genes coding for proteins can be interrupted by so-called introns which have to be spliced out after transcription. Transcription control by attenuation is not shown here.

⇒ Explain normal growth of those cells!
⇐ Situation: Mutation has NO deleterious effect.
[Before, the system chose to report on the more likely case of a non-functional promoter. This time, the additional information of cells growing normally is a hint that actually the less likely case occurred where a mutation in the promoter did not effect transcription.]

⇒ Assume another mutation in the leader region that causes constitutive synthesis of *phe*!
⇐ Situation: Repression is disabled. Possible explanation: repressor does not bind to operator.
[This is a case of backward chaining. The system tries to deduce the result of permanent biosynthesis from some other fact. It had the general information that an operator can bind a repressor and thereby inhibit transcription. This rule is unified with the current situation at the *phe*-operon.]
⇒ Assume: another mutation is introduced in the leader region. *Phe* synthesis is still regulated by *phe* content in the medium but the rate of *phe* synthesis in low *phe* medium is much higher than in wild type.
⇐ Situation: No complete explanation was found. Mutation is likely to occur between operator and structural genes because *phe* synthesis is still intact (excludes significant changes in the structural gene coding for *phe* synthesising protein) and because regulation still works (excludes serious damage of operator). However, no functional element is known in this region to account for the change in *phe* production at low *phe* concentration in medium. Generally, mutations can induce changes in transcription or translation speed. Taken together with the fact of unusual high *phe* production at low *phe* concentration the mutation site could have a positive effect on expression that was enhanced by the second mutation or a negative effect that was disabled. The later is statistically more likely.
[This is the hypothesis discovery mode, in which the system tries to instantiate new facts to cover the missing link in the explanation. The facts given to the program can be explained by the transcription control mechanism *attenuation* which the program does not know about yet.]
⇒ What types of negative effects are there?
⇐ Transcription or translation might be halting or pausing at the mutated site or the site changes the elongation rate of RNA polymerase or ribosomes. It could also produce a repressor protein or introduce a new promoter. The latter is very unlikely, as that promoter would have to be regulated by *phe*, too.
[A list of hypotheses is generated by examining all known steps from transcription to peptide synthesis. Each link could fail in a specific manner[43].]

2.2.2. Knowledge based Representation of Materials and Methods

The "Materials and Methods" section of molecular biological research papers is a good candidate to start with for the construction of a knowledge base. It is important because materials and methods must be included in every paper to allow reproduction of an experiment. Progress in molecular biology is firmly linked to refining or introducing methods. Compared to the rest of a research paper, it is relatively easy to extract the "methods" information because only restricted language without arguments and hypotheses is used, similar to cooking recipes[48]. If one day it will be possible to automatically re-encode the "materials and methods" sections of the majority of molecular biological papers such a knowledge base could be of great help for retrieving detailed and specific information about methods and materials across different publications. In combination with a language generator

48 R. McCartney, B. Moreland, M. Pukinskis, *Case acquisition from plain text: reading recipes from a cookbook*, **Technical Report** TR-CSE-92-20, University of Connecticut, Department of Computer Science and Engineering, 1992.

it could assist in automatically preparing the "methods and materials" section of new manuscripts and also be useful for organising and documenting work in the laboratory.

One way to capture the meaning of materials and methods from flat text is to use an object-oriented representation in combination with semantic nets, as was done by K. Baclawski, R. P. Futrelle, N. Fridman and M. J. Pescitelli[8]. Their approach of an electronic Laboratory Notebook is described here. Materials and methods describe a number of steps in which *input materials* are transformed into *output measurements*. The source of initial materials is referenced but their preparation is usually not further defined in the paper. One major problem in automatic literature understanding is naming conventions. Although there are international recommendations on how to name and classify a chemical compound[49], a protein[50], or an enzyme[51], not everybody follows them. Often, irregular names or mnemonic abbreviations are recognised by the research community and therefore will continue to be used. The corpus of the "methods and materials" section is encoded in SGML, the Standard Generalised Mark-up Language[52]. SGML is an ISO standard which is capable of representing the detailed logical structure of a text but not its typography and format. SGML tags every non-simple word item in the document. Every section heading, paragraph, superscript (etc.) has its own standard SGML tag. What SGML does not specify are matters such as font, column widths or page layout. The SGML format is used here for storage and as the standardised format for data exchange. The approach of Baclawski *et al* does in a first version classify names into broad categories as e.g. protein, plasmid etc. but does not try to resolve a systematic name into its components. As the database develops categories will be subclassified in a top-down fashion rather than analysed bottom-up. From this, the authors want to create a lexicon and thesaurus using both automated and human input.

Processes are one or more steps to transform input substances into output substances. Single steps have the parameters temperature, duration, input and output material, type of modification and a unique subset of their own, step-specific parameters. Steps can be combined to describe a more complex process which allows a hierarchical organisation of processes. Examples of elementary steps are incubation, centrifugation or resuspension. Most often, a particular piece of equipment is associated with a laboratory step. Sometimes different names refer to the same physical process but emphasise a particular purpose, e.g. as with *resuspend* and *wash*, where *resuspend* indicates that a different medium must be used. When answering queries on *washing*, a retrieval system must therefore be able to consider *resuspension*, too. Processes are linked via input and output substances which is often implicitly contained in the description of materials and methods and only vi-

49 R. S. Cahn, O. C. Dermer, **Introduction to Chemical Nomenclature**, Butterworths, London, 1979.

50 IUPAC-IUB Commission on Biochemical Nomenclature, *Abbreviations and Symbols for the Description of the Conformation of Polypeptide Chains*, **Journal of Biological Chemistry**, vol 245, no 24, pp. 6489-6497, 1970.

51 E. C. Webb, **Enzyme Nomenclature**, Academic Press, London, 1992.

52 M. Bryan, **SGML: An Authors Guide to the Standard Generalised Mark-up Language**, Addison-Wesley, 1988.

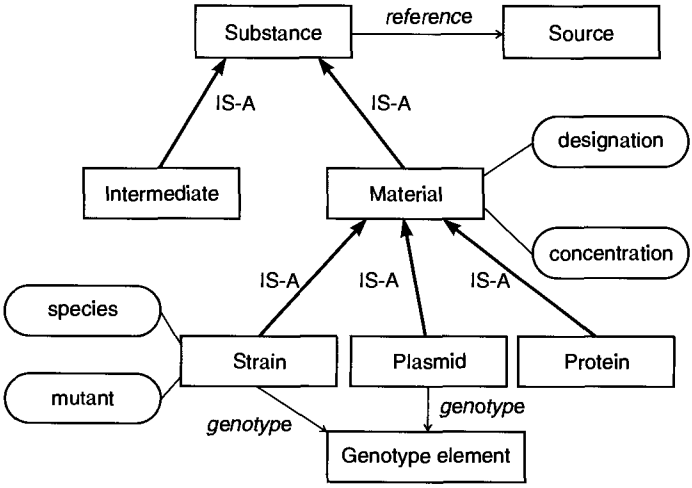


Figure 8: Schema for MATERIAL.
Each MATERIAL is a subclass of SUBSTANCE. There are INTERMEDIATE substances which generally refer to the unnamed output of one process which immediately becomes an input to the next step. Materials are the substances explicitly named in the paper which have various parameters as e.g. designation, concentration. There are various kinds of materials, such as strains, plasmids and proteins, each of which has its own set of parameters in addition to those of all materials. There are also relations between classes, such as the REFERENCE relation between SUBSTANCE and SOURCE (see ref. 8).

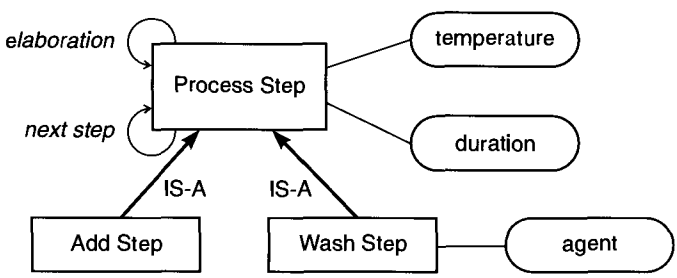


Figure 9: Schema for PROCESS.
All processes are subclasses of the class PROCESS-STEP which can be an elaboration of another step or the next step in a sequence of steps. Each step may have general attributes such as temperature, duration and some specific attributes (e.g. angular velocity of a centrifuge) (see ref. 8).

olations of this assumption are noted. In addition to physical output every process produces an output of information which has to be represented in the knowledge base. Fragments of two basic structures, the MATERIAL schema and the PROCESS schema, are presented by Baclawski *et al.* A semantic net representation of the MATERIAL schema is shown in Figure 8, of the PROCESS schema in Figure 9.

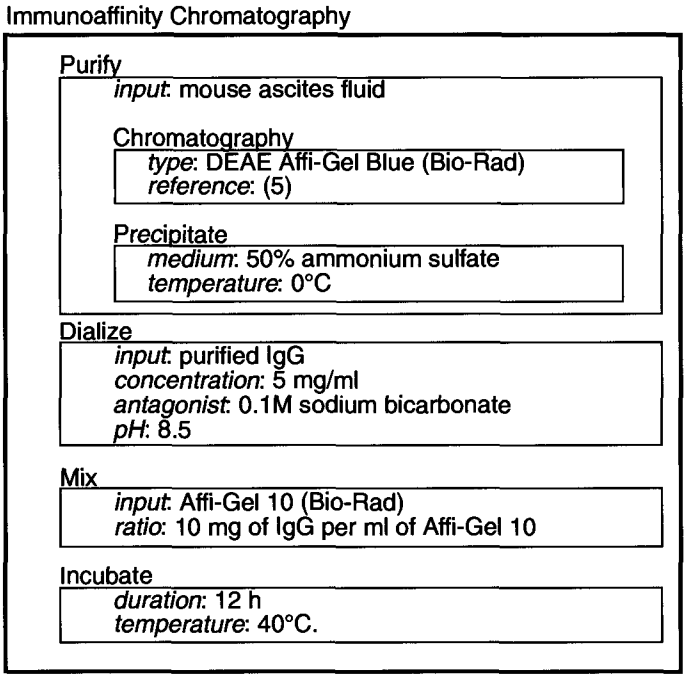


Figure 10: Frame Representation of Immunoaffinity Chromatography.
The single immunoaffinity chromatography process is divided into four subprocesses. More nested object definitions are required to replace some words and phrases in the diagram.

Baclawski *et al* give an example of the analysis of a part of a "materials and methods" section. The frame representation of this excerpt is shown in Figure 10. The text to be represented is the following:

> "Immunoaffinity chromatography. IgG was purified from mouse ascites fluid by DEAE Affi-Gel Blue (Bio-Rad) chromatography (5) followed by precipitation in 50% ammonium sulphate at 0°C. Purified IgG (5 mg/ml) was dialysed against 0.1M sodium bicarbonate, pH 8.5, mixed with Affi-Gel 10 (Bio-Rad) at a ratio of 10 mg of IgG per ml of Affi-Gel 10 and incubated for 12 h at 40°C."[53]

This example shows how knowledge on methods and materials can be stored in a structured format. It remains to refine the schemata of MATERIAL and PROCESS to capture all relevant details for a large scale application. One way to proceed is to start encoding complete "methods and materials" sections of a number of articles and see what objects and properties occur. This is exactly what Baclawski *et al* intend to do for articles related to chemotaxis of bacteria. Chemotaxis is the ability to control direction of movement of a unicellular organism in liquid phase in response to concentration changes of nutrients (attractor) or toxins (repellent). Chemotaxis

53 A. M. Stock, J. B. Stock, *Purification and characterisation of the CheZ protein of bacterial chemotaxis*, **Journal of Bacteriology**, vol 169, no 7, pp. 3301-3311, 1987.

requires sophisticated molecular switches as bacteria can only differentiate between a tumbling phase, which leads to a random change of direction, and a short straight movement. Steering is done by adapting the tumbling frequency.

Copyright of the journals to be extracted for a methods and materials knowledge base is an issue because of its obvious commercial value. Also, development of a thesaurus and a lexicon might be more expensive when done from scratch than using commercially available databases. From this it becomes clear that the successful development of a knowledge based expert system on materials and methods requires co-operation of: publishers of journals to provide the data; computer scientists to develop and implement the structure of a knowledge base; biologists to explain the relation among objects and methods in molecular biology; and linguists to understand the structure of the language for the design of an interface to enable a computer to *read* a text and decompose it into its basic terms and relations. Hardware requirements for this project can be met by fast workstations, mass storage devices (magnetic and optical discs) and a scanner with optical character recognition software to facilitate text input.

2.2.3. Knowledge based Exploration in Molecular Pathology

Senex[54,55] of S. S. Ball, V. H. Mah and P. Miller is a computer program written in Lisp and based on object-oriented representation as provided by the Common Lisp Object System (CLOS) and the Common Lisp Interface Manager (CLIM). Senex focuses on molecular mechanisms of the infirmities of old age, especially those related to the loss of cognitive function. Common Lisp objects are used to represent molecules and molecular processes. The Medical Subject Headings (MeSH)[56] index is the basis of the classification hierarchy implemented in Senex. The usual slot-filler notation allows a detailed specification of an object. Distinction between class objects and instances is made and default values are provided. Inheritance of properties is conditionally propagated down the object hierarchy. Knowledge acquisition in Senex must be done manually by a knowledge engineer using Lisp functions, macros, CLOS objects and methods. The system provides a number of graphic display options that allow visual inspection of the knowledge base, e.g. structure and alignment of genes and proteins, metabolic pathways, and organic molecules. Objects on the screen are mouse-sensitive and directly linked to their semantics. Senex can compute cartoons from a symbolic representation, e.g. for chemical reactions or membrane protein topology.

54 S. S. Ball, V. H. Mah, P. Miller, *New Senex directions towards a prototype gene expression map of the central nervous system*, **Working notes Symposium Artificial Intelligence and Molecular Biology**, AAAI, Stanford University CA, pp. 22-25, 1990.

55 S. S. Ball, V. H. Mah, *Senex: A CLOS/CLIM Application for Molecular Pathology*, **Proceedings First International Conference on Intelligent Systems for Molecular Biology**, AAAI Press, Menlo Park CA, pp. 38-46, 1993.

56 *Medical Subject Headings* is a controlled indexing terminology developed by the U.S. National Library of Medicine to describe the contents of medical and biological research papers. Source: National Library of Medicine (U.S.), **Medical Subject Headings**, National Tech-

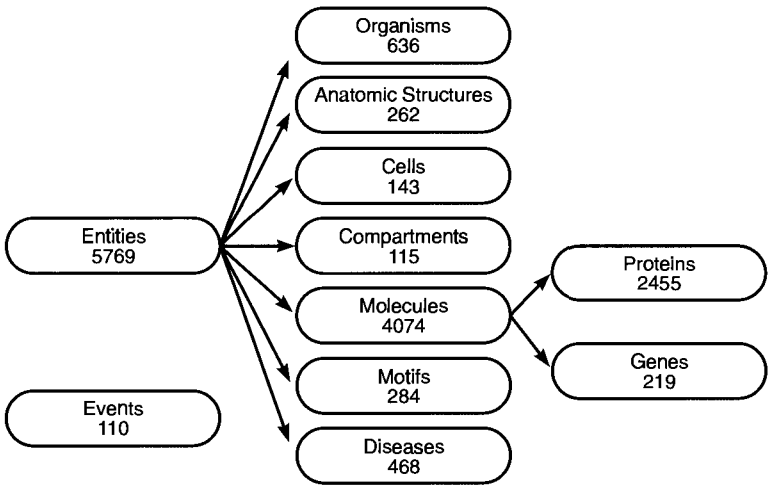


Figure 11: Classification Structure in Senex.
This is the class hierarchy and the numbers of instances implemented per class in Senex (see ref. 55).

So far, the authors have been experimenting with different search algorithms to browse through the knowledge base. For example, by telling Senex to ignore cell type, anatomic considerations and organism type, reactions of one enzyme occurring in different environments can be found. If pathways are found to differ significantly in different hosts such a result can serve as the basis for a new hypothesis to be tested in the laboratory. Three main difficulties seem to prevent a wider use of Senex as a knowledge based expert system. First, the user is required to be at least familiar with Lisp, CLOS and CLIM, a prerequisite that seriously diminishes the number of potential end users. Second, the choice of Lisp is definitely favourable for the realisation of symbolic programming but still involves a non-negligible restriction to suitable hardware platforms[57]. Third is the problem of knowledge acquisition: How will Senex learn in future? If every fact had to be hand-coded in Lisp this would restrain the growth of the knowledge base to a practical extent. Truth maintenance was also not addressed here.

nical Information Service, Springfield, VA 22161, tel + 703-487 5650, 875 pages, 1993. Internet World Wide Web Server at http://www.nlm.nih.gov.

57 Currently only Macintosh and Sun Sparc II Stations are supported. A Lisp system and an object-oriented programming environment for Lisp are required.

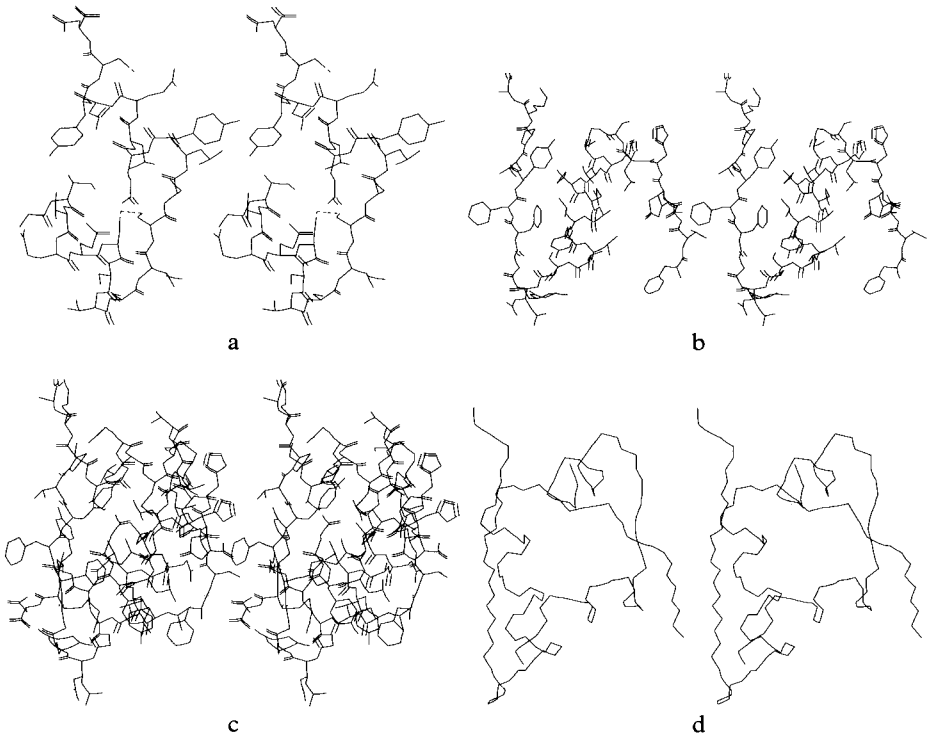


Figure 12: Four Skeletal Drawings of Pig-Insulin.
Diagram *a* shows the A chain of insulin, diagram *b* the B chain and diagram *c* both A and B chains, all including amino acid side chains. Diagram *d* shows the A and B chains but without side chains for an easy identification of the backbone with its three α-helices and one β-strand. The drawings are generated from original X-ray data of the 1INS entry of pig-insulin of the Brookhaven data bank[60]. The two halves in each diagram are stereo projections of the same molecule. To perceive a three-dimensional image of the molecule the reader can superimpose both structures by looking at them cross-eyed or with both eyes diverging. Alternatively, stereo glasses may be used.

2.2.4. Planning Cloning Experiments with Molgen

One of the first artificial intelligence programs for molecular biology came out of the Molgen project[58]. M. Stefik in his Ph.D. thesis[59] developed a method that can automatically generate a *plan* for producing a genetically modified bacterium. As an example the transformation of *E. coli* bacteria to produce rat-insulin will be presented.

58 M. Stefik, *Planning with Constraints. Parts 1&2*, **Artificial Intelligence**, no 16, pp. 111-169, 1981.

59 M. Stefik, *Planning with Constraints*, **Ph.D. Thesis**, Department of Computer Science, Stanford University CA, also as Report STAN-CS-80-784, 1980.

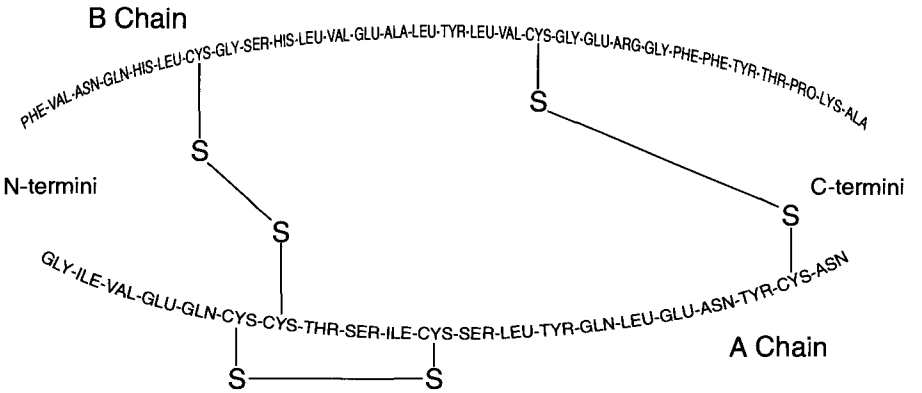


Figure 13: Primary Structure and Disulphide Bridges in Pig-Insulin.
Insulin has disulphide bridges between cysteine residues A7:B7, A20:B19 and A6:A11. Disulphide bridges are a means of stabilising the structure of a protein that is to be released into extracellular lumen.

Biological Background. Insulin[61] is a small protein of 51 amino acid residues. One insulin molecule consists of two polypeptide chains labelled A and B, which are covalently linked by two disulphide bridges between cysteine residues A7:B7 and A20:B19. A third disulphide bridge is found within chain A (A6:A11; Figure 13). The three-dimensional structure[62] of insulin is compact (Figure 12). Only the ends of chain B extend outside the otherwise globular sphere of the protein. Chain A has two α-helices and is folded between the arms of chain B, which has one α-helix and one small β-strand near the C-terminal end. Biologically active insulin is a dimer or a polymer of pairs of AB chains. A non-polar core is formed by buried aliphatic residues from both chains. Insulin is stabilised by several hydrogen bonds and salt bridges between both chains. The two insulin chains are produced from a single precursor molecule called proinsulin. Proinsulin is derived from preproinsulin by cutting off the N-terminal signal sequence which directs the nascent chain to the endoplasmatic reticulum. Insulin was the first protein to be sequenced in 1953 by F. Sanger.

Insulin is an important regulator for fuel metabolism. The secretion of insulin by β-cells of the pancreas is stimulated by glucose and the parasympathetic nervous system and serves as a signal for the fed state. Storage of fuels and the synthesis of

60 F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, E. F. Jr. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, M. Tasumi, *The protein data bank: A computer-based archival file for macromolecular structures*, **Journal of Molecular Biology**, no 112, pp. 535-542, 1977. Internet World Wide Web Server at http://www.pdb.bnl.gov.

61 J. Bordas, G. G. Dodson, H. Grewe, M. H. J. Koch, B. Krebs, J. Randall, *A Comparative Assessment of the Zinc-Protein Co-ordination in 2*Zn-Insulin as determined by X-Ray Absorption Fine Structure and X-Ray Crystallography*, **Proceedings of the Royal Society London**, vol 219, pp. 21, 1983.

62 M. J. Adams, T. L. Blundell, E. J. Dodson, G. G. Dodson, M. Vijayan, E. N. Baker, M. M. Harding, D. C. Hodgkin, B. Rimmer, S. Sheat, *Structure of Rhombohedral 2 Zinc Insulin Crystals*, **Nature**, vol 224, pp. 491, 1969.

proteins is elevated. Gluconeogenesis in the liver is suppressed and glycolysis accelerated which in turn increases the synthesis of fatty acids. The molecular mechanism of insulin controlled metabolism is known. Insulin acts by binding very tightly to receptors in the plasma membrane of target cells. The activated insulin receptor is an enzyme which catalyses the phosphorylation of tyrosine residues in target proteins. Binding of insulin causes autophosphorylation of the β-chains of the insulin receptor which maintains elevated kinase activity even after dissociation of insulin from the receptor.

When a person suffers from the disease *diabetes mellitus* his insulin level is too low, which results in an overproduction of glucose by the liver. Because insulin is deficient, glucose can not enter the cells where it is needed as a fuel. Instead, it is finally excreted in the urine when the blood glucose level exceeds the resorptive capacity of the renal tubules. Water accompanies the excreted glucose with the effect that an untreated diabetic in the acute phase is both hungry and thirsty. Glucose, more abundant than ever, is spurned. If no insulin is administered an overproduction of ketone bodies occurs that eventually lowers the blood pH level. In combination with dehydration low blood pH can cause the patient to fall into a coma. This is why insulin is needed in commercial quantities for the therapy of an estimated 1% diabetics of the population in industrialised countries. Until the advent of gene technology, insulin had to be extracted and purified from natural sources which made it rather expensive. Now, one can introduce the insulin gene into *E. coli*, make *E. coli* express the desired protein and harvest it. We will see Molgen generate the specific steps for this experiment.

The basic concept in gene technology for the production of proteins is as follows. First, one has to identify and characterise the desired protein. This involves either sequencing the protein itself or its gene. Then, DNA coding for the structural part of the protein has to be prepared. For short sequences (less than 100 nucleotides) this can be done by synthetic chemistry. Longer sequences have to be isolated from natural sources. This involves locating the desired sequence on a chromosome, cutting it out and separating it from other constituents. The purified gene is then introduced into a host cell (e.g. *E. coli* or yeast) and expressed. Finally, the protein has to be extracted from the cells. This can be facilitated by adding a signal sequence in front of the structural gene that causes the protein to be discharged by the cell into the environment where it can be extracted by chromatography. The practical difficulty with this procedure lies in the preparation and purification of large genes and the purification of the desired protein. Obviously, for medical purposes it must be guaranteed that the product is not contaminated by any potentially deleterious impurities.

Cutting genes is done by restriction enzymes. There are endonucleases that cut within a DNA sequence and exonucleases that digest DNA from one of its termini. Most endonucleases are sequence specific i.e. they cut only between certain sequence patterns. Figure 14 shows specific patterns for some endonucleases that are commercially available. Note that the sequences on both strands are palindromic.

DNA ligase is an enzyme that joins the ends of double-stranded DNA fragments. It catalyses the formation of a phosphodiester bond between the 3'-hydroxyl group at the end of one DNA chain and the 5'-phosphate group of the other end. In bacteria, NAD^+ is the energy source, in animal cells and bacteriophage it is ATP. The

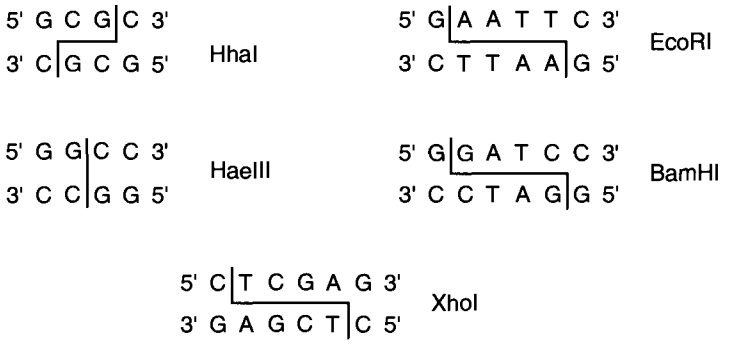


Figure 14: Restriction Enzymes.
Five common endonucleases and the patterns for their cleavage sites are shown. The two sequences are coding and template strand in double-stranded DNA. They are related by a 180° rotation.

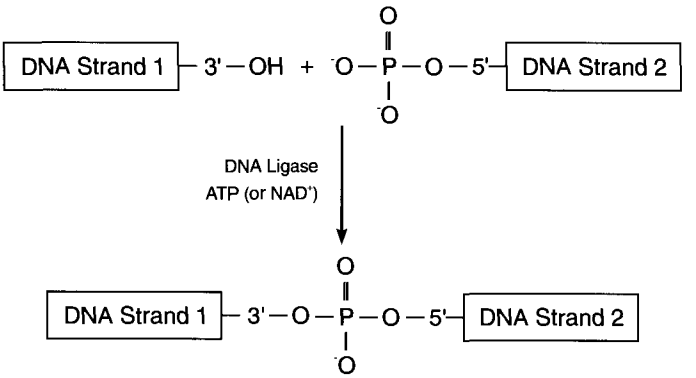


Figure 15: DNA Ligase Joins DNA Fragments.
Consuming one ATP (or NAD^+) the enzyme DNA ligase joins two DNA fragments at their 3' and 5' ends.

joining process is essential for normal DNA replication. It can also be used in recombinant DNA technology to link a desired gene into a vector. Some DNA ligases can join blunt ends (e.g. T4 DNA ligase), others need sticky ends with overlapping strands. The reactions involved in DNA ligation are the following.

$$\text{DNA Ligase} + \text{ATP (or NAD}^+) \rightarrow \text{DNA Ligase} - \text{AMP} + PP_i \text{ (or NMN)}$$
$$\text{DNA Ligase} - \text{AMP} + PO_4^- - 5' - \text{DNA} \rightarrow \text{DNA Ligase} + \text{AMP} - PO_4^- - 5' - \text{DNA}$$
$$\text{DNA} - 3' - \text{OH} + \text{AMP} - PO_4^- - 5' - \text{DNA} \rightarrow \text{DNA} - 3' - PO_4^- - 5' - \text{DNA} + \text{AMP}$$

$$\text{DNA} - 3' - \text{OH} + PO_4^- - 5' - \text{DNA} + \text{ATP (or NAD}^+) \xrightarrow{\text{combined}} \text{DNA} - 3' - PO_4^- - 5' - \text{DNA} + \text{AMP} + PP_i \text{ (or MNM)}$$

DNA ligase and restriction endonucleases can be used to insert a gene at a specific location in a chromosome or vector by the *cohesive-end method* (Figure 16). First, the DNA fragment to be extended is given sticky ends by joining a decameric linker to either end of the fragment by T4 ligase. The linker contains a sequence specific

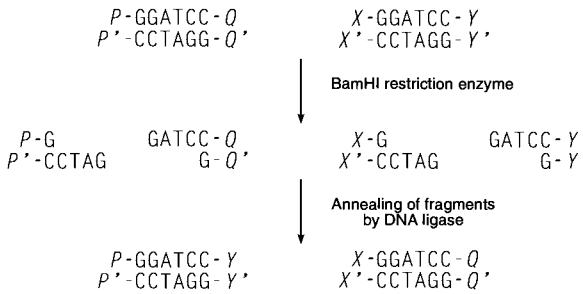


Figure 16: Joining DNA by the Cohesive-End Method.
Two fragments, each with a single BamHI restriction site can be uniquely cut and ligated with each other. Thus, the genes P and Y (and Q and X) are joined. Quoted letters denote the complementary strand. If P was connected to Y a circular DNA molecule is produced.

for a particular restriction enzyme. That endonuclease is then used to cut the linker. Now the target has defined sticky ends. The same is done with the DNA fragment to be inserted. After that, both DNA fragments have complementary ends which can form base pairs and be subsequently annealed by ordinary DNA ligase. The cohesive-end method for joining DNA can be made general by using short, chemically synthesised DNA linkers which can be made specific for any particular restriction enzyme.

Plasmids are naturally occurring circular, duplex DNA molecules. They can range in size from 2,000 nucleotides in one strand to 100,000 nucleotides. They can carry genes for the production of toxins, metabolism of natural products, or for inactivation of antibiotics. The latter property makes them especially useful for recombinant DNA technology. Plasmids are accessory chromosomes that can replicate independently of the host chromosome and which are dispensable under certain conditions. A bacterial cell may have no plasmid at all or as many as 20 copies of the same plasmid. One of the most widely known plasmids is called pBR322. It contains genes for resistance against two antibiotics, tetracycline and ampicillin (Figure 17). pBR322 can be cleaved at a number of unique sites by different endonucleases and have DNA fragments inserted into such cleavage sites. Plasmids are often called vectors because they can be used to transport DNA into bacteria cells. Once a plasmid is inside a cell its genes are transcribed by RNA polymerase of the host. The mRNA transcripts of the genes are then translated into protein by host ribosomes. Thus, *E. coli* can be made to produce proteins that are not coded for on its own genes. Some restriction sites in pBR322 lie in one of the antibiotic resistance genes. If DNA is inserted by the cohesive-end method into one of these sites the respective antibiotic resistance is lost for the cell carrying the recombinant plasmid. This effect is called *insertional inactivation*. Cells sensitive to both antibiotics have no plasmid and those resistant to both antibiotics carry a plasmid with no insert. Cells that are sensitive to one antibiotic carry a plasmid with a DNA insert in the reading frame of the other antibiotic. To select for these cells, first, all cells without a plasmid are eliminated by adding the antibiotic that was not inactivated. Then the second antibiotic is added. This prevents cells with no antibiotic

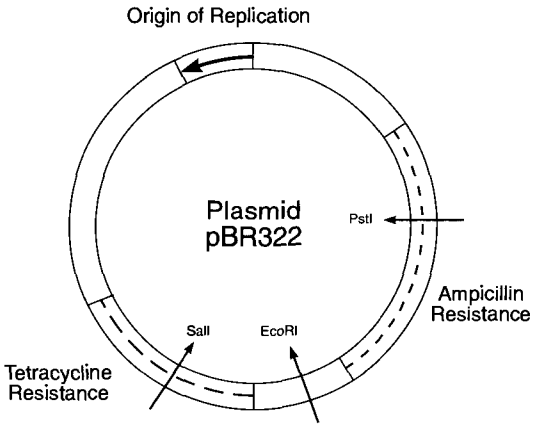


Figure 17: Plasmid pBR322.
This plasmid is one of a whole set of vectors that can be used to introduce novel genes into *E. coli*. The DNA fragments are inserted into one of the many unique restriction sites of pBR322, only three of which (PstI, SalI and EcoRI) are shown here. Origin of replication is the location where duplication of DNA starts.

resistance from further growth but does not kill them immediately. Other cells with intact metabolism rendered by a plasmid without an insert in the antibiotic gene continue to grow. Now, another toxin which causes premature termination of protein synthesis is added to the medium. This has the effect of amino acid depletion of those cells that were not impaired by the second antibiotic. Now, these cells will die because they cannot regenerate their supply of essential metabolic substrates. The cells inactivated by the second antibiotic will survive because they do not experience the destructive effect of the toxin. Their colonies can later be transferred onto an antibiotic-free medium and allowed to recover. Modified viruses, e.g. λ-phage, can also be used as a vector to carry large DNA fragments into cells. Viruses have the advantage that they can easily penetrate cells while introducing plasmids into cells at high rates is more difficult.

Planning Genetic Experiments with Molgen. The problem that Molgen was supposed to solve is this:

> What laboratory steps are needed to make a bacterium produce rat-insulin?

Or, as the user has to specify in Lisp:

```
(CULTURE-1 ORGANISMS:(BACTERIUM-1 EXOSOMES:
   (VECTOR-1 GENES:(RAT-INSULIN))))
```

This is a nested list that starts with the symbol CULTURE-1, followed by the keyword[63] ORGANISMS:. The argument for ORGANISMS: is a list beginning with BACTERIUM-1 followed by the keyword EXOSOMES:. The EXOSOMES: argu-

63 Keywords in Lisp end with a colon and are used to structure an argument list. In some sense they are similar to keywords in Unix commands. The expression after a keyword can in a Lisp function be referred to by that keyword. The order of the keywords in an argument list

ment is another list starting with the symbol VECTOR-1 followed by the keyword GENES:. Finally, the argument for GENES: is the list (RAT-INSULIN). This is a nested frame representation that can be read as:

> "The goal is to have a culture of bacteria which carry one exosome with a gene for rat-insulin on it."

Such bacteria can produce rat-insulin in a fermenter. The variables CULTURE-1, BACTERIUM-1 and VECTOR-1 will have to be instantiated with suitable values by the Molgen program. CULTURE-1 is the name of the whole task which can be anything. BACTERIUM-1 and VECTOR-1 will hold the names of a strain of bacteria and a compatible, suitably modified vector. There are a number of implicit constraints for the two classes BACTERIUM-1 and VECTOR-1.

- Bacteria must be able to absorb the plasmid VECTOR-1. This is not given for every combination of plasmid and bacterium.
- The strain of bacteria itself should not be resistant to a certain antibiotic.
- The transformed bacteria should carry a resistance to another antibiotic to allow for screening.
- The plasmid has to carry an antibiotic resistance gene into the bacteria.
- The antibiotic resistance gene of the plasmid has to be intact in the transformed bacteria.

The last constraint is required if one does not need to discriminate between bacteria carrying a plasmid with an insert and those carrying a plasmid without an insert. None of these constraints are explicitly mentioned in the goal. They have to be inferred from background knowledge on molecular biology and from the goal description. Some of the constraints are not even explicitly contained in the knowledge base but only arise in the course of planning. The first, simple plan developed by Molgen to produce the modified bacteria according to the goal description is shown in Figure 18. It shows the requirement to merge a vector and the rat-insulin gene and then merge that hybrid vector with a bacterium. Details on which bacterium to use, which vector and how to merge are not known at this stage.

Molgen Architecture. Molgen has a three-layer control structure which is divided into *strategy space*, *design space* and *laboratory space* (Figure 19). The original goal is given to the interpreter which starts an operator in strategy space. *Strategy space* is the most abstract control layer and has four operators: FOCUS, RESUME, GUESS and UNDO. They represent the knowledge on planning strategy. The four operators are organised in two problem-solving strategies: least-commitment cycle and heuristic cycle. The *least-commitment cycle* defers making irrevocable decisions as long as possible and starts with the FOCUS operator to initiate new design steps. These are kept on an agenda in one of four states: *done*, *failed*, *suspended* or *cancelled*. Only one design step can be processed at a time, the others are suspended. If no new design steps can be generated any more, FOCUS returns control to the RESUME operator which selects one of the suspended design steps and evaluates it. If no new steps can be generated and no suspended steps can be resumed, Mol-

---

is arbitrary which makes it easy to specify only a subset of arguments and use default values for others.

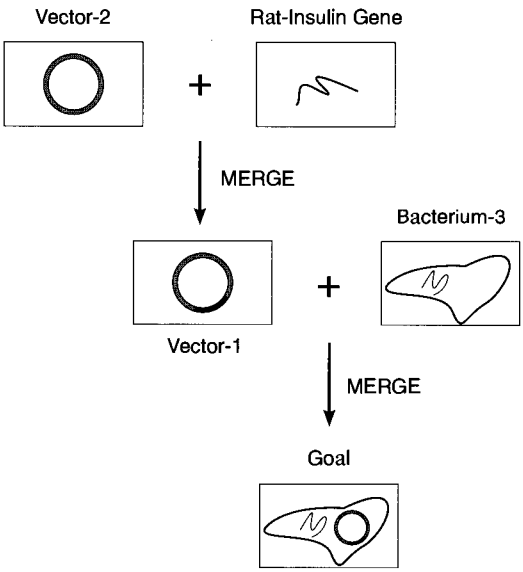


Figure 18: Original Plan for Molgen.
This is the first plan for making *E. coli* express rat-insulin which will be further refined by Molgen. VECTOR-2 and BACTERIUM-3 are variables which will be later instantiated with an actual plasmid and bacterium, respectively. VECTOR-1 is a variable from the original goal (see ref. 58).

gen leaves the least-commitment cycle and switches to the *heuristic cycle.* If at some point planning gets stuck because the problem is under-constrained the GUESS operator is selected. It finds the most promising design step which will be then evaluated. If the problem was over-constrained the UNDO operator tries to remove a previous design step and its consequences from the plan. GUESS is the least developed of the strategy operators in Molgen.

The design space knows about four types of objects: *constraint*, *difference*, *refinement* and *tuple*. *Constraints* express a relationship between plan variables. They are represented as predicates in the form of lambda-expressions in Lisp. Constraints can be used as elimination rules for object selection. A constraint is associated with a set of plan variables which refer to laboratory objects. As long as the variables are not yet bound they may be seen as a condition to be satisfied. Alternative selections that do not satisfy active constraints are eliminated. Constraints that have already been satisfied by some objects can be interpreted as commitments from the perspective of plan refinement. The least-commitment strategy tries to defer the decision about accepting a constraint as long as possible. This keeps as many options as possible open for further plan refinement because the more specific a plan becomes, the fewer choices for completion are left. Constraints are part of the description of an object and by formulating constraints about objects Molgen can make commitments about partial solutions without making specific selections for an object. Constraints can also be seen as a communication medium between subproblems

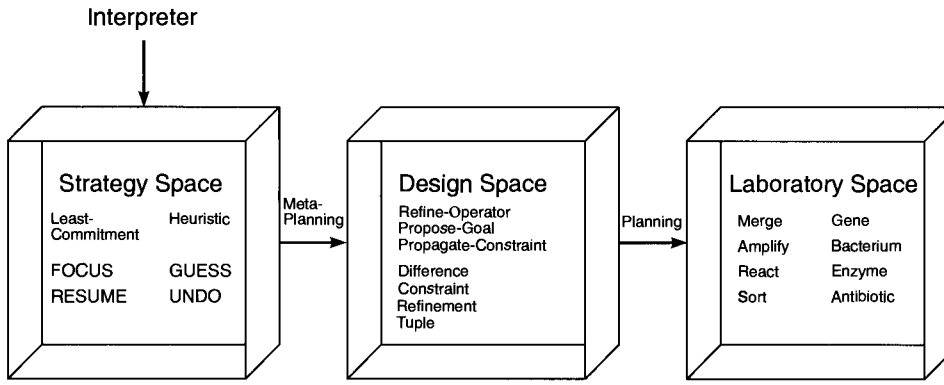


Figure 19: Planning Space in Molgen.
Strategy space has four operators. *Design space* has four different types of objects and eight operators (not all shown in the diagram). *Laboratory space* has four types of operators and 74 hierarchically organised objects (not all listed here, see ref. 58).

where they represent an intended relationship between (possibly uninstantiated) plan variables.

A *difference* in Molgen is the result of comparing an object from a goal or a partial plan with a laboratory object. This is essentially done by unification, i.e. the symbolic expressions of partially instantiated object and actual laboratory object are matched. If they contain different constants for the same argument or if the argument in one object matches a yet uninstantiated variable in the other object, these mismatches are recorded and will be returned as part of the difference description. Later, Molgen will then try to minimise the differences by modifying one of the objects or both.

A *refinement* is a specialisation of one of the four design operators of a laboratory step. For example, merging an open plasmid with a gene requires ligation while merging a plasmid and a bacterium is done by transformation. *Tuples* list all matches from the knowledge base of available objects for a set of constraints.

The constraint propagation approach in Molgen is a combination of hierarchical planning and constraint satisfaction. Three basic things can be done with constraints:

- constraint formulation,
- constraint propagation and
- constraint satisfaction.

Constraint formulation is the addition of new constraints as commitments in the design process. Hierarchical planning proceeds by first formulating more abstract constraints and only later those with increasing detail, a procedure similar to human planning. *Constraint propagation* creates new constraints from a set of old constraints in a plan. This is used for communicating requirements between subproblems in Molgen. In molecular genetics, single steps in an experiment are mostly under-constrained, e.g. there are many ways to select a vector, an endonuclease

or a strain of bacteria. Only the combination of constraints from different parts of a problem restricts the search space effectively. Sometimes, however, not all constraints can be satisfied together. *Constraint satisfaction* is done by searching a data base for objects that meet all conditions imposed by a set of constraints. If Molgen cannot find any object satisfying some constraints off the shelf, it marks these constraints as unsatisfied and later proposes to build the required object from other available resources. Thus are subgoals created.

Design space in Molgen has three types of operators: comparison operators to compare goals and compute differences, temporal extension operators to extend a plan in time and specialisation operators to make a plan more specific. There are two design *operators for comparison:* FIND-UNUSUAL-FEATURES and CHECK-PREDICTION. FIND-UNUSUAL-FEATURES examines laboratory goals by recursively comparing an actual goal with its prototype. If a *difference* is found at any level, FIND-UNUSUAL-FEATURES stops and returns a difference description. CHECK-PREDICTION takes a simulation of a proposed laboratory step and compares it with its forward goal. Sometimes both differ in which case CHECK-PREDICTION alerts to that fact. PROPOSE-OPERATORS is responsible for *temporal extension* of a plan by selecting abstract laboratory operators to reduce differences. As laboratory operators are hierarchically organised PROPOSE-OPERATORS has to consider only the most abstract ones. If more than one laboratory operator is relevant PROPOSE-OPERATORS puts them all on a *refinement* list and suspends its operation. PROPOSE-GOAL creates goals for laboratory steps, the effects of which may be subsequently simulated by PREDICT-RESULTS. The PREDICT-RESULTS operator activates a simulation model associated with a laboratory operator. The result of that simulation can be used by CHECK-PREDICTION to examine whether that step leads to the goal state or if other steps are needed.

Specialisation operators add details to a plan. The REFINE-OPERATOR selects a laboratory step to replace a (more abstract) laboratory operator. REFINE-OPERATOR is activated if there is a laboratory operator that has its goals and input completely specified, in which case a suitable laboratory step is chosen. PROPAGATE-CONSTRAINT is an operator in the design space that updates the description of earlier objects in a plan after new constraints have been introduced. If, for example, it becomes necessary that a particular bacterium be resistant to some antibiotic and it carries only one plasmid, then a new constraint must be introduced which will have to be propagated to that plasmid that it should carry an appropriate antibiotic resistance gene. REFINE-OBJECT tries to satisfy the constraints for an object each time when new constraints are introduced into the plan. Constraint satisfaction can be done by retrieving a suitable object from the laboratory space or, if there is no such object, by setting up a new goal to develop the object.

Laboratory space holds knowledge about available objects and methods in a molecular biological laboratory. Laboratory objects and operators are hierarchically organised. There are 74 objects altogether on six levels, which include, for example, ANTIBIOTIC, CULTURE, DNA, ENZYME, ORGANISM, and SAMPLE. Molgen has four abstract laboratory operators (called MARS operators from their first letter): MERGE, AMPLIFY, REACT, and SORT. MERGE can be specialised into LIGATE and TRANSFORM; AMPLIFY can be INCUBATE or GET-OFF-SHELF; REACT can be CLEAVE or ADD-HYDROXYL and SORT can be done by either ELEC-

TROPHORESIS or antibiotic SCREENing. The original knowledge base of Molgen contains enough objects for the problem on rat-insulin producing *E. coli* but for a wider use much more objects would have to be included. Operators and objects communicate via message passing as described earlier.

Example Run of Molgen. Let us now return to the example of Molgen symbolically modifying *E. coli* to produce rat-insulin. The first, abstract plan was shown in Figure 18. Molgen arrived at this point after FIND-UNUSUAL-FEATURES had compared BACTERIUM-1 of the goal with its prototype definition of a bacterium. The difference found was that BACTERIUM-1 contains a special exosome. This difference is resolved by PROPOSE-OPERATORS which suggests that MERGE can be used to produce the required bacterium. Similarly, Molgen finds the difference between a normal plasmid and the one required for the goal is the presence of a rat-insulin gene. The next step produces a *refinement*. The MERGE operator to combine the plasmid and the bacterium is replaced by the specific laboratory step TRANSFORM. Although neither bacterium nor vector are instantiated at this time the fact that a bacterium and a plasmid are to be merged is sufficient to conclude that only TRANSFORM is applicable here. As Molgen knows that transformation works only with biologically compatible plasmids and bacteria, a new constraint representing that condition is made explicit so that it can be propagated to other objects. The compatibility constraint could be also satisfied immediately by instantiating the variables BACTERIUM-3 and VECTOR-1. However, this is postponed as a consequence of the least-commitment strategy which prevents Molgen from making early decisions that would constrict a plan too soon. The situation of the plan at this time is shown in Figure 20.

In the next phase, Molgen uses the PREDICT-RESULTS and CHECK-PREDICTION design operators to simulate the transformation of bacteria with a vector. The knowledge base of Molgen contains in the frame representation of TRANSFORM the information that transformation may have two possible outcomes: one, that the plasmid is taken up by the bacterium and the other that it is not. Because a plasmid is a rather large hydrophilic molecule it does not spontaneously cross the hydrophobic cell membrane of a bacterium. To facilitate permeation the experimenter either chemically or electrically perforates the cell membrane for a short time. This must be done very carefully because otherwise the cells are completely disrupted and die. Sometimes a plasmid can enter a cell during perforation but sometimes not. PREDICT-RESULTS makes explicit the fact that two kinds of bacteria will be produced, one with a plasmid and another one without a plasmid. The current situation is shown in Figure 21.

CHECK-PREDICTION compares the result of the simulation with the desired goal and finds as a *difference* the extra bacteria without a plasmid. PROPOSE-OPERATORS suggests using the SORT operator to remove the incompetent bacteria. SORT is subsequently refined by REFINE-OPERATOR and substituted with SCREEN. The knowledge base has in its description of SCREEN the information that SCREEN can be used to separate bacteria (this made it applicable for REFINE-OPERATOR) and that an antibiotic is required in addition. Consequently, Molgen has to introduce a new object into the plan, an antibiotic labelled by the variable ANTIBIOTIC-1. At this time, ANTIBIOTIC-1 is an uninstantiated variable. Molgen knows an antibiotic will have to be used here but defers the decision for

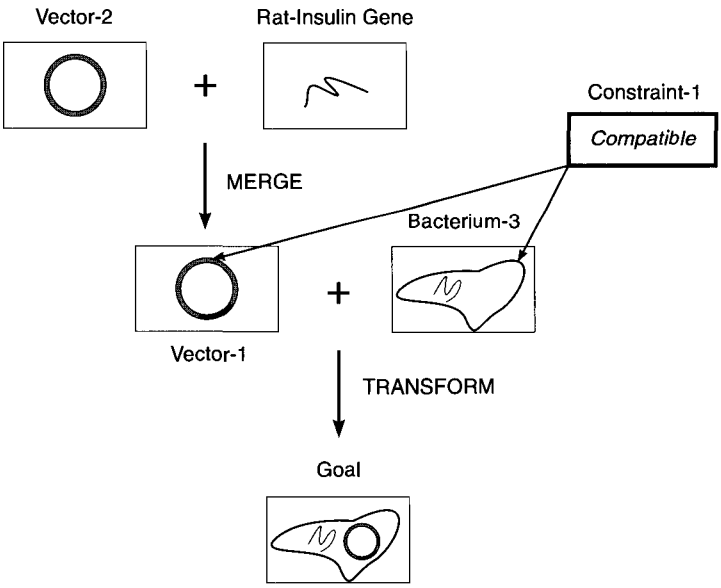


Figure 20: Introducing a Constraint.
At this stage, Molgen has refined the MERGE operator into a TRANSFORM step and introduced a new constraint (CONSTRAINT-1) into the plan (see ref. 58).

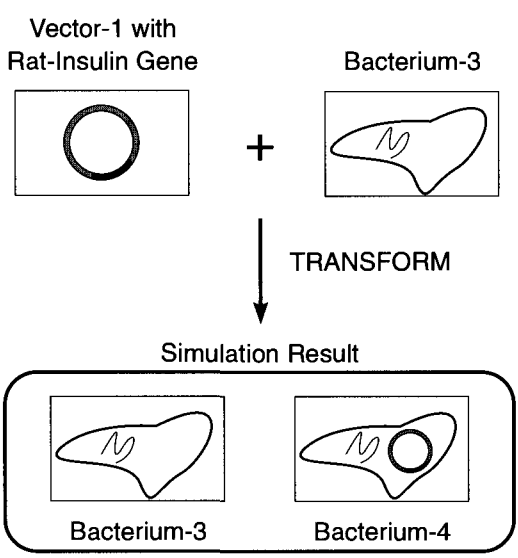


Figure 21: Simulation of Transformation.
Symbolic simulation of transformation of bacteria reveals that an unwanted component BACTERIUM-3 is found in the result of TRANSFORM. BACTERIUM-3 lacks the desired hybrid plasmid (see ref. 58).

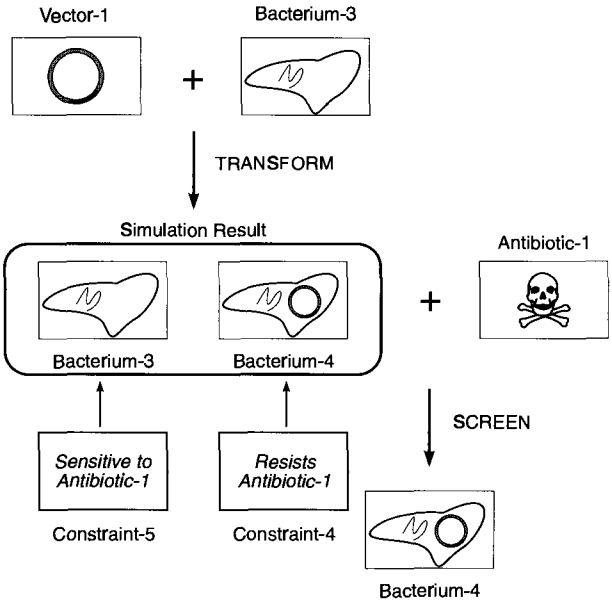


Figure 22: Introduction of an Antibiotic into the Plan.
Having selected SCREEN to separate the two types of bacteria, Molgen introduces the variable ANTIBIOTIC-1 (see ref. 58).

any particular antibiotic until later. Also, as a consequence from the application of SCREEN, two more constraints are introduced. To save BACTERIUM-4 but to remove BACTERIUM-3 SCREEN requires the first to be resistant against the antibiotic (CONSTRAINT-4) and the latter to be sensitive (CONSTRAINT-5). The situation is depicted in Figure 22.

Planning can be seen as a *generate-and-test process* where hypotheses are generated on how to accomplish the desired goal. These hypotheses must then be evaluated to check whether they actually achieve the goal. Constraint propagation in Molgen helps to keep the number of hypotheses small. Constraints that effect some objects in one subgoal may carry implications for other objects. These implications are made explicit so they can act to further restrict the number of plausible hypotheses. One instance of constraint propagation that occurs in the *E. coli* rat-insulin example is the propagation of the requirement for a gene coding for an antibiotic. In the situation we just left Molgen a molecular biologist would perform the following reasoning.

"To carry out antibiotic screening in the current situation requires the desired bacteria (those with a plasmid; BACTERIUM-4) to be resistant against an antibiotic, whereas the others (BACTERIUM-3) should lack this resistance. Genetic information for expression of antibiotic resistance is not coded on the chromosome of *E. coli*, hence it must be provided by an extra-chromosomal element. The only exosome which occurs in BACTERIUM-4 but not in BACTERIUM-3 is VECTOR-1. Therefore, VECTOR-1 should carry the antibiotic resistance gene. VECTOR-1 was made from VECTOR-2 and the rat-insulin gene. Rat-insulin

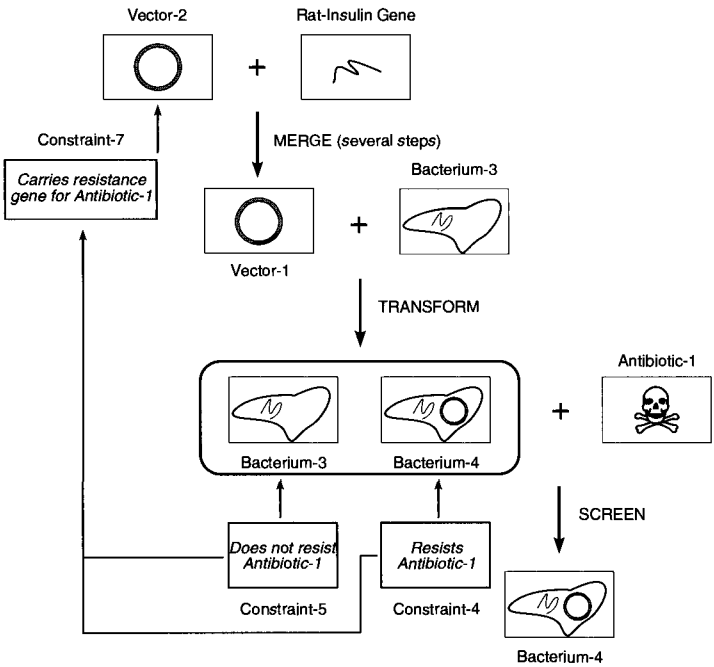


Figure 23: Propagating CONSTRAINT-7.
Constraints are propagated one step at a time in Molgen (see ref. 58).

does not confer antibiotic resistance and so I must conclude that VECTOR-2 will have to provide the antibiotic resistance gene."

The conclusion of the previous argument is the new constraint CONSTRAINT-7 which is propagated back from the SCREEN operator to VECTOR-2. Figure 23 shows the situation after propagation of CONSTRAINT-7. Now, REFINE-OPERATOR is called to instantiate the laboratory objects BACTERIUM-3 and VECTOR-1 based on the constraints CONSTRAINT-1, CONSTRAINT-4, CONSTRAINT-5 and CONSTRAINT-7.

To finish the plan Molgen had to refine the MERGE step to combine VECTOR-2 and the rat-insulin gene. This is done by taking rat-insulin gene off the shelf and ligating it with a suitable linker, in this case Hind3decamer which happened to be the only linker included in Molgen's knowledge base. The linker and the vector must then be cut by HindIII restriction enzyme and ligated together. The completed plan is shown in Figure 24.

Based on the limited knowledge base Molgen generated four compatible solutions for bacterium, vector, antibiotic, enzyme and linker. One of them (*E. coli* transformed with plasmid pMB9 carrying tetracycline resistance and a HindIII

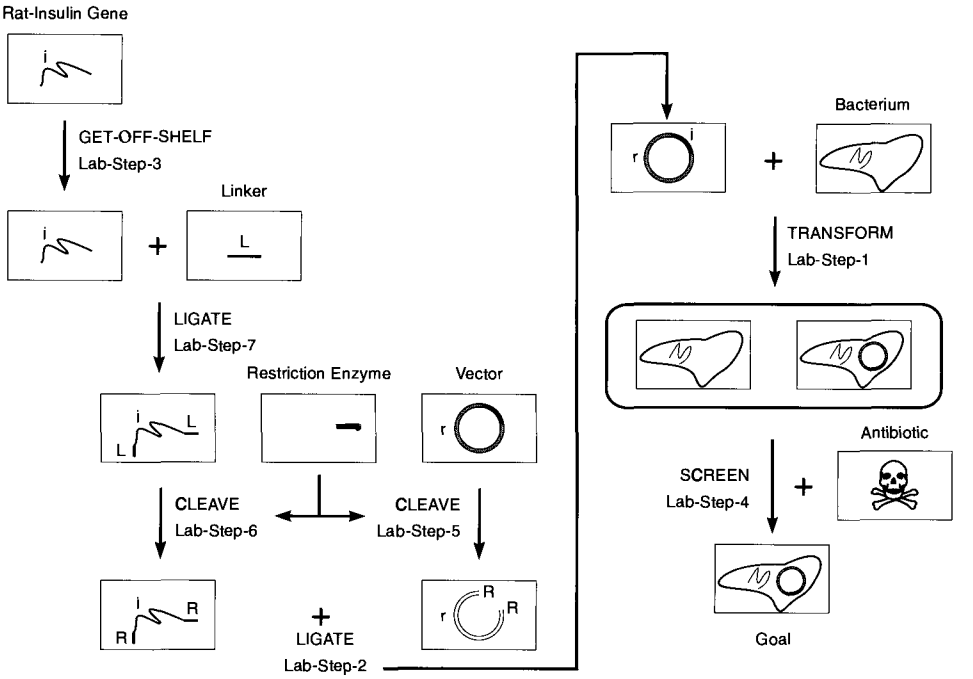


Figure 24: Final Plan.
This plan shows all laboratory steps (capitalised) that Molgen found necessary to accomplish the goal. Numbering of the laboratory steps reflects the order in which the steps were considered during planning, not the order of execution. "L" denotes the linker Hind3decamer, "i" identifies the rat-insulin gene, "R" is the open restriction site after treatment with HindIII and "r" is a label for antibiotic resistance (see ref. 58).

restriction site) had been previously reported in a molecular genetics research paper[64].

Summary. In its planning activity Molgen considered and satisfied 12 constraints that were not explicitly mentioned in the goal but were discovered during planning based on the definition of laboratory objects and procedures. These are the 12 constraints: 1) Bacterium and vector have to be biologically compatible. 2) The vector and 3) the rat-insulin gene must both have sticky ends prior to ligation. 4) Bacteria carrying the vector should be resistant against an antibiotic 5) while those without the vector should be not. 6) The vector used for transformation should carry an antibiotic resistance gene and therefore 7) must be made from a vector carrying such a gene. 8) The restriction enzyme must not cut the antibiotic resistance gene or 9) the rat-insulin gene. 10) However, the vector to carry the rat-insulin gene must have a restriction site for that restriction enzyme. 11) The DNA

64 A. Ullrich, J. Shine, J. Chirgwin, R. Pictet, E. Tischer, W. J, Rutter, H. M. Goodman, *Rat-insulin genes: construction of plasmids containing the coding sequence*, **Science**, vol 196, pp. 1313-1319, 1977.

Constraint	Total	Considered	Antibiotic	Bacterium	Enzyme	Linker	Vector
(none)	3456	0	9	3	32	1	4
compatible	1152	4	9	**1**	32	1	**4**
resistance	160	5	**3**	1	32	1	**4**
enzyme cuts	21	21	3	1	**6**	1	**4**
does not cut	10	10	**2**	1	**6**	1	4
enzyme cuts	4	4	2	1	**1**	**1**	3

Table 4: Effectiveness of Constraint Posting.
This table shows the number of quintuple combinations (bacterium, vector, antibiotic, enzyme, linker) depending on the constraints imposed. Each line tells how many choices there are for bacterium, vector, antibiotic, enzyme, and linker and the product of all (column "Total"). The numbers in column "Considered" say how many combinations Molgen tested for that constraint. The numbers in bold face indicate objects that are bound by a constraint. The effect of constraints is accumulated down the table. Constraints are listed in their order of appearance during the planning experiment. Although the last constraint does not involve the vector directly, selection of the linker eliminates five enzymes. This in turn excludes one of the vectors which was only compatible with those five enzymes (see ref. 58).

that carries the rat-insulin gene to be ligated with the vector must have sticky ends complementary to the ends of the vector. 12) The linker attached to the rat-insulin gene must have a restriction site identical to the one of the vector.

Table 4 shows how constraint propagation reduces the number of combinations in the search space of suitable plans for accomplishing the rat-insulin goal. Without any constraints the knowledge base of Molgen presents 3456 combinations for a quintuple of bacterium, vector, antibiotic, enzyme and linker. The table shows how this number is reduced by the constraints to a total of *four* valid combinations. The actual benefit of constraint propagation in this example is even greater because several variables for each of the five objects occur during different stages of planning.

There are some comments on Molgen in general and its performance on the *E. coli* rat-insulin example in particular.

1) The knowledge base does not account for the fact that like transformation, ligation also does not perform at a rate of 100%. Therefore, some vectors may be missing the rat-insulin gene and only reconnect their own ends or those of some other plasmid. These incomplete plasmids can not cause expression of rat-insulin in bacteria. However, as some fraction of the plasmids will take up the rat-insulin gene some of the transformed bacteria that show antibiotic resistance will also produce rat-insulin but not all of them. To remove those bacteria with incomplete plasmids a procedure with two antibiotics has to be used, as describe at the beginning of this chapter. Alternatively, different restriction enzymes for both ends of the DNA insert and of the vector could be used to guarantee specificity in the ligation step.

2) Today, there are certainly many more linkers available than the one in Molgen's knowledge base. Other linkers and enzymes have to be included to produce meaningful results.
3) Constraint propagation is only useful for nearly decomposable problems with nearly independent subproblems. If a goal contains independent subproblems with no constraints between them or if it requires a strictly sequential ordering constraint propagation cannot do much.
4) Molgen's implementation of constraint satisfaction and constraint propagation is based on purely syntactic matches. Logically equivalent predicates can not be recognised. In practice, this leads to extremely long run times as semantically proper constraints can sometimes not be applied because of syntactic differences.
5) Molgen uses constraints to test a situation for its plausibility. It does not, however, use constraints to actually *guide* planning. Plan generation is done independently of the constraints.
6) Molgen uses constraints only on objects, not on processes. An extension to include partial process descriptions which can be refined during planning could allow smoother plan refinement.
7) No meta-constraints are allowed, i.e. one can not express a condition like "use a maximum of 15 steps".
8) Time is not explicitly represented in Molgen. This has the disadvantage that the same object has to be identified by different variable names in different stages of a plan. Changes to one object over time can not be made explicit. Also, maintaining records about different worlds is not possible, only sequential examination of different plans.
9) The knowledge engineering bottleneck. To be of practical value it is vital for Molgen's knowledge base to contain all important details about at least one domain in molecular genetics. Obviously, missing a fact may result in failure to formulate some necessary constraint and end up proposing experiments that will not work.

Despite its weaknesses Molgen demonstrates how in principle a computer can be used to generate plans for the production of genetically modified organisms. With the proper extensions this approach could be used for step 3 ("Planning experiments" in chapter 2.2.1) as one component of an "intelligent" assistant for the molecular biologist.

2.2.5. Expert Systems for Protein Purification

Purification of proteins is an important task in basic research and biotechnology. In the laboratory pure enzymes are needed for catalysing various biochemical reactions. Concentrated solutions of pure protein are used to grow crystals for X-ray structure determination. In medicine, highly purified proteins act as therapeutic agents. Protein purification can be difficult and costly, depending on the protein and on the desired purity. A *downstream process* is a series of separation steps that starts immediately after protein synthesis and that leads to a purified protein prod-

uct. The optimal process provides a product of specified quality, in terms of purity and consistency, at the lowest possible cost. The wide range of protein characteristics provides many options for a single purification step. A successful design of an efficient downstream process may be a fundamental precondition for the realisation of a research project or an industrial effort to commercialise a product (both with strong economic implications for all individuals involved). Since there is no generally applicable schema for processing a solution containing one desired protein among many other constituents a number of heuristics have been used to guide protein purification. Heuristics for downstream processing of multi-component mixtures can be broadly classified into four categories[65,66].

1) *Method heuristics* are based on the description and comparison of the applicability of unit operations. For example, the advantage of centrifugation over ultrafiltration for yeast but not for bacteria is such a heuristic. Other method heuristics are: remove mass separating agents first; avoid refrigeration and vacuum; favour ordinary destillation.
2) *Species heuristics* focus on the properties of compounds in the mixture. Heuristics considering the nature of impurities are: perform easy separations first; perform high-recovery separation last; remove corrosive and hazardous components first.
3) *Design heuristics* specify the order of process steps, e.g. perform a low resolution step before a high resolution step. Other design heuristics are: remove desired product last; favour direct sequence; use the cheapest separator first; favour the smallest product set.
4) *Composition heuristics* decide on a technique based on the greatest difference in properties of product and impurities, e.g. favour 50/50 split and remove the most plentiful component first.

Expert systems can include knowledge about all four types of process design and provide a means to implement these and other rules on a computer. In combination with a data bank of chemo-physical properties of chemical compounds and with simulation models of single purification steps such a system can generate suggestions to the experimenter with explanations and a justification based on a large knowledge base of rules governing downstream process design and on knowledge about the constituents of the current sample.

An educational computer program distributed by IRL Press[67] simulates some basic purification methods (ammonium sulphate precipitation, heat treatment, gel filtration, ion exchange chromatography, chromatofocusing, isoelectric focusing, and SDS-PAGE). The user starts with a solution containing 20 proteins and is chal-

65 V. M. Nadgir, Y. A. Liu, *Studies in chemical process design and synthesis*, **AIChE J.**, 29, pp. 926-934, 1983.

66 Y. A. Liu, *Process synthesis: some simple and practical developments*, in **Recent Developments in Chemical Process and Design**, (Y. A. Liu, H. A. McGee Jr, W. R. Epperly Eds.), Wiley-Interscience, New York, 1987.

67 **Protein Purification** by A. G. Booth, for IBM-PC compatibles and Macintosh, Oxford Electronic Publishing, Oxford University Press, Walton Street, Oxford OX2 6DP, England, tel + 44 865 56767, 1993. Internet World Wide Web Server of Oxford University at http://www.ox.ac.uk.

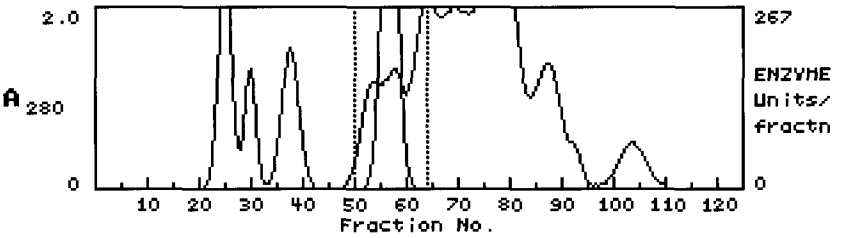
Protein Purification Sample Record

Step	Procedure	Protein (mg)	Enzyme (units)	Enzyme yield(%)	Enrichment	Cost (m.h./100U)
	Initial	511.0	3600	100.0	1.0	
1	Gel Filtration	227.3	3598	100.0	2.2	0.19
2	Isoelectric Foc.	83.7	3598	100.0	6.1	0.81
3	Amm. Sulphate	50.0	3578	99.4	10.2	1.15

Table 5: Purification Record.
The effects of three subsequent purification steps are summarised. "Protein" shows the total amount of protein including the desired enzyme. "Enzyme" shows how much specific enzyme activity was retained after the last step. A unit of enzyme activity is the quantity of enzyme that is needed to catalyse the transformation of 1.0 μ mol (10^{-6} mol) substrate at optimal conditions and 25°C. "Enzyme yield" shows how much enzyme is kept and lost during purification. "Enrichment" is the ratio of enzyme (in units) divided by protein (in mg) and normalised to the initial values. "Cost" is given in man hours per 100 units of enzyme. This type of purification summary is also generated by the Protein Purification program[67].

lenged to purify them all. At first, this is done by trial and error until the experimenter learns about the strengths and weaknesses of individual purification steps. From there she develops strategies for which methods to prefer and in what order to apply them. Time and expenses are limited and monitored. The benefit of the program is twofold. First, it helps to gather some experience on the applicability of certain purification steps based on a mathematically reliable simulation and second, it does so at a fraction of the cost of real experiments. The disadvantages are, of course, that there is only a limited number of purification methods to select from and that the program cannot be used for proteins other than the 20 preprogrammed ones. No practical experience is gained for how to perform a technique in the laboratory. The later problem is inherent to all theoretical simulation models but the first two can be addressed by expert system technology as will be demonstrated later in this chapter.

Protein Properties. Proteins are made from α-amino acids. There are 20 naturally occurring α-amino acids which serve as the basic structural units of a protein. Each amino acid consists of an amino group ($-NH_2$), a carboxyl group (–COOH), a hydrogen atom (H) and a specific side chain (R), all bonded to one carbon atom, called the α-C atom (short: Cα). The side chains vary in size, shape, charge, hydrogen-bonding capacity and chemical reactivity. It is the variety of side chain properties that gives rise to a large number of different protein structures and protein functions. Amino acids in solution at neutral pH are predominantly dipolar ions. They are said to be protonated because they carry an extra positive charge in the form of a hydrogen cation (H^+). The carboxyl group carries an extra negative charge in form of a single electron. The state of ionisation varies with pH. Seven amino acids have also an ionisable side chain.



Situation after step 1	
Total protein	97.9 mg
Total enzyme	2992 units
Enrichment	5.2
Yield of enzyme	99.7%
Cost so far	0.23 man-hours/100U

Press the space bar to continue.

Figure 25: Simulation of Gel Filtration.
This screen is generated by the Protein Purification[67] program. In the upper half, fractionation and optical density measurement of eluate coming from a gel filtration column is simulated. The simulation is based on the composition of the input mixture (known to the programmer) and on the physics of gel filtration. The area enclosed by dotted lines in the middle shows the curve of an enzyme activity assay. Fractions 52 to 61 contain the desired enzyme. These fractions will have to be pooled for further purification because one can see from the optical density plot that there is more than one protein in these fractions. The lower part summarises the simulated purification.

The interesting properties of a protein with respect to purification are those that can be exploited by a separation process. Purification operations can be divided into three broad categories, each employing different aspects of a protein. These are 1) the presence of chemical groups which render the protein specific properties. For example, glycosylated proteins can be isolated by adsorption on a heparin column. 2) The three-dimensional conformation of a protein can make it bind specifically to a biological substrate or to an immobilised antibody on a column. 3) Non-specific, collective properties that result from the unique combination of amino acids in each protein. Among the latter are: size, shape, charge, pK and hydrophobicity of the protein. When designing a downstream process it is assumed that substituent groups and affinity to biological substrates or antibodies are known. Non-specific properties can be either measured empirically or estimated theoretically. If available, more confident results are obtained by using an experimental method instead of a theoretical approximation, especially in the case of large proteins.

The size of a molecule can be approximated by its molecular weight or calculated as the sum of all constituent amino acid residues. Hydrophobicity of a single amino acid can be either empirically calculated or measured as the change in free energy of short oligopeptides being transferred from organic solvent into wa-

68 *Alchemy III for IBM-PC and Macintosh*, **Tripos Associates** (Evans & Sutherland), 1699 South Hanley Road, Suite 303, St. Louis, Missouri 63144, tel (USA) 314 – 6471099, 1993.

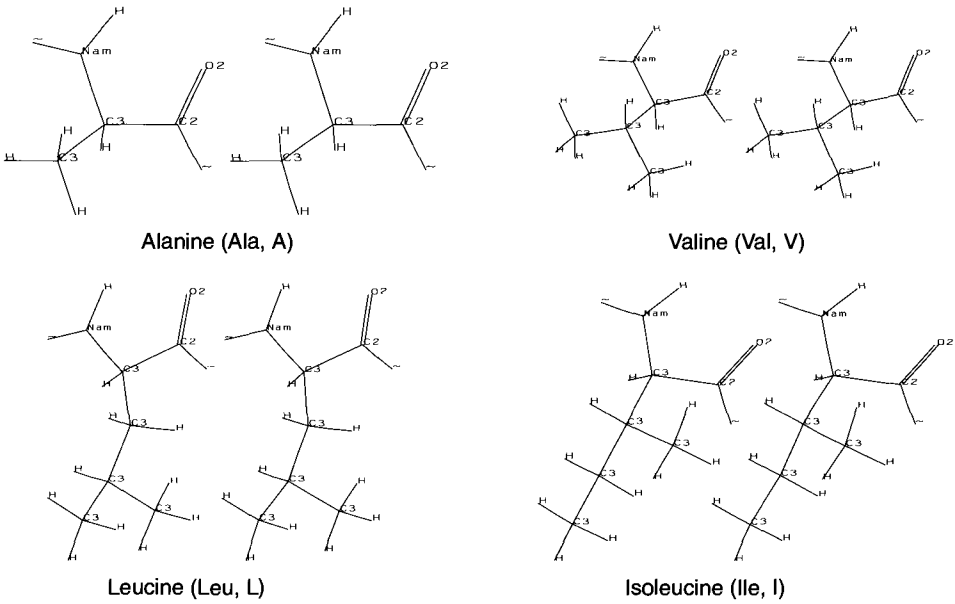


Figure 26: Hydrophobic Amino Acids.
The four amino acids alanine (Ala, A), valine (Val, V), leucine (Leu, L) and isoleucine (Ile, I) are shown in stereo projection. The three-letter and one-letter codes are given in parentheses. To perceive a three-dimensional image the reader has to superimpose both halves of a diagram by either converging or diverging the two eyes (the latter produces a mirror image of the actual conformation). Alternatively, stereo glasses may be used which allow one eye to look at one half of a stereo-pair only. The stereo-diagrams were created with the Alchemy III program[68]. There is rotational freedom around single bonds which have one line but not around double bonds which have two lines. Atom labels here and in the following figures are as follows: C3 is a carbon atom with sp3 orbitals; C2 is carbon with sp2 orbitals; CAR is aromatic carbon; NAM is amide nitrogen atom; N3 is nitrogen with sp3 orbitals; N2 is nitrogen with sp2 orbitals; NPL3 is planar trigonal nitrogen; S3 is sulphur with sp3 orbitals; O2 is oxygen with sp2 orbitals; O3 is oxygen with sp3 orbitals; H is hydrogen. The "~" sign marks terminal bonds that are extended in the formation of a polypeptide chain. The four amino acids alanine, valine, leucine and isoleucine are mainly hydrophobic and therefore occur preferably in the interior of proteins.

ter. The sum of free energy changes over all residues in a protein determines its total hydrophobicity. There are various hydrophobicity scales. Some are based on the solvent accessible surface area of a residue, others on the free energy change of phase transfer. It remains an open question whether surface area calculations and

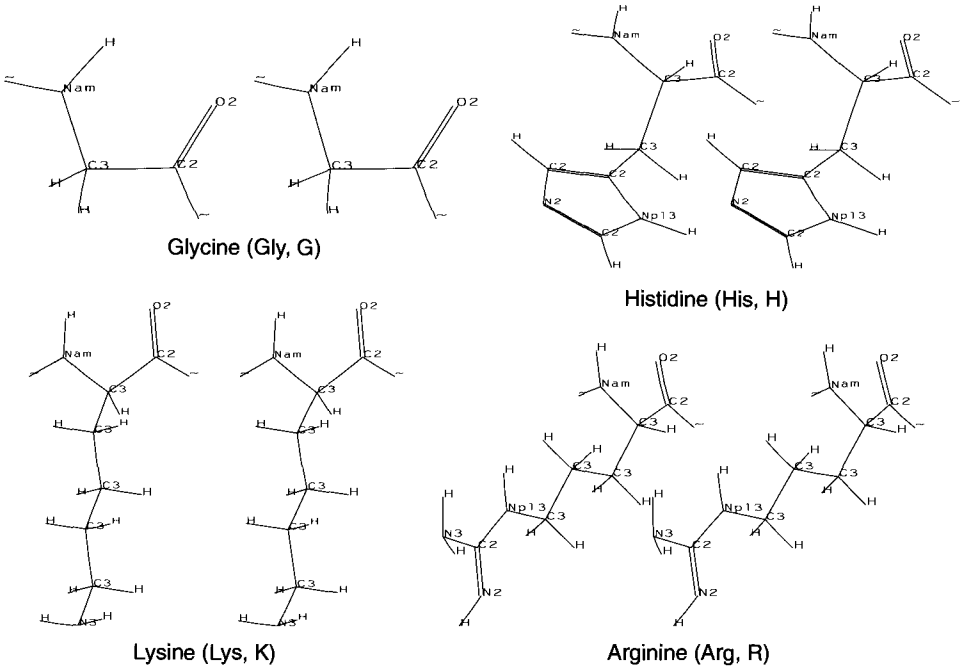


Figure 27: Basic Amino Acids.
The three basic amino acids histidine (His, H), lysine (Lys, K), and arginine (Arg, R) are shown next to the smallest amino acid glycine (Gly, G). Glycine has no side chain. The basic amino acids can take up a positive charge in form of a hydrogen cation H^+ at their N3 or N2 atoms. They are hydrophilic and easily participate in chemical reactions which is why they are often found in the active site of enzymes. (See also legend of Figure 26).

energy measurements are linearly correlated[69,70]. Four hydrophobicity scales are shown in Table 6.

The shape of a protein can be estimated from its apparent frictional coefficient in gel electrophoresis or ultracentrifugation if its mass is known. Charge and pK can

69 R. H. Wood, P. T. Thompson, *Differences between pair and bulk hydrophobic interactions*, **Proceedings of the National Academy of Sciences U.S.A.**, vol 87, pp. 946-949, 1990.
70 I. Tunon, E. Silla, J. L. Pascual-Ahuir, *Molecular surface area and hydrophobic effect*, **Protein Engineering**, vol 5, no 8, pp. 715-716, 1992.
71 J.-L. Fauchere, V. E. Pliska, *Hydrophobic parameters π for amino acid side chains from partitioning of N-acetylamino acid amids*, **Eur. J. Med. Chem.-Chem. Ther.**, vol 18, pp. 369-375, 1983.
72 G. J. Lesser, G. D. Rose, *Hydrophobicity of amino acid subgroups in proteins*, **Proteins: Structure, Function and Genetics**, vol 8, pp. 6-13, 1990.
73 D. Eisenberg, R. M. Weiss, T. C. Terwillinger, *The hydrophobic moment: a measure of the amphiphilicity of α-helices*, **Nature**, vol 299, pp. 371-374, 1982.
74 J. Cornette, K. Cease, H. Margalit, J. Spouge, J. Berzofsky, C. DeLisi, *Hydrophobicity scales and computational techniques for detecting amphipathic structures in proteins*, **Journal of Molecular Biology**, vol 195, pp. 659-685, 1987.

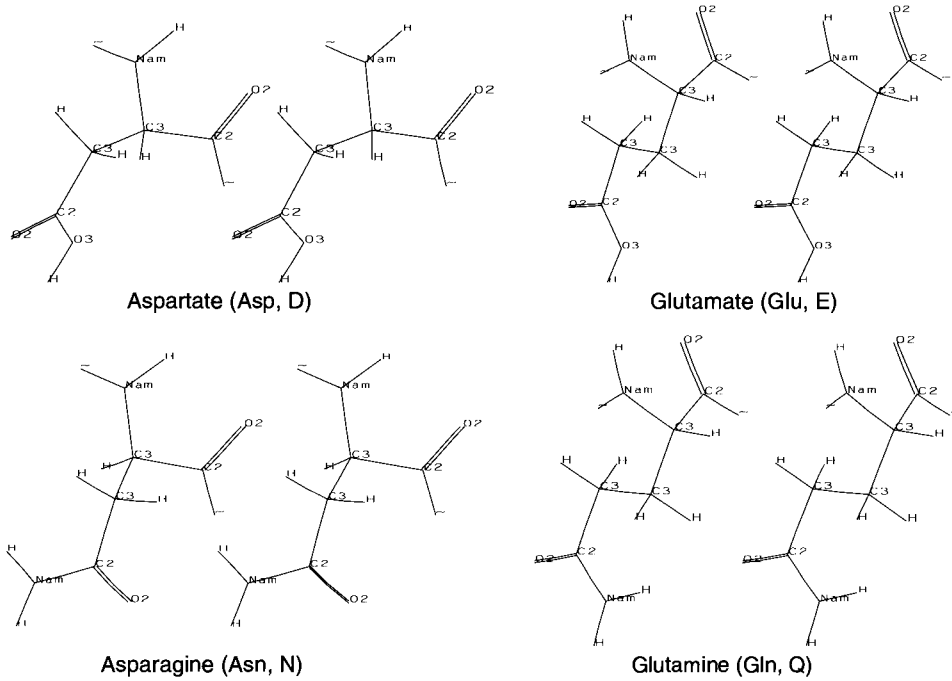


Figure 28: Acidic Amino Acids.
Aspartate (Asp, D), glutamate (Glu, E), asparagine (Asn, N), and glutamine (Gln, Q) carry an extra carboxyl-group (–COOH) which can donate a hydrogen cation (H^+) and become negatively charged. These amino acids are very hydrophilic and primarily located at the outside of proteins. (See also legend of Figure 26).

be measured by titration: the pH[75] of a protein solution of defined concentration is monitored as a base (or acid) is added in small quantities. The quantity of base (or acid) added is then plotted over the measured pH. The resulting titration curve has an inflection point which marks the pK of the protein. In a solution with pH = pK half of the protein is dissociated. The charge of a protein can be determined by how much base (or acid) was needed to change the pH by one unit.

Purification Methods[76]. Here some of the most common methods for protein purification are briefly explained. *Ammonium sulphate precipitation* is used for salting-out a protein from solution. Salt at high concentrations changes the structure of the solvent which is then believed to exert pressure on the protein to hide its hydrophobic stretches from the solvent through aggregation and precipitation. The salt concentration for salting-out must be individually determined for each protein species. Thus, salting-out can be used to fractionate proteins. Ammonium sulphate is used because it is chemically inert with proteins. Typical salt concentra-

75 The pH of a solution is a measure of its concentration of H^+ ions. It is defined as pH = $\log_{10}(1/[H^+]) = -\log_{10}[H^+]$ with $[H^+]$ as the concentration of H^+ ions stated in mol/litre.
76 T. G. Cooper, **The Tools of Biochemistry**, Wiley-Interscience, 1977.

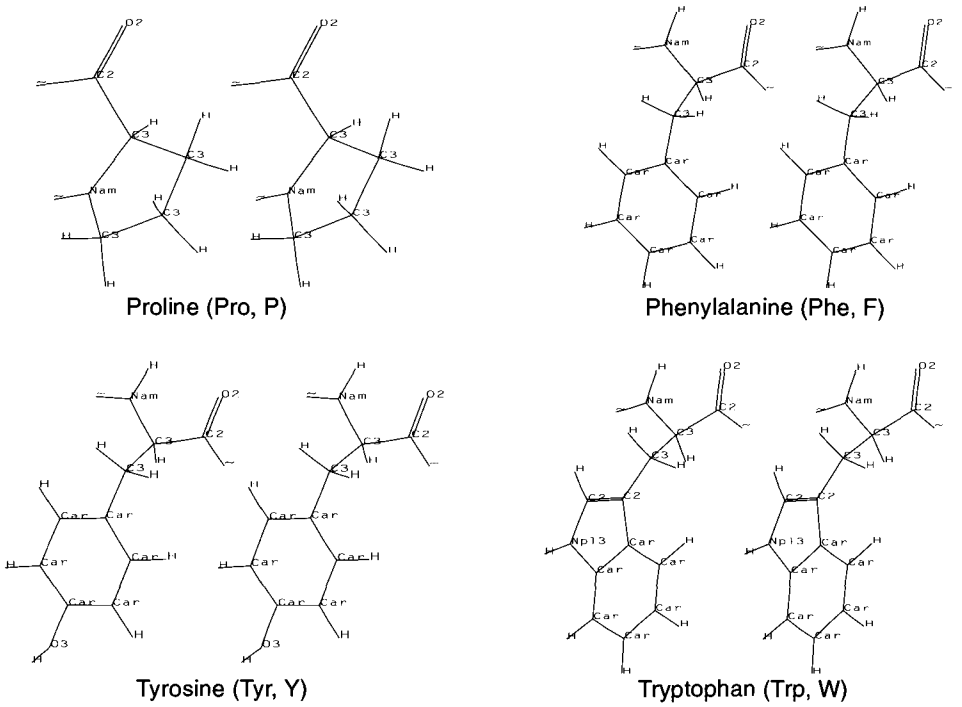


Figure 29: Aromatic Amino Acids and Proline.
Phenylalanine (Phe, F), tyrosine (Tyr, Y) and tryptophan (Trp, W) have aromatic side chains. They are rather hydrophobic and tend to occur in the core of proteins. Proline is special because its side chain bends back again to the main chain. Therefore, the torsion angles between NAM and C3 (Cα) and between C3 (Cα) and C2 are constrained in proline. Because of its inflexible backbone proline interrupts secondary structures. (See also legend of Figure 26).

tions range from 0.5 to 3 mol/l. Other agents for precipitation are organic solvents which precipitate proteins in their order of hydrophilicity or polyelectrolytes which precipitate proteins by size. Salting-out can also be used to concentrate dilute protein solutions. This effect should not be confused with *salting-in*, which *increases* the solubility of a protein at low ionic strength and low salt concentration and which does not depend on a particular type of salt. Salting-in increases solubility by decreasing the electrostatic free energy of a protein. Salting-out is a relatively cheap purification method but not very selective. There is no general mathematical or physical model for salting-out.

Heat treatment in the range of 30° – 100°C selects proteins which are more stable at high temperatures than others. The disposition of a protein to denature and degrade at a certain temperature is based on its size, composition and conformation and therefore can be used to separate it from other proteins. Heat treatment is relatively inexpensive separation method but not very specific. It must be carried out with much care to avoid destroying the protein product.

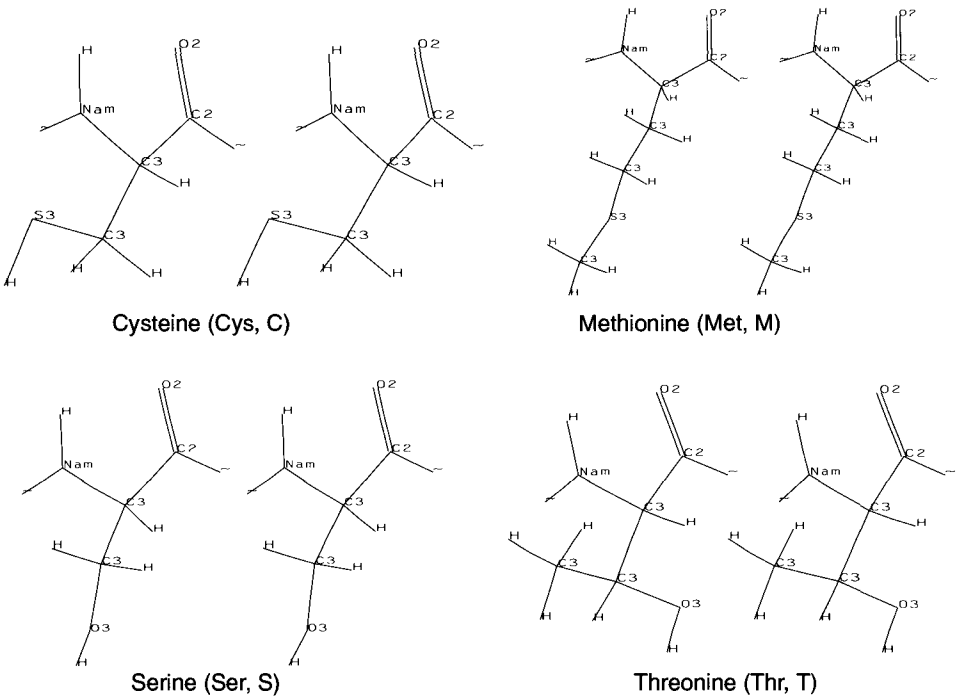


Figure 30: Polar Amino Acids and Amino Acids with Sulphur.
Cysteine (Cys, C) and methionine (Met, M) have one sulphur atom in their side chains. Two cysteine residues can form a disulphide-bridge. Serine (Ser, S) and Threonine (Thr, T) have polar side chains containing an oxygen atom which makes them chemically more reactive then their dehydroxylated counterparts alanine and valine. Like basic and acidic amino acids these residues tend to occur on the outside of a protein. (See also legend of Figure 26).

Gel electrophoresis separates proteins according to their shape, size, mass and charge. There are many variations of gel electrophoresis. The one most widely used is known as Sodium-DodecylSulfate-PolyAcrylamid Gel Electrophoresis or short SDS-PAGE. A protein sample is placed on a chemically inert polyacrylamid gel. The pore size of the gel can be adjusted during polymerisation so that it can be designed to function as a molecular sieve which retains large proteins and allows small ones to pass. The buffer in the gel and protein solution contains the anionic detergent SDS ($H_3C-(CH_2)_{10}-CH_2OSO_3^- - Na^+$). Mercaptoethanol is also added to reduce and break any disulphide bonds between cysteine residues. SDS causes the protein to unfold and adopt an approximately extended conformation with on the average one SDS molecule attached per two residues. The resulting complex of SDS and denatured protein carries a negative charge which is approximately proportional to the mass of the protein. The gel is then loaded with the sample and an electric field applied. The denatured, negatively charged proteins migrate to the anode with the velocity v:

Residue	π	rank	f	rank	Consensus	rank	PRIFT	rank
Ala	0.308	11	0.73	11	0.25	14	0.22	10
Arg	-1.005	7	0.62	4	-1.76	10	1.42	13
Asn	-0.602	3	0.63	7	-0.64	6	-0.46	2
Asp	-0.770	5	0.61	3	-0.72	8	-3.08	8
Cys	**0.983**	**14**	**0.89**	**20**	**0.04**	**12**	**4.07**	**15**
Gln	-0.220	2	0.63	6	-0.69	7	-2.81	6
Glu	-0.638	4	0.58	2	-0.62	5	-1.81	3
Gly	0.000	8	0.69	10	0.16	13	0.0	9
His	0.132	9	0.74	12	-0.41	4	0.46	11
Ile	**1.805**	**19**	**0.87**	**17**	**0.73**	**20**	**4.77**	**19**
Leu	**1.702**	**17**	**0.87**	**16**	**0.53**	**17**	**5.66**	**20**
Lys	**-0.990**	**6**	**0.51**	**1**	**-1.10**	**9**	**-3.04**	**7**
Met	1.232	16	0.84	14	0.26	15	4.23	16
Phe	1.790	18	0.88	19	0.61	19	4.44	17
Pro	**0.720**	**12**	**0.62**	**5**	**-0.07**	**1**	**-2.23**	**5**
Ser	**-0.040**	**1**	**0.64**	**8**	**-0.26**	**3**	**-0.45**	**1**
Thr	0.257	10	0.68	9	-0.18	2	-1.90	4
Trp	**2.252**	**20**	**0.87**	**18**	**0.37**	**16**	**1.04**	**12**
Tyr	0.961	13	0.77	13	0.02	11	3.23	14
Val	1.218	15	0.86	15	0.54	18	4.67	18

Table 6: Four Hydrophobicity Scales for Amino Acids.
For each amino acid residue four measures of hydrophobicity are listed. The first index in row "π" is the logarithm of the distribution ratio[71] of an amino acid between $C_8H_{17}OH$ and water, normalised to the value of glycine. The second index ("f") is the mean fraction buried[72] i.e. the difference between the solvent accessible surface area of an amino acid residue in the tripeptide Gly-X-Gly and the average from a set of 61 proteins. The "Consensus" column is a combination of five different scales[73]. The fourth scale is the PRIFT scale[74]. Each of these indices can be used to estimate the overall hydrophobicity of a protein of known primary structure but unknown conformation. Because the four scales were derived by different methods they rank individual amino acid residues differently according to their hydrophobicity. The residues with highest and lowest hydrophobicity indices in any one of the scales are highlighted in bold type face.

$$v = \frac{E \cdot q}{d \cdot f} = \frac{E \cdot q}{d \cdot 6 \cdot \pi \cdot r \cdot \eta}$$

E is the electrical potential and d the distance between the electrodes; q is the net charge of the protein-SDS complex and f its frictional coefficient which can be substituted by the product of two constants, the radius r of an approximately spherical molecule and the viscosity η of the solution. Proteins of different size carry different amounts of SDS which gives them a negative charge roughly proportional to their size. Hence, large proteins move faster than small ones because they carry a greater electrical charge. After migrating for some time, the proteins can be made visible by various staining techniques. The desired bands are then cut out and dialysed. The mobility of most polypeptide chains is linearly proportional to

the logarithm of their mass[77]. Membrane proteins and proteins with a large carbohydrate content migrate anomalously. SDS-PAGE is a rapid and sensitive separation method of high resolution but can process only small quantities. A variation of gel electrophoresis is *isoelectric focusing*. Here, a stable pH gradient is automatically established in a gel through a mixture of polyampholytes in an electrical field. Polyampholytes are polymers with different electrical charges. In an electrical field those with the most positive charges migrate first to the cathode where they increase the pH by supplanting the H^+ cations in this area. They also decrease the strength of the electric field. Then, the next weaker species of polyampholytes is attracted and so on until a continuously decreasing pH gradient from cathode to anode is established. When a protein sample is put onto the gel, the protein moves to a place where its charge is compensated by the surrounding pH. The location where the protein is immobilised is called its *isoelectric point*. Thus, proteins can be separated by their native net charge without denaturing them. SDS-PAGE and isoelectric focusing are often applied one after another, the first to separate proteins by mass and the second by charge. This is called *2D gel electrophoresis*.

Gel filtration separates proteins according to their size. The sample is applied to the top of a column which is loaded with beads of insoluble but highly hydrated polymers on either carbohydrate or polyacrylamide basis. The beads are typically 0.1 mm in diameter and porous. Small proteins can enter these beads but not large ones. Therefore small proteins have a larger volume accessible to them and move more slowly to the bottom of the column. Large molecules which cannot enter the beads arrive first at the end and can be fractionated. Gel filtration chromatography can handle larger quantities of protein than gel electrophoresis but separates at a lower resolution. The retention of a substance on a gel filtration column can be described by three indices:

$$V_e\% = \frac{V_e}{V_0}, \qquad R = \frac{1}{V_e\%} = \frac{V_0}{V_e}, \qquad K_d = \frac{V_e - V_0}{V_g}.$$

$V_e\%$ is the relative volume of elution defined by the ratio of V_e, the total volume accessible to the buffer between entering and exiting the column, and V_0, the total volume of the column minus the volume of the gel V_g. R is called the retention constant. K_d is the distribution coefficient of a substance between eluate and gel and is approximately logarithmically correlated to molecular weight in the range between 10^3 and 10^5 dalton[78].

Ion exchange chromatography separates proteins according to their net charge. The sample is given on top of a column filled with charged, immobilised beads. One can use positively charged beads (e.g. protonated diethylaminoethyl-cellulose) or negatively charged beads (e.g. ionised carboxymethyl-cellulose). The protein of interest and other constituents of the sample are bound electrostatically to the column. After the sample has been absorbed by the column a buffer gradient of continuously increasing ionic strength is applied which elutes the compounds in their order of electrical charge, those with the lowest charge first. The eluate is fractionated. *Hy-*

77 K. Weber, M. Osborne, **The Proteins**, 3rd edition, Academic Press, vol 1, p. 179, 1975.
78 N. Ui, **Anal. Biochem.**, vol 97, pp. 65-71, 1993.

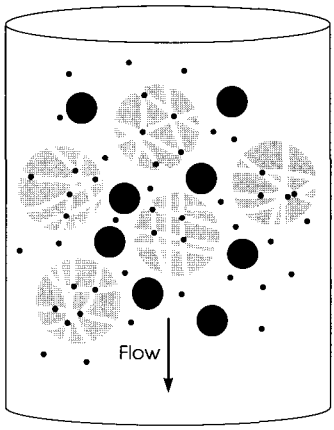


Figure 31: Gel Filtration Chromatography.
Small molecules can enter the aqueous space inside the porous beads and therefore have a longer path to migrate through the column. Large molecules do not enter the beads and pass the column first.

drophobic and *reverse phase liquid chromatography* separate components of a mixture based on their hydrophobic character.

Affinity chromatography exploits the biospecificity of a protein. A natural substrate or ligand of a protein is covalently bound to a matrix and filled into a column. The sample to be purified is added on top of the column. While passing through the column only the desired protein binds specifically and tightly to the immobilised substrate and is retained in the column. After all other constituents have left the column (because there is nothing else for them to bind to) the desired protein is eluted with a solution of substrate. Affinity chromatography is by far the most accurate chromatographic method. It is expensive because a new column has to be prepared for each different protein species and it is applicable only where a protein is known to have a specific affinity to a substrate.

Ultracentrifugation separates proteins according their shape, density and mass. The centrifugal force F_c is equal to the product of the effective mass m' and the centrifugal field $\omega^2 r$:

$$F_c = m'\omega^2 r = m(1 - \bar{v}\rho)\omega^2 r.$$

The effective mass m' is less than the physical mass m because the displaced fluid exerts an opposing force. The buoyancy factor is 1 minus the product of partial specific volume of the particle and the density of the solution. With f as the frictional coefficient of the particle the migration velocity v is:

$$v = \frac{F_c}{f} = \frac{m(1 - \bar{v}\rho)\omega^2 r}{f}.$$

To be able to characterise a protein's sedimentation behaviour independently of centrifuge and angular velocity the sedimentation coefficient s is defined as the ratio

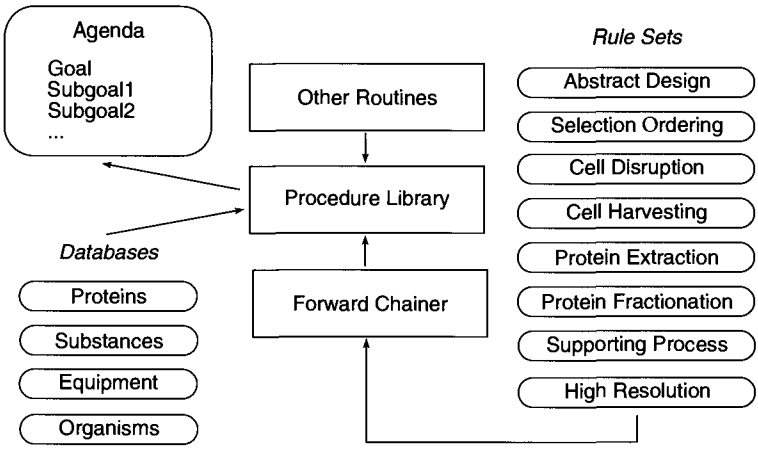


Figure 32: BioSep Designer System Architecture.
The basic components of BioSep Designer and their interactions are shown. See text for explanation.

of sedimentation velocity and centrifugal field. It is usually expressed in Svedberg units (S) which are equal to 10^{-13} seconds.

$$s = \frac{v}{\omega^2 r} = \frac{m(1-\bar{v}\rho)}{f}.$$

Ultrafiltration. Today membranes can be made with well defined, small pore size. A mixture of compounds of different sizes (near and above 100 nm) can be separated by pressuring them at 0.5 – 4 bar under chemically inert nitrogen atmosphere through a membrane. Large molecules are retained while smaller ones pass through. Ultrafiltration can also be used to concentrate the input solution efficiently at large scale. Separating proteins by ultrafiltration is of lower resolution than gel filtration. One disadvantage when handling small amounts of proteins is the adsorption of protein by the membrane.

BioSep Designer. C. A. Siletti in his Ph.D. thesis[79] developed a knowledge based program that *designs protein recovery and purification processes* for large scale manufacturing of biologically active proteins. The program does so by using a model of product and contaminants and selecting single steps which best exploit their differences. Like Molgen (section 2.2.4), it employs a top-down algorithm starting with the specification of the problem and is guided by heuristics to determine feasible alternatives for each stage of the purification process. An objective function is used to select optimal designs. BioSep designer was implemented in Lisp in an object-oriented programming style (section 2.1.2) and runs on Symbolics Lisp machines.

BioSep Designer is composed of three basic modules: 1) an agenda with the control mechanism, 2) an inference engine, and 3) several databases. The agenda

79 C. A. Siletti, *Computer Aided Design of Protein Recovery Processes*, **Ph.D. thesis**, Department of Chemical Engineering, Massachusetts Institute of Technology, MIT, 1988.

records the major procedures executed during a design. The inference engine evaluates production rules which suggest the applicability of new procedures. These are put on the agenda and executed. Once a procedure was performed it is taken from the agenda and put on a history list. Details about organisms, proteins, equipment and other substances are stored in several databases.

The control structure in BioSep Designer guides the decomposition of a goal into subgoals, provides a mechanism for selecting among alternative subgoals and records the design progress. A *goal* is a task on the agenda. *Tasks* access specific domain information. If enough information is available to implement a task the corresponding procedure is executed and the state of the design updated. If the task can not be implemented directly the procedure will place new subgoals on the agenda. The appropriate subgoals depend both on the goal and on current domain information. The control scheme is a simplified form of the blackboard control architecture. In a blackboard system a scheduler is used to determine which tasks should be executed next while with BioSep Designer this information resides with meta-level tasks or subgoals. After the basic purification steps have been determined the next task is to select equipment for each abstract process. Equipment selection tasks are interconnected because the choice of equipment at one place influences the purification steps further downstream. The immediate subgoal is therefore to check the interactions that arise from the selection of equipment and find the order of equipment selection tasks that minimises the chance for dead end designs and gives preference to the most critical process. Table 7 lists the major objects in BioSep Designer.

A node in the search for the best design is represented by an object of type *flowsheet* which has the following attributes. *Abstract-Design* holds a description of the basic processing steps without information on the equipment. A single protein recovery problem can give rise to several abstract designs based on different underlying assumptions. *Equipment* corresponds to a list of currently specified equipment. *Assumptions* are attributes of global objects such as the cellular location of the product. The preceding search node in the current context is referenced under *Parent-Flowsheet*. Nodes following the current flowsheet are stored under *Child-Flowsheet*. New flowsheets are created whenever there are alternative ways to proceed with the design process. For example, part of a typical design procedure might be to select a unit operation for the protein extraction step. The search procedure would continue by selecting one of the current alternatives, removing it from the queue and establishing all its associated assumptions. The system then tries to generate successors of the selected flowsheet by forward chaining on the appropriate rule base, in this case the protein extraction rules. For each of the alternatives found the system would create a new flowsheet object, predict its cost and add it to the list of potential designs.

A *unit operation* is characterised by a data model and a behavioural model. The data model is a set of attributes describing the physical dimensions, configuration, operating conditions and the purpose of the unit operation. The behavioural model is a set of relations among the attributes of the data model and the conditions of the feed and product streams. The following attributes are common to the data model of most equipment. *Functionality* contains the purpose of a unit operation. For example, cell harvesting, cell debris removal and precipitate clarification can

Object	Attributes	Methods
object	name, type, support-table	assert-that, assert-deletion
substance	cost, molecular weight, specific gravity	–
protein	amino acid sequence, composition, conformation, isoelectric point, hydrophobicity, stable pH range, refoldability, oxidation state	estimate-hydrophobicity, estimate-isoelectric point, estimate-molecular weight
micro-organism	genus, species, size, shape, protein composition, product composition	–
process stream	to, from, purpose, flowrate, concen-tration of solids, protein and product	–
equipment	cost, mode, temperature, pH, pressure	–
flowsheet	parent abstract design, parent flowsheet, child flowsheet, equipment list, assumptions	establish-flowsheet
abstract process	(e.g. cell-disruption is one)	–
abstract design	list of abstract processes	–
design record	global information	–

Table 7: Major Objects in BioSep Designer.
This table lists the most important object types in BioSep Designer together with their class attributes and methods (see section 2.1.2 for object-oriented programming).

all be done with a centrifuge. *Separation-Principle* is the physical or chemical unit that is exploited to effect separation. *Operating-Mode* can be batch or continuous. *Holdup-Time* specifies the residence time for continuous units and the processing time for batch units. *Number-of-Units* specifies how many units of the equipment are used. *Operating-Temperature* is defined as the maximum temperature to which the product is exposed. *Operating-pH* is the pair of lowest and highest pH encountered during the unit operation. *Capital-Cost* includes the cost of the equipment which is multiplied by an installation cost factor (e.g. 4.0) to determine the total capital investment of the unit. *Operating-Cost* includes material costs, utilities and labour associated with running the unit. Table 8 lists all unit operations.

Because unit operations sometimes have to be selected before the feed and product streams are completely specified, BioSep Designer uses models on three levels of abstraction to estimate the performance of a unit operation. The *Descriptive Model* is the least detailed model of a unit operation. It is simply a set of qualitative, symbolic descriptions of the types of acceptable feed streams and their corresponding product streams. The *Estimation Model* is a set of equations that relate the approximate performance and cost of a unit operation to information that is available at

the outset, e.g. the required annual production or initial protein and product compositions in the source organism. The *Simulation Model* is a set of equations relating the flowrates and compositions of the feed and product streams to the specific sizes, operating conditions and costs of a unit operation.

With the ability to automatically create and modify designs there must be a way to assign equipment specifications and analyse the performance of a design. This is done by simulating the unit operations under the conditions specified in the current design. Therefore, BioSep Designer must be able to solve simple differential equations which are used to describe a unit operation. Example 5 of section 2.1.1 showed an elementary differentiating program in Lisp. The reader might also want to compare the task of BioSep Designer with the simple purification / extraction example program in the same section (Example 4).

Qualitative information is used to define alternatives at different stages in the design and also to circumvent some modelling. All qualitative information is in the form of rules which are used for forward chaining. The condition specifies the applicability of a rule and its consequence contains the appropriate Lisp function call to be executed. Rules are organised into seven groups. The *Abstract-Design* rules identify basic alternative design strategies, e.g. "IF THE PURIFICATION DESIRED IS GREATER THAN 0.5 THEN INCLUDE HIGH-RESOLUTION". *Selection-Ordering* rules place priorities that are used to determine the order in which to make interdependent decisions, e.g. "IF THE CURRENT BASIC DESIGN INCLUDES CELL DISRUPTION THEN LOAD AND CHECK THE CELL DISRUPTION RULES". *Cell-Harvesting* rules identify alternative harvesting methods, e.g. "IF THE CELL SHAPE IS MYCELIAL THEN TRY A ROTARY FILTER". *Cell-Disruption* rules select alternative ways to open cells, e.g. "IF THE PRODUCT IS IN THE PERIPLASM THEN USE OSMOTIC SHOCK TO OPEN THE CELLS". *Protein-Extraction* rules determine suitable methods for isolating protein from non-protein contaminants. *High-Resolution* and *Protein-Fractionation* rules help decide between alternative high-resolution unit operations for purifying the product from protein contaminants, e.g. "IF NOTHING ELSE SEEMS TO WORK THEN TRY TO BUILD AN AFFINITY COLUMN". Finally, *Supporting-Process* rules ensure that all necessary equipment has been selected and that operating conditions are checked, e.g. "IF ION EXCHANGE IS USED AND PRODUCT IS RELATIVELY BASIC THEN A CATION EXCHANGER SHOULD BE USED AND THE BUFFER pH SHOULD BE SLIGHTLY LOWER THAN THE ISOELECTRIC POINT".

The decomposition of a goal into subgoals follows the hierarchical organisation of rule sets. The initial state of the design takes the form of a description of the bioreactor including the microbe, one or more substrates, the cell concentration, the identity of the product and any objectives and constraints for the problem. Then, (1), *Abstract-Design* rules are used to break down the design problem into a series of basic, abstract processing steps thereby effectively reducing the combinatorial complexity of the search space. (2) *Selection-Ordering* rules are used to find the best order for assigning process equipment to each of the basic processing steps from step 1. Step 2 is a meta-level task that employs two heuristics: most constrained units first and then most critical units first. (3) Now equipment can be instantiated. This involves examining the constraints imposed by the equipment already selected. (4) *Supporting-Process* rules ensure compatibility between adjacent units.

Type of Operation	Unit Operations in BioSep Designer
fermentation	fermenter
solid liquid separation concentration	centrifugation tangential flow microfiltration vacuum precoat filtration
cell disruption	high pressure homogenisation bead milling osmotic shock
protein extraction	ammonium sulphate precipitation
protein fractionation buffer exchange salt removal	cation / anion exchange chromatography size exclusion chromatography hydrophobic interaction chromatography biospecific, immunoaffinity adsorption

Table 8: Unit Operations used by BioSep Designer.
Except for fermentation all operations are used to separate compounds. Fermentation was included in BioSep Designer to get estimates for the cost and quantities of the upstream process to determine the economic feasibility of the overall process and the capacity needed for the downstream operations.

General Specifications		Bioreactor Specifications		Organism Information	
annual production	500 kg	carbon source	glucose	product location	cytoplasm & inclusion bodies
purification goal	95%	nitrogen source	yeast-e		
production hours/day	24	cell concentration	15 g/l	protein composition	0.65
production days/year	200	turn around time	12h	product composition	0.1
product use	pharm	cells	E. coli	product distribution	0.75
selection	cost	mode	batch	protein distribution	0.15

Table 9: Initial Parameters for γ-Interferon Purification.
"Composition" is the fraction of weight of the total organism. "Distribution" is the fraction of total protein (or product) in inclusion bodies.

For example, intermediate concentration or solid-liquid separation steps may become necessary. (5) Process units are connected on all levels of abstraction to complete the first flowsheet. (6) Sizes and operating conditions are optimised based on the specific flow rates known from the flowsheet. (7) Finally, the cost or any other objective function is calculated for each step of the whole process. A variation of the A*-algorithm (section 2.1.3) is used to guide steps 1 – 7 to the optimum. A difference between the standard A*-algorithm and the one used in BioSep Designer is that the solutions of the latter are further refined using increasingly detailed models. However, it cannot be guaranteed that the cost of a more refined design is lower than a previous design which would be a prerequisite for A*. The problem is solved with an error estimate that is guaranteed to overestimate the actual error at any node.

As a case study, a plan developed by BioSep Designer to purify γ-interferon is discussed. The initial parameters for the program are summarised in Table 9.

The design starts with evaluating the *abstract design* rules which have two important functions. First, they determine the presence of alternative assumptions and, second, they develop abstract designs based on those assumptions. An *abstract design* consists of a sequence of generalised operations. Details on the equipment will be specified later. In this case, the *denatured product* rule (1) is triggered which asserts that the product is to be treated as a soluble cytoplasmic protein and assumes for the current design that the product is in the inclusion body phase. Other rules that are used in this stage of the design are the following. (2) The *non-cytoplasmic product distribution* rule asks the user to supply the product distribution among different phases unless those numbers have already been specified. (3) The *distributed cytoplasmic proteins* rule uses those values to calculate the effective protein and product distributions (unless they are zero in which case there is no distribution among phases). (4) The *cell disruption* rule infers that cells have to be opened to release the intracellular product. (5) The inclusion product rule states that if some of the product is in the inclusion body phase assume that all of it will be treated as an inclusion body product and alternatively assume that only the cytoplasmic product will be recovered. (6) The *denatured product* rule infers that a renaturing step is required as the product comes from inclusion bodies where it is in denatured form. (7) The *inclusion distributed* rule asserts that if only a portion of the product is precipitated in the inclusion body phase an additional precipitation step is needed to maximise recovery. Then, the *high resolution* rule is triggered (8) because the purification goal is greater than 50%. Therefore, the *high resolution* rule set will have to be processed later on. (9) Because the product comes from cells the *cell harvesting* rule asserts that the cells must first be harvested. (10) The *alternative abstract design* rule adds a task on the agenda to establish an alternative design if there had been alternative assumptions. In the present case this was encountered in rule 5 with the decision to focus on either inclusion bodies or cytoplasmic protein. (11) The *glucose source* rule asserts that if glucose is the carbon source the contaminants are related to glucose. (12) The *always after* rule asserts a task at the top of the agenda which will always be carried out first. This task causes an initial estimation of the costs to be calculated on the basis of the abstract design created so far and the descriptive model of operating units. It also invokes the *selection ordering* rules hereafter. For the case of an inclusion body product these 12 steps lead to the following abstract design with six generalised operations: bioreactor, cell harvesting, cell disruption, protein extraction from inclusion bodies, protein renaturation and high resolution. For the case of cytoplasmic product the steps are bioreactor, cell harvesting, cell disruption, protein unfolding, protein renaturation, and high resolution.

The *selection ordering* rule set is now invoked to find feasible types of equipment for each abstract process and to determine the best order for equipment selection. Feasible process units for each abstract process are established by the rules *feasible cell disruption processes*, *high resolution candidate selection*, *protein extraction candidate selection*, *renaturation candidate selection* and *cell harvesting candidate selection*. These rules make no attempt to rate alternative choices. Constraints are placed on the order in which process units may be selected by the rules *high resolution then renaturation*, *extraction and fractionation*, *renaturation then resolution*, *purity emphasis* and *cell harvesting and cell disruption* which are interpreted as follows. (1) The *extraction and fractionation* rule says that if both protein extraction and high resolution have to

be done and the purity requirement is less than 99% then specify the extraction step first to maximise extraction recovery. High resolution steps will usually be able to achieve purification up to 99%. (2) The *high resolution then renaturation* rule states that if renaturation is required and no affinity separation column will be used it is beneficial to specify the high resolution step first to improve performance in e.g. ion exchange chromatography. (3) The *renaturation then resolution* rule says that renaturation must be carried out before the specification of an affinity column. (4) The *purity emphasis* rule states that for purification goals of greater than 99% specify high resolution methods first and then extraction. This is motivated by the *most constrained first* heuristic. For the γ-interferon purification example all these rules have to be tested twice, once for the inclusion body product case and once for the cytoplasmic product case. The selection ordering for the inclusion body product case is cell disruption, cell harvesting, unfolding and renaturation and high resolution. For the cytoplasmic product case it is cell disruption, cell harvesting, preliminary fractionation and high resolution.

At this point the search for an optimal design begins. The initial cost estimate for the inclusion body process is $6800/kg for 95% pure γ-interferon and $4600/kg for the cytoplasmic process. As the error margin is 50% both alternatives are retained and further elaborated. Now the abstract designs are extended stepwise by selecting compatible equipment from the database. The following rules are involved. (1) The *yeast or bacilli cytoplasm* rule states that osmotic shock is feasible for cell disruption if the organism is not mycelial. (2) The *add alternatives* rule is a meta-rule that creates new flowsheets for alternative choices of equipment. (3) For harvesting the cells the rule *non slimy organisms* says that if the organism is not mycelial then membrane filtration is feasible. (4) For preliminary fractionation the rule *preferred processes* checks if there are preferred operations for this product; if there aren't any, the rule *no preferred processes* looks for compatible operations. (5) For high resolution the rule *preferable units* looks for operations with an estimated purity contribution of over 1.5. These operations would be tried first. (6) The *pure enough* and *not pure enough* rules control whether further high resolution steps are needed. (7) The rule *basic protein* decides on a cation exchange column and adjusts the pH with respect to the isoelectric point of the product. (8) The *cation exchange* rule recommends the proper ion exchanging material with respect to the pH. (9) The *debris removal* rule recommends centrifugation for debris removal if no inclusion bodies are involved. (10) The *nucleic acid removal* rule is triggered for pharmaceutical applications where DNA and RNA must be completely removed from the product. This is done by precipitation and subsequent centrifugation. Out of 200 intermediate designs 32 were near optimal. The best design employed the steps listed in Table 10.

Once a design is completed the user may ask the program for explanations of why certain decisions were made. The system answers using a backtrace of the rules that led to the statement in question. The system can also be asked to simulate a design using a detailed model of purification units. This gives precise values for the instantaneous and average flowrates, the dynamic response of a process unit and the overall economic performance. Alternatively, another objective function can be selected instead of cost, e.g. reliability of purification, yield, or total purity.

Step	Purpose	Operation
1	initial cell concentration	centrifugation
2	cell disruption	high pressure homogenisation
3	lysate removal	centrifugation
4	nucleic acid removal	precipitation and centrifugation
5	protein extraction	controlled pore glass adsorption
6	final purification	cation exchange

Table 10: Final Design for γ-Interferon Purification.

Summary. BioSep Designer was a pioneer for the knowledge based design of protein purification processes. The domain chosen is narrow enough to build comprehensive knowledge bases and there is plenty of information in the form of case studies and databases on individual process units available for this purpose. The way in which BioSep Designer simulates explicit reasoning on the different stages of a design closely models a human expert's rule based approach. Explicit and symbolic reasoning in an interactive mode enables the program to give an explanation and justification for each of its design decisions. This favours the acceptance of the system with human experts. As the task of generating purification designs is combinatorial with respect to the number of facts to be considered it is certainly worth the effort developing such a system. The program can also be useful for educational purposes where students can try and evaluate different designs. Simulation of single separating operations based on the properties of equipment and material enable the program to produce a detailed flowsheet that can suggest a number of alternative designs for the purification of a product. This might lead to more economical purification designs.

With this first prototype version of BioSep Designer, however, some shortcomings must be noted. The databases used then have become outdated and would have to be reviewed. The choice of including control information in meta-rules allows great flexibility but makes is difficult to maintain the system. Adding a new meta-rule to one of the rule bases may change the interaction of previous rules depending on their order. Sometimes it is hard to predict the full consequences of adding a new meta-rule. Another problem is the representation of the strength or suitability of application and conclusion of a rule, e.g. in the form of certainty factors. This is not provided for in BioSep Designer. A comparison with Molgen (section 2.2.4) shows that in BioSep Designer an attempt is made to optimise the decisions which equipment to choose in contrast to the dynamic propagation of constraints in Molgen. The disadvantage of the approach in BioSep Designer lies in the fact that all potential interactions and relations between different pieces of equipment must be known beforehand and implemented before the program is executed while in Molgen the program itself takes care of those interactions and keeps track of them during runtime. Another criticism along the same line is that an abstract design is rigidly determined by the *abstract design* rules. Every design has to follow these rules. This means that they will have to be updated by the programmer along with any updates and extensions in the rule base of methods and equipment

to account for newly introduced shortcuts or alternatives for single abstract design steps.

Finally, the implementation in Lisp on a Symbolics machine was certainly beneficial during prototyping but seriously limits the system's use in biotechnological laboratories which rarely employ Lisp machines and knowledgeable Lisp programmers. The program would therefore have to be recoded in C or another widely used programming language which could also provide a significant speedup as Lisp code is sometimes rather slow. The evaluation of a run with 300 designs was reported to take approximately 20 minutes on a Symbolics Lisp machine.

J. A. Asenjo's Expert System for Protein Purification. J. A. Asenjo, L. Herrera and B. Byrne developed a 'second generation' expert system for the design of protein purification processes[80]. Their emphasis was on collecting a more complete and more accurate collection of purification rules from literature and industrial experts. They incorporated these rules into a so-called *expert system shell*, a computer program that contains a representation formalism for facts and IF-THEN rules. It also has a built-in inference engine to perform forward and backward chaining and a control structure to guide the processing of subgoals. A knowledge acquisition module supports the user with the input of domain specific knowledge into the expert system shell. An explanation module provides routines to analyse the chain of events that led to a particular conclusion. All components are visible to the user via the user interface. The authors' explicit intention with this program was to *amplify* human experts but *not to replace* them. J. A. Asenjo had previously[81] suggested five main heuristics as the implicit basis for the rule bases in his expert system.

Rule 1) Choose separation processes based on different properties.
Rule 2) Separate the most plentiful impurity first.
Rule 3) Use high resolution purification as soon as possible.
Rule 4) Choose processes that exploit physico-chemical differences most effectively.
Rule 5) Do the most arduous step last.

Asenjo *et al* identify two stages in protein purification, first recovery and isolation and, second, purification. The first stage includes cell separation, cell disruption, debris separation and concentration. This stage ends with a total protein concentration of about 60-70 g/l. The second stage consists of pre-treatment for further purification, high resolution purification and polishing of the final product. The sequence of the fundamental steps for the two stages is shown in Figure 33.

Recovery. The first stage goes from cell harvesting to a crude solution of protein with a total concentration of 60-70 g/l. Harvesting on a large scale is done by one of very few industrial relevant techniques: centrifugation, rotary vacuum filter or membrane filtration. One rule for this step is "IF THE MICROBIAL SOURCE IS FUNGI THEN SELECT MICROPOROUS MEMBRANE SYSTEM WITH CERTAIN-

80 J. A. Asenjo, L. Herrera, B. Byrne, *Development of an expert system for selection and synthesis of protein purification processes*, **Journal of Biotechnology**, vol 11, pp. 275-298, 1989.

81 J. A. Asenjo, *Selection of operations in separation processes*, in **Separation Processes in Biotechnology**, (A. J. Asenjo Ed.) Marcel Dekker, New York, 1989.

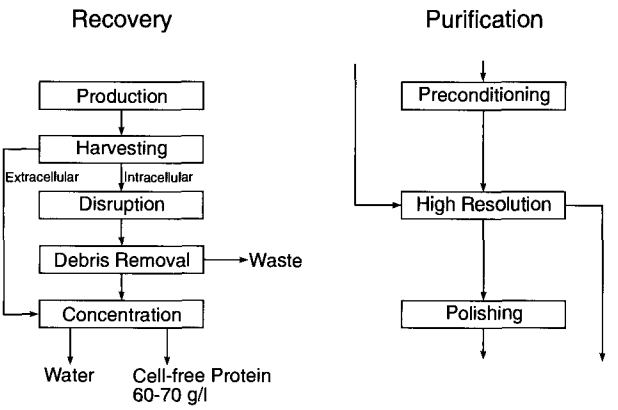


Figure 33: Two Stages of Protein Purification.
The first stage includes recovery and isolation and the second purification and polishing of the final product (see ref. 80).

TY 0.4 AND SELECT ROTARY VACUUM FILTER WITH CERTAINTY 0.3 AND SELECT FILTER PRESS WITH CERTAINTY 0.3". The choice of techniques depends on equipment efficiency and availability, microbial source and economic considerations. An extracellular product requires the liquid fraction to be collected for further processing, e.g. as with mammalian cell culture. For an intracellular product one uses the solid fraction for the subsequent cell disruption step. The different routes for extracellular and intracellular product are encoded in two rules, each specifying one route. One of the rules for disruption is "IF DISRUPTION IS REQUIRED AND MICROBIAL SOURCE IS BACTERIA THEN WITH CERTAINTY 1.0 USE HIGH PRESSURE HOMOGENISER AND ASSERT THAT THE RESULTING DEBRIS IS SMALL". Nucleic acids which are released during mechanical disruption need to be removed by precipitation with polyethyleneimine. As this is the only industrial relevant procedure for nucleic acid precipitation the expert system does not have to consider any alternatives. Separation of cell debris and soluble protein is needed for an intracellular product after cell disruption. One rule for debris separation is "IF DEBRIS IS SMALL THEN USE ULTRACENTRIFUGATION WITH CERTAINTY 0.4 AND MICROPOROUS MEMBRANE SYSTEM WITH CERTAINTY 0.6". The recovery stage is completed by concentrating the protein solution to 60-70 g/l. For most cases this is the concentration where viscosity is still low enough to prevent material loss through transportation.

Purification. The second stage starts with preparatory treatment of a concentrated protein solution from the recovery stage. Usually, the following high resolution step requires a clear solution with no precipitate and the use of special buffers. Preparatory treatment is not intended to increase purity of the product but to give a high yield. The requirement for preconditioning is expressed in two rules. Another seven rules specify various selections for the preconditioning equipment. High resolution purification is the next step. This is done by one of a number of chromatographic techniques which are known to achieve purity of 95-98%. For higher purity two or more high resolution steps are performed. If the buffer require-

ments are different in two successive high resolution steps there is a rule to suggest either gel filtration or diafiltration to change the buffer. The default rule for high resolution purification is "IF NO HIGH RESOLUTION EQUIPMENT COULD BE FOUND SO FAR THEN USE HIGH RESOLUTION ION EXCHANGE WITH CERTAINTY 0.8 AND AFFINITY CHROMATOGRAPHY WITH CERTAINTY 0.2". Finally, polishing is done to obtain ultra high purity for pharmaceutical applications. To remove oligomers of the product or hydrolysis products separation by size can be performed by gel filtration. Each of the steps described requires a particular piece of equipment to be selected. This is done in several modules that process rules to decide on the respective equipment. For a complete listing of the rules the reader is referred to the original article cited above.

This 'second generation' prototype expert system had several limitations. The rulebase did not include knowledge about protein products in inclusion bodies of *E. coli*. Also, there was no rational basis given as to how certainty factors were generated and used. A severe drawback in comparison with BioSep Designer was that chemo-physical properties of the product and contaminants were only taken into account when the rules were formulated but could not explicitly be considered by the system during run time. The authors said that the recovery stage could be well structured but not so the purification stage. This is reflected in the small number of general high purification rules that are applicable without detailed information on product and contaminants. It was also noted that the knowledge acquisition process poses a major restriction on a system of this kind because much knowledge is either unavailable for reasons of confidentiality or too vague and unspecific to be easily expressed in plain symbolic rules. This was particularly true for the high resolution purification rules. In comparison with BioSep Designer it must be noted that this approach seemed to better separate control structure (which was handled by the expert system shell) and domain information.

In a later paper[82] Asenjo's group addresses some of the difficulties encountered with the 'second generation' prototype version. Now, databases of chemo-physical properties of proteins are included. The properties to be exploited for the design of purification processes are listed in Table 11. The rulebase was extended to some 130 rules. The system is now implemented in the Nexpert Object[83] expert system shell in an object-oriented manner.

For large scale purification the following chromatographic operations were considered[84]. *Adsorption chromatography* based on van der Waals forces, hydrogen bonds, polarity or dipole moments is used for fractionation of crude feedstocks. *Gel permeation* is based on the molecular size and can be used for desalting, solvent removal, buffer exchange and end polishing. Its resolution is moderate and permeation columns have low capacity compared to other methods. However, this is the method of choice for desalting. *Affinity chromatography* is one of the methods with the best resolution. It is based on biochemical affinity and can be used for

82 E. W. Lesser, J. A. Asenjo, *The rational design of purification processes for recombinant proteins*, **Journal of Chromatography**, vol 1, no 584, pp. 35-42, 1992.
83 NExpert Object, Nexus GmbH, Dortmund, Germany, 1993.
84 J. A. Asenjo, I. Patrick, in **Protein Purification Applications** (E. L. V. Harris, S. Angal Eds.), IRL Press, Oxford, UK, pp. 1-28, 1990.

Property	Determined through
charge	titration curve
biospecificity	biological affinity
hydrophobicity	transfer energy; prediction
isoelectric point	isoelectric focusing
molecular weight	centrifuge; calculation
shape	centrifuge

Table 11: Protein Properties Utilised for Separation.

large amounts in short time although it may be quite expensive. *Reversed phase liquid chromatography* is based on hydrophobic and hydrophilic interactions between protein and matrix and is used for fractionation. This method has an excellent resolution, intermediate capacity but it may denature the protein. *Hydrophobic interaction chromatography* is based on surface hydrophobicity and can be used for partial fractionation with high resolution, capacity and speed. *Ion exchange chromatography* is based on the charge distribution of protein and contaminants and is for its high resolution, speed and capacity very well suited for an initial purification step.

The quantification of the quality of a separation operation is calculated as follows[85]. First, the *deviation DF* in one property of the product to the main contaminant protein is calculated.

$$DF = \frac{\text{Protein Value} - \text{Contaminant Value}}{\max(\text{Protein Value, Contaminant Value})}$$

$$DF = 1.0 \text{ for affinity chromatography}$$

This requires information about the average molecular mass, hydrophobicity and isoelectric point of certain fractions of the host cell. For this purpose Asenjo *et al* empirically measure those qualities for the main bands of protein in *E. coli*, yeast and Chinese hamster ovary cells. A factor η describing the efficiency of a separation operation must be estimated from experience. The *separation coefficient SC* can now be calculated from:

$$SC = DF \cdot \eta$$

$$\text{with } \eta = \begin{cases} 1.00 \text{ for affinity chromatography} \\ 0.70 \text{ for ion exchange} \\ 0.35 \text{ for hydrophobic interaction chromatography} \\ 0.20 \text{ for gel permeation} \end{cases}$$

SC characterises the ability of a separation operation to separate two or more proteins. The *separation selection coefficient SSC* can be used to select the preferred separation operation with respect to the concentration θ of the contaminant protein:

85 J. A. Asenjo, F. Maugeri, *An Expert System for Selection and Synthesis of Protein Purification Processes*, in **Frontiers in Bioprocessing II**, (P. Todd, S. K. Sikdar, M. Bier Eds.) ACS Books, Washington, pp. 359-379, 1992.

$$\theta = \text{concentration factor} = \frac{\text{concentration of contaminant protein}}{\text{total concentration of contaminant protein}}$$

$$SSC = DF \cdot \eta \cdot \theta$$

To account for economic differences in different separation methods a cost factor CF is introduced to calculate the *economic separation coefficient ESC*:

$$ESC = \frac{SSC}{CF} \quad with$$

$$F = \text{cost factor} = \begin{cases} 1.00 \text{ for affinity chromatography} \\ 0.70 \text{ for gel permeation} \\ 0.35 \text{ for ion exchange} \\ 0.20 \text{ for hydrophobic interaction chromatography} \end{cases}$$

It must be noted that the parameter η and the cost factor CF are subjective and have to be estimated from empirical measurements. The program compares the physicochemical properties hydrophobicity, molecular weight and isoelectric point of the product with those of the main contaminants. It then applies the above formulae to select the most appropriate high resolution chromatography step as defined by ESC. No examples on the performance of the program were given.

Summary. It is difficult to assess the advance of the work of Asenjo *et al* on expert systems for protein purification as no case studies were provided. However, the model they present tries to simulate the design process of human experts which is based on heuristics, rules and facts. To become successful in practice the evaluation scheme has to be substantiated with reliable values for η and CF. The system of Asenjo *et al* in its current state lacks the ability of BioSep Designer to simulate single operations or a whole sequence of separation operations. For the generation of complete flowsheets this kind of information will have to be included.

Both Asenjo's and Siletti's work demonstrate the knowledge based approach in a real world problem, the design of protein purification processes. The general benefit of the knowledge based approach is that the method of inference of the programs resembles the line of reasoning in human experts. Therefore, one can ask the program at any time for an explanation and justification of why e.g. a particular piece of equipment was chosen. As a less effective alternative a large database of products and purification processes had to be implemented which would then only be applicable to those products with known purification procedures. Such databases could not make use of chemo-physical properties except as to index the preprogrammed solutions. However, such data could be used to learn good rules for an expert system.

How crucially are heuristic approaches needed in this domain anyway? If one assumes 5 alternative ways of growing cells with the desired protein product, 10 ways of cell harvesting, 10 ways of cell disruption, 5 ways of debris removal, 5 ways of concentration, 5 ways of preconditioning, 10 ways of high resolution purification, another 5 ways of preconditioning plus 10 ways of high resolution purification, and finally 10 ways of polishing this amounts to $5^5 \cdot 10^5 = 312.5$ million different combinations. It is likely that one or more high resolution steps in addition may be required to achieve the desired purity. Also, intermediate preconditioning operations may become necessary which altogether could add another factor of, say, 100. Al-

though such number of alternative designs could currently be calculated on super computers one after the other in the course of days to weeks heuristic approaches make much of this already now available on small laboratory computers (e.g. PCs). The most critical issue remains the accurate simulation and cost evaluation of single separation operations which can be only achieved with access to industrial experience and data on the real performance of those operations.

2.2.6. Knowledge based Prediction of Gene Structure

The Nobel prize for physiology or medicine in 1993 went jointly to R. J. Robbers and P. A. Sharp, not for the work to be described in this chapter but for their independent reports at a meeting in Cold Spring Harbour in 1977 on the discovery of a single adenovirus messenger RNA molecule corresponding to four distinct encoding regions on DNA. This was first evidence of what became later known as *gene splicing*. Up to then the general consensus was that genes were continuous stretches of DNA which served as immediate templates for messenger RNA molecules and that these molecules were themselves templates for protein synthesis. Belief in this model was strong since all studies of procaryotic organisms had supported that theory. Now a more complicated mechanism had to be taken into account. This section begins with a brief introduction into RNA splicing which is followed by the presentation of two knowledge based approaches for the prediction of splice sites and gene structure.

RNA Splicing. Splicing of RNA in the cell nucleus is one of the post-transcriptional modifications of messenger RNA in eucaryotes and is carried out by splicosomes and so-called small ribonucleoprotein particles (snRNPs). The purpose of gene splicing is to remove intervening sequences (introns) from the RNA transcript that interrupt the structural coding fragments (exons). These will then be concatenated in their proper order and eventually translated into one continuous peptide string by a ribosome. Other modifications to mRNA are capping of the 5'-end with 7-methylguanosine and polyadenylation of the 3'-end. Capping is done immediately after transcription and proceeds by first hydrolysing the terminal 5'-phosphate of the first ribonucleotide. Then, the diphosphate of the mRNA end attacks the α-phosphorus atom of GTP to form a 5'-5'-triphosphate linkage. The N(7) atom of the terminal guanine and sometimes also the adjacent riboses are methylated. Caps are important for splicing and protection of RNA against degradation by phosphatases and nucleases. Ribosomal RNAs and transfer RNAs do not have caps. During polyadenylation about 250 adenylate residues are added to the end of the mRNA transcript which was cut by an endonuclease at an AAUAAA context-dependent termination signal. The purpose of polyadenylation was suggested to be a structural feature to adjust the lifetime of a mRNA molecule but this is still speculative.

Splice sites are specified through sequence patterns. The sequence of an intron starts with GU and ends with AG. Apart from this regularity there are no other apparent obligatory features, only variations of consensus sequences. At the 5'-end of vertebrate intron RNA one often finds AGGUAAGU and at the 3'-end there is a stretch of ten pyrimidines (C or U), then an arbitrary base, a C and then the

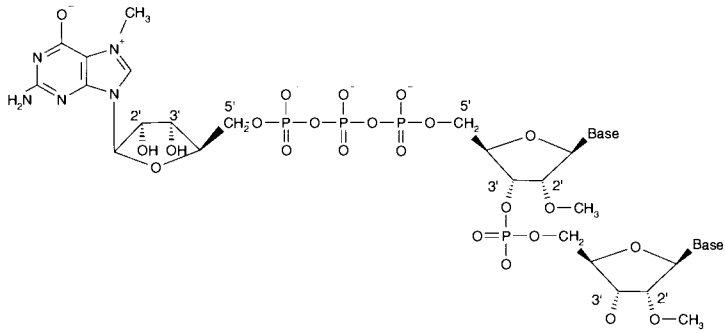


Figure 34: Chemical Structure of the 5'-Cap on mRNA.

invariant AG. The branch site in yeast is nearly always UACUAAC but can be quite different in mammals.

Gene splicing is found in all eucaryotes but only in very few instances in procaryotes. The maintenance of a splicing apparatus with the resulting overhead on the DNA level is compensated by an evolutionary advantage of an increased probability for the recombination and synthesis of new proteins with improved or new biochemical functions. Exons were often found to represent structural building blocks in proteins, e.g. motifs, super-secondary structures or domains. These are regular, compact subunits and it seems more likely that new, stable protein conformations be generated by combining a set of "qualified" building blocks than by random concatenation of arbitrary sequence fragments. So not introns *per se* are useful but the separation of structural coding fragments into (functional) subunits. In accordance with this argument it was shown that insertions or deletions within introns do not interfere with protein function except when a splice sites is changed.

Accurate prediction of splice sites is important for the determination of structural coding regions in raw DNA sequences when no mRNA transcripts are available. This is the case with the large genome sequencing projects on yeast and on the human genome. Theoretical methods should provide the means to determine the native primary structure of the coded-for proteins. In a next step, protein primary structure can then be compared with known sequences and used for molecular modelling and structure prediction. Much work has already been published on the problem of splice site recognition[86,87,88,89,90] and on the prediction of structural

86 R. Staden, *Computer methods to locate signals in nucleic acid sequences*, **Nucleic Acids Research**, vol 12, pp. 505-519, 1984.

87 R. Staden, *Measurements of the effect that coding for a protein has on a DNA sequence and their use for finding genes*, **Nucleic Acids Research**, vol 12, pp. 551-567, 1984.

88 K. Nakata, M. Kanehisa, C. DeLisi, *Prediction of splice junctions in mRNA sequences*, **Nucleic Acids Research**, vol 13, pp. 5327-5340, 1985.

89 M. Kudo, Y. Lida, M. Shimbo, *Syntactic pattern analysis of 5' splice site sequences of mRNA precursors in higher eucaryotic genes*, **CABIOS**, vol 3, pp. 319-324, 1987.

90 S. Brunak, J. Engelbrecht, S. Knudsen, *Prediction of human mRNA donor and acceptor sites from the DNA sequence*, **Journal of Molecular Biology**, vol 220, pp. 49-65, 1991.

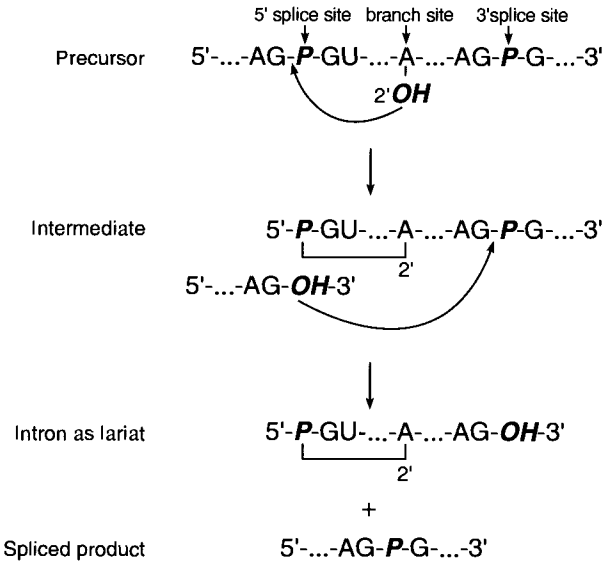


Figure 35: RNA Splicing.
During splicing the 2'-OH of adenosine in the branch site (somewhere in the middle of the intron) attacks the phosphate of the 5'-splice site to form a lariat intermediate. This frees the 3'-end of the upstream exon to attack the 5'-phosphate of the 3'-splice site and to bind to the downstream exon (direction of transcription: start at 5' → (upstream location) – (downstream location) → 3' end). The intron is released as lariat. Symbols in italics denote chemical elements, symbols in plain type face indicate ribonucleotides.

coding regions[91,92,93,94,95,96]. Another approach uses artificial neural nets and will be described in a later chapter in this book. As none of these methods achieve 100% sensitivity and some have quite low specificity there is still room and need for improvement.

Hierarchical Rule Based System for Prediction of Gene Structure. R. Guigo, S. Knudsen N. Drake and T. Smith developed a rule based system to pre-

91 J. W. Fickett, *Recognition of protein coding regions in DNA sequences*, **Nucleic Acids Research**, vol 10, pp. 5303-5318, 1982.
92 G. Fichant, C. Gautier, *Statistical method for predicting protein coding regions in nucleic acid sequences*, **CABIOS**, vol 3, pp. 287-295, 1987.
93 R. Staden, A. D. McLachlan, *Codon preference and its use in identifying protein coding regions in long DNA sequences*, **Nucleic Acids Research**, vol 10, pp. 141-156, 1982.
94 M. Gribskov, J. Devereux, R. R. Burgess, *The codon preference plot: graphic analysis of protein coding sequences and prediction of gene expression*, **Nucleic Acids Research**, vol 12, pp. 539-549, 1984.
95 A. S. Kolaskar, B. V. B. Reddy, *A method to locate protein coding sequences in DNA of procaryotic systems*, **Nucleic Acids Research**, vol 13, pp. 185-194, 1985.
96 P. W. Hinds, R. D. Blake, *Delineation of coding areas in DNA sequences through assignement of coding probabilities*, **Journal of Biomolecular Structure and Dynamics**, vol 3, pp. 543-549, 1985.

dict gene structure and structural coding regions in DNA[97]. The rules of this system are not implemented in the usual symbolic IF-THEN form as was described in the previous chapter but hierarchically guide the process of exon prediction. Their basic strategy involves (1) identifying local patterns for splice acceptor and donor sites, promoters, initiation and stop codons and poly(A) signals; (2) assigning a measure of likelihood to each of these elements; (3) generating all allowed combinations and ranking them. The structure of the highest ranking arrangement is then suggested as a model for the true gene. Two underlying assumptions are inherent to this approach. First, there is the need for an *a priori* measure for ranking DNA patterns according their regulatory function. This is difficult because not all regulatory functions and patterns on DNA are already known. Hence the ranking will sometimes not accurately reflect the actual disposition of a pattern to act as a signal. Second, an *ad hoc* defined threshold has to be introduced to maximise sensitivity.

The authors introduce the concept of *exon equivalence*. Two exons are defined to be equivalent if they can occur in exactly the same gene models. For example, if within one exon there is only one smaller exon predicted (either within the same or a different reading frame) they are both equivalent because they could alternatively be used in a particular gene model. In general, however, equivalent exons do not have to overlap. A gene model is a linear arrangement of exon equivalence classes. Searching for gene models composed of equivalent exon classes instead of individual exons helps to reduce the combinatorial explosion without missing any valid gene arrangements.

The *first step* is the analysis and prediction of local patterns for splice acceptor and donor sites, promoters, initiation and stop codons, and poly(A) signals. This is done by generating a statistical profile of nucleotide frequencies from different exons. For initiation codons the first AUG triplet is a significant signal in all eucaryotic mRNA. However, the corresponding triplet in DNA, ATG, is not a specific indicator for the beginning of translation because it might occur in a non-expressed exon or an intron. Therefore, the context of a start triplet, i.e. its neighbouring sequences will have to be considered. For initiation codons Guigo *et al* chose an interval extending 3500 bases downstream from the 5'-cap site and only exons actually known to be expressed were included in the data set for learning the profile. In about 1% of all cases this caused a true initiation site to be missed but at the same time it kept down the number of false positive predictions. The profiles express the probability of finding one of the four nucleotides at a certain position in a sequence of the training set. It is calculated as follows.

> Let $S = \{s_1, s_2, ..., s_n\}$ with $(s_k = s_{k1}, s_{k2}, ..., s_{kL})$ be a set of n aligned DNA sequences of length L, then compute the profile M_{4xL} as $M_{ij} = \frac{1}{n}\sum_{k=1}^{n} I_i(s_{kj})$ with $i = \mathrm{A, C, G, T}$ and $j = 1, ..., L$ where $I_i(x) = \begin{cases} 1 & \text{if i = x} \\ 0 & \text{otherwise.} \end{cases}$

97 Guigó R., Knudsen S., Drake N., Smith T., *Prediction of Gene Structure*, **Journal of Molecular Biology**, 226, pp. 141-157, 1992.

Position	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5
Base	G	C	C	A/G	C	C	A	T	G	G	?	G
X^2	27.9	12.6	100.9	123.7	53.6	46.1	300.0	300.0	300.0	25.1	0.1	32.5

Table 12: Profile of Translation Initiation Site.
This is the ideal sequence for initiation codons. The X^2 values show the deviation from an equal distribution of all four nucleotides. Most positions are significant with an error probability of 1% ($X^2 0.01 = 11.35$). Position 4 does not have any significant preferences and is therefore unlikely to be involved in the active site.

It is necessary to be very specific about the composition of training data sets used for deriving patterns or profiles. Ideally, the training set should be independent of the test set which is used only for evaluation of the rules. How were exon sequences collected in this case? Only exons without alternative splicing arrangements were considered. Instances of DNA reorganisation as in histocompatibility antigens and immunoglobulins were discarded as were pseudogenes and mutants. Other standard integrity constraints were applied. This data set includes unique sequences and similar sequences. The latter raise the problem that if they are not treated specially the profile would be trained to respond to particular features of those sequences. This is not desirable because such patterns would then only reflect the prevalence of certain sequence families in the training set but not necessarily *general* functional patterns. To account for this bias Guigo *et al* calculated weights for each sequence in the training set. First, all eligible exons were aligned pairwise and the maximal segment pair score noted. These scores were then used to cluster the exons by the maximal linkage method. Finally, exons in each cluster were given weights depending on their location within the cluster. The sum of the weights of all exons within one cluster is set to 1.0, the same value which is given to any unclustered sequence. Thus, the weighted profile can be calculated as:

$$M_{ij} = \frac{1}{W} \sum_{k=1}^{n} w_k I_i(s_{kj}) \quad \textit{where} \quad W = \sum_{k=1}^{n} w_k \quad \text{is the sum of the weights.}$$

Table 12 shows the data for the initiation codon profile. Similar weighted profiles are constructed for the acceptor and donor sites. For *acceptor sites* the pattern (position -14)YYYYYYYYYYYYX[C or G]AGG(position 2) was found to be optimal (Y is C or T, X is any nucleotide and C at position -1 can sometimes also be a G). The ideal *donor site* pattern was found to be (position -3)[A or C]AGGT[A or G]AGT(position 5).

In a *second step*, the frequency profiles are used to predict individual exons of three types: start exons, internal exons and terminal exons. Start exons have one initiation site upstream of its donor site and no stop codon in the same reading frame between them. An internal exon has one acceptor site upstream of a donor site both of which must lie in the same reading frame and at least a certain minimal distance apart with no stop codons between them. A terminal exon has one acceptor site with one stop codon following in frame. To minimise false positive predictions on-

ly the three best donor sites per open reading frame[98] are selected. To rank and further reduce the number of exons other numerical measures are calculated: fraction of nucleotides, codon position correlation between different sites in successive codons, and the slope of each of these variable at the beginning and end of an exon. The extreme values for these attributes are calculated from a set of known exons. A potential exon is discarded if its values lie outside those boundaries.

The *third step*, the construction of gene models is based on an approximate definition of equivalent exons because their exact determination would again require the whole exponential search space to be explored. Instead, a computationally inexpensive approximation for the concept of exon equivalency is used: two internal exons are said to be equivalent if they overlap with the same predicted start, internal and terminal exons, if they can be read in the same reading frame and if they have the same codon remainder. Then, a depth-first search through the space of equivalent exon classes is performed, beginning with one of the start exon classes which is then extended by a compatible terminal exon class to give the first candidate of a complete gene. Every compatible internal exon class is then used to expand the gene until all combinations are examined. Then a different terminal exon class is chosen and the examination of internal exon classes repeated until all combinations have been tested. Finally, the whole procedure is repeated for any remaining start exon classes.

A test was carried out to measure prediction accuracy. Unfortunately, the test set was biased because some of the sequences in the test set had also been included in the training set. Under these conditions, in 79% of 169 vertebrate genes the true exon arrangement was correctly predicted although in only 28% of the cases it was ranked number one. 80% of nucleotides in true exons were found in the prediction whereas 88% of nucleotides predicted to be part of a gene actually occurred in an expressed gene. There was a slight underprediction in the number of exons. Later, another test with 28 genes not previously seen by the algorithm but with varying homology to the training sequences was carried out. Only 69% of the true exon nucleotides were covered and 84% of the nucleotides in the test set predicted to occur in exons were actually found there. In 54% of the cases the true gene was among those predicted but only in 14% was it predicted as the top-ranking arrangement.

There may be several reasons for the low prediction quality. (1) As Guigo *et al* note, frameshift errors in the open reading frames used for initial exon identification can not be detected or corrected. (2) All genes with no alternative splicing were assumed to be processed by the same splicing machinery. This may not be true in which case more information about the splicing mechanism of each of the genes would be required before this approach could produce better results. This is in agreement with the weak performance on the new test set of 28 genes where genes with a different splice mechanism could have been included. (3) Alternatively, this could be simply a sign that the data base of splice sites is at the moment far from complete and novel sequence patterns are likely to emerge. Another way to proceed is to narrow down the taxonomic coverage of the genes in the training

98 An open reading frame (ORF) contains a fragment of DNA between a potential translation initiation site and the next translation stop codon in frame. The start codon defines the reading frame. Up to three open reading frames can overlap.

phase with the expectation that an evolutionarily more closely related and therefore more homogeneous set of sequences is likely to produce more specific patterns.

Co-operative Knowledge Acquisition System for Splice Site Recognition. E. Mephu Nguifo and J. Sallantin developed a conceptual modelling environment to analyse and predict splice sites[99]. They use an extension of Russell's definition of different levels of abstraction[100] which makes a distinction between the conceptual levels of *example*, *fact*, *regularity*, *hypothesis*, *concept formation*, *proof* and *objection*. On the first level of abstraction there are 767 positive examples for donor sites of length 60 nucleotides and 90 counter examples. The splice site data set was compiled by M. O. Noordewier, G. G. Towell and J. W. Shavlik[101]. The sequences were divided by half into training and test sets. *Facts* in this application describe which of the nucleotides A, G, C, T (or a combination of them if they could not uniquely be determined: S for C or G; R for A or G; D for A, G or T; N for any nucleotide) occur at which position. A *regularity* occurs at least once in the data set and is composed of any of the mentioned nucleotide classes (*facts*). A *hypothesis* defines a pattern of nucleotides and a corresponding set of objects that exhibit that pattern. Hypotheses are selected if they are valid, i.e. the number of examples is sufficiently large, and if they are coherent which means they cover only few or none counter examples. Concept formulation involves the co-operative revision of hypotheses. At the beginning, there were 2282 regularities of which 458 are valid and coherent. Hypotheses are scored by the percentage of positive and negative examples covered and by their similarity.

An *example* can be validated empirically or by analogy. *Empirical validation* looks at the number of regularities that are in agreement with the example. If this number is large enough, the example is accepted to be class member. If the number of regularities is too small the example is said to be a counter example. In between these limits an example remains ambiguous. Ambiguities of that kind have to be resolved by interaction with the user who can change the rate of acceptance based on the current performance of the system. E. Mephu Nguifo and J. Sallantin emphasise the importance of giving the expert user enough freedom to bias the system according to his expertise. *Validation by analogy* follows the argument that a sequence that verifies "enough" regularities and is similar to "enough" examples or one that verifies "few" regularities but which is similar to a "particular" example is likely to be a positive example. "Particular" in this context is defined as one of the initial positive examples that are not similar to any other initial positive examples. *General objections* are any patterns (negation of disjunctions) such that if one of them is found in a sequence that sequence can no longer be classified as a positive example because the pattern is rather general and occurs in the majority of regularities. *Contextual*

99 E. Mephu Nguifo, J. Sallantin, *Prediction of primate splice junction gene sequences with a cooperative knowledge acquisition system*, **Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology**, (L. Hunter, D. Searls, J. Shavlik Eds.), AAAI Press, Menlo Park CA, pp. 292-300, 1993.

100 B. Russell, **The Principles of Mathematics**, Allen Unwin Ed., London, 1956.

101 M. O. Noordewier, G. G. Towell, J. W. Shavlik, *Training knowledge-based neural networks to recognize genes in DNA sequences*, in **Advances in Neural Information Processing Systems**, vol 3, Morgan Kaufmann, 1991. Database can be obtained by anonymous ftp from ics.uci.edu under the directory /pub/machine-learning-databases/molecular-biology.

objections are negations of successful, general regularities (conjunctions). If any of them apply to a sequence that sequence cannot be similar to an initial example.

Training of this interactive system is called *co-operative revision* and is done by successively changing data on the different epistemological levels. For example, on the level of *empirical validation* the threshold for percent regularities required to accept or reject an example as class member can be adjusted. On the *hypothesis level* a decision has to be made about how many examples should be covered by a regularity to accept that hypothesis and how many counter examples can be tolerated. If, for example, *validation by analogy* produces a result contradictory to established theory the user might want to revise the concept formulation of "splice site". The occurrence of *general objections* that suppress valid positive examples requires a change on the level of facts. The user can either try to remove irrelevant facts or add more details. The user is formally guided and progress is structured by the above mentioned nine conceptual levels. Such computer assisted knowledge refinement for discrimination of splice sites from non splice sites led to the construction of a set of rules that have the following error rates, given in percent of false predictions on an independent test set: 9.63% for acceptor sites, 4.96% for donor sites, 10.12% for not-acceptor site and 4.44% for not-donor site. This is quite well in agreement with the prediction accuracy of other symbolic methods but definitely worse than artificial neural net approaches. It is the strength of this approach to assist the expert in a symbolic, knowledge based way in an incomplete, noisy and biased domain. Artificial neural nets cannot achieve this because their results are expressed as numerical weights without background theory on the objects processed. However, the expert assistant presented by E. Mephu Nguifo and J. Sallantin is not as accurate as other methods. This may have its grounds in a lack of biological expert knowledge when the program was executed, from limitations of the database or from underlying differences in the splicing mechanism.

2.2.7. Artificial Intelligence for Interpretation of NMR Spectra

This chapter examines nuclear magnetic resonance spectroscopy (NMR) and how knowledge based methods can be used to heuristically generate conformations of a protein that meet the distance constraints determined in NMR experiments. First, here is a brief introduction to NMR.

2.2.7.1. Nuclear Magnetic Resonance

Next to X-ray crystallography NMR has become an experimental alternative for routine structure elucidation of biological macromolecules. Improvements in supra-conducting technology brought sufficiently strong magnetic fields that can achieve a high enough resolution to separate the signals of thousands of atoms in one protein. NMR uses a rather concentrated solution (approximately 40 mg/ml) of protein (or DNA). Analysis of proteins in solution allows their native conformation and environment to be investigated whereas X-ray crystallography requires the use of solid crystals. In contrast to X-ray crystallography, the dynamics of a chemical reaction between protein and substrate can be monitored with NMR.

Nucleus	Spin
^{1}H	1/2
^{2}H	1
^{11}B	3/2
^{13}C	1/2
^{14}N	1
^{15}N	-1/2
^{17}O	-5/2
^{19}F	1/2
^{23}Na	3/2
^{29}Si	1/2
^{31}P	1/2
^{35}Cl	3/2

Table 13: Spins of Biochemically Relevant Nuclei.
The spins are given in units of $[h/2\pi]$ (h is Planck's constant).

However, NMR is still limited to proteins of medium size (up to about 300 amino acid residues) because of the small energy separation between different nuclear excitation levels. NMR cannot give accurate information on the structure of water molecules around a biopolymer because the electromagnetic signals measured in NMR represent an average over time *and* possibly different conformations. Therefore, a set of NMR measurements often yields conflicting distance constraints that arise from molecules of the same class but which are found in extremely differing conformations. Alternative conformations or the absence of a fixed conformation are common in NMR data.

As the name suggests, the principle of NMR is based on the interaction of an external magnetic field with the nuclei of certain atom types. Only atoms with an even number of protons and neutrons lack a magnetic moment and cannot be detected by NMR. Table 13 lists some of the most often used atom types.

If a magnetic field is applied to a probe of nuclei with finite spin the nuclear spins align with the magnetic field. When the field is turned off the spins re-orient themselves to assume the Boltzmann distribution which is in the case of a nucleus with spin 1/2 and two orientations given by:

$$\frac{N_{(m=-1/2)}}{N_{(m=+1/2)}} = e^{-\frac{\Delta E}{kT}}.$$

N is the number of nuclei in each state. The energy difference between the two states when the magnetic field is switched on is given by:

$$\Delta E = \gamma \cdot \frac{h}{2\pi} B,$$

with B as the external magnetic field, γ the gyromagnetic ratio (a constant for each atom type) and h the Planck constant. From the above equation it is obvious that very strong magnetic fields must be used to detect the small amounts of emitted energy. From the equation

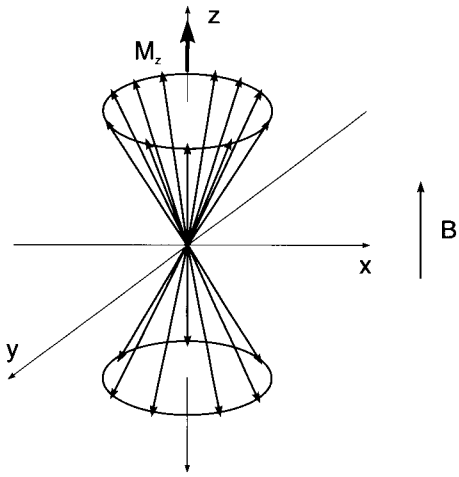


Figure 36: Magnetic Momentum Related to Relaxation Time *T1*.
The arrows from the centre indicate magnetic vectors of several nuclei that have been translated to one common origin. As more local spins point upward along with the external magnetic field *B* the net magnetic vector *Mz* of the sample points along the *z*-axis.

$$\Delta E = h\nu = \gamma \cdot \frac{h}{2\pi} B$$

the frequency of the electromagnetic radiation required to push the nuclei from their natural distribution into the higher energy state can be calculated. For example, ^{1}H protons in a magnetic field of 1.4 Tesla have a resonance frequency of 60 MHz (corresponding to a wavelength of about 5 meter) to lift the nuclear spins into an excited state of higher energy. In a NMR experiment an external magnetic field is applied. Then, a pulse of excitation frequency is sent through the sample and the intensity change over time of energy emitted during relaxation of nuclear spins is measured. A Fourier transformation is performed to obtain the NMR spectrum which shows intensity of emission over frequency.

There are five main parameters in a NMR experiment. **(1)** The *intensity I* of the emitted radiofrequency signal is related to the number of atoms and interactions of a specific type. It can be observed to change in response to a chemical reaction. For example, in a ^{31}P-NMR experiment on a frog's muscle one sees three distinct peaks of the α, β and γ phosphate atoms of ATP[102] decrease over time when the muscle is activated. Even more apparent in this case is the decrease of phosphocreatine, the precursor of ATP and the increase of inorganic phosphate which results from the degradation of ATP by hydrolysis.

(2) The *chemical shift* δ is the position of the signal on the frequency scale. Nuclei are shielded from the external magnetic field by electrons in their own atom (prima-

102 ATP is short for adenosinetriphosphate, the main "energy moelcule" in biological cells. Hydrolysis of 1 mol ATP into adenosinediphosphate (ADP) and phosphate yields -7.3 kcal.

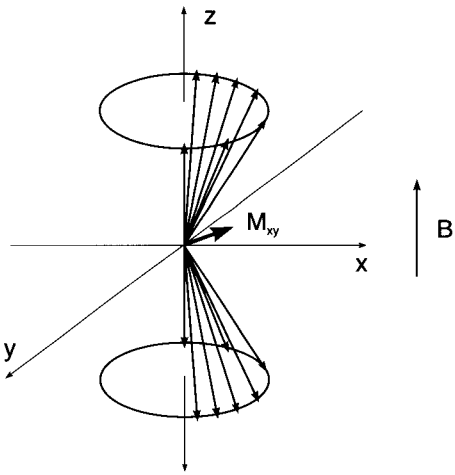


Figure 37: Magnetic Momentum Related to Relaxation Time *T2*.
The arrows from the centre indicate magnetic vectors of several nuclei that have been translated to one common origin. In this case a transversal momentum arises which induces a net magnetic vector *M*xy of the sample within the *xy*-plane. Relaxation can occur also by other means than induced by an external magnetic field. Therefore, in most cases *T2* is smaller than *T1*.

ry shift) and by those of other atoms (secondary shift). This effect is proportional to the external field B :

$$B_{eff} = B \cdot (1 - \sigma)$$

with σ as the shielding constant and B_{eff} the effective magnetic field for a shielded nucleus. Secondary shifts are important for biological macromolecules. For example, the NMR spectra of a protein in extended and folded conformation are quite different. Practically it is more convenient to measure the chemical shift δ_{obs} on the frequency scale and normalise this scale by using a signal from a reference compound δ_{ref}:

$$\delta = 10^6 \left(\frac{\delta_{ref} - \delta_{obs}}{\delta_{ref}} \right).$$

This scale has the advantage of being independent of the magnetic field B and is usually based on tetramethylsilane (TMS, $Si(CH_3)_4$) with $\delta_{TMS} \equiv 0$. Table 14 shows the range of chemical shifts for hydrogen atoms bound to different chemical groups.

(3) The *spin-spin coupling* constant J gives information on interactions of atoms which are only a few bonds apart. Figure 38 shows how the electron cloud of one atom can communicate the orientation of its nuclear magnetic spin to another atom along a chemical bond and thereby change its electronic shield against an external magnetic field. This electronic interaction along chemical bonds has the effect that the signal of neighbouring hydrogen atoms (in general all types of neighbouring atoms with an equivalent chemical environment and a nuclear spin) is split into a

Hydrogen in ...	**δ [ppm]**
Aromatic NH	15 – 9
Peptide NH	11 – 7
Aromatic H	9 – 6
CH	6 – 2
CH_2/CH (aliph. side chain)	6 – 0
CH_3	3 – 0

Table 14: Hydrogen Compounds ranked by Chemical Shift.
This table gives an approximate ranking of the chemical shift of various hydrogen compounds. Individual shifts can be quite different due to environment effects (solvent, neighbouring groups, or state of ionisation). Aromatic and peptide nitrogens with their high electronegativity draw away electrons from hydrogen so that the H atom is more exposed to the external magnetic field and experiences a stronger shift.

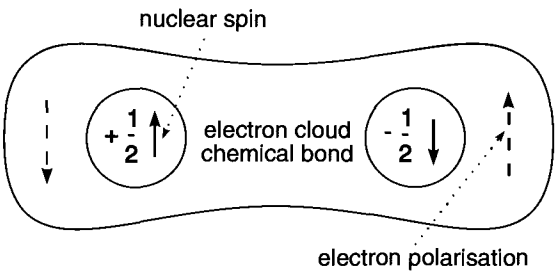


Figure 38: Spin-Spin Coupling via Electronic Polarisation.

multiplet structure. In the case of two equivalent hydrogen groups this is a triplet because each of the ^1H nuclei can be oriented in one of two ways: up (↑) or down (↓). This gives rise to four states ↑↑, ↓↑, ↑↓, ↓↓ which become macroscopically visible as three signals ↑↑, ↓↑, ↓↓ with an intensity ratio of 1:2:1.

The value of J is the distance on the frequency scale between the peaks of a split signal. If two similar groups influence each other (i.e. $\Delta\delta$ is small) and they are only one or two bonds apart (i.e. J is large) then a full multiplet structure arises in the NMR spectrum. If the two groups are rather dissimilar with respect to their electronic shielding, i.e. $\Delta\delta$ is large and they are three bonds apart their spin-spin coupling becomes weak and a simple pair of doublets is observed. If they are even further apart two single peaks arise in the spectrum and no measurable spin-coupling occurs. For the case $\Delta\delta = 0$ the two nuclei are said to be equivalent and no multiplet structure is observed. This is illustrated in Figure 39.

For illustration of multiplet structure the ^1H-NMR spectrum of ethanol (CH_3CH_2OH) is given in Figure 40. The three-dimensional structure of ethanol is shown in Figure 41. The methyl-group (CH_3-) has three equivalent hydrogen atoms which split the signal of the CH_2-group into a quartet. The three hydrogen atoms can occur in the four states ↑↑↑, ↓↓↓, ↑↑↓, ↑↓↓ each of which gives rise to

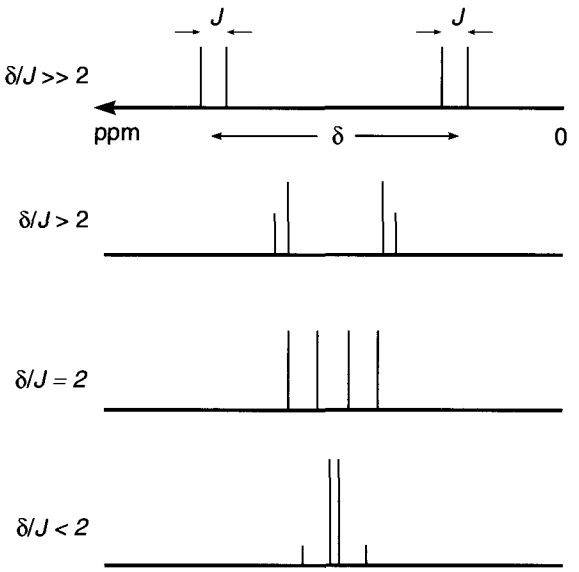


Figure 39: Chemical Shift and Spin-Coupling.
Schematic drawing of different multiplet structures. With decreasing $\delta/\mathcal{J}$ ratio the multiplet approaches the theoretical intensity distribution over all spin combinations. Vertical bars indicate the intensity at the respective position on the frequency scale.

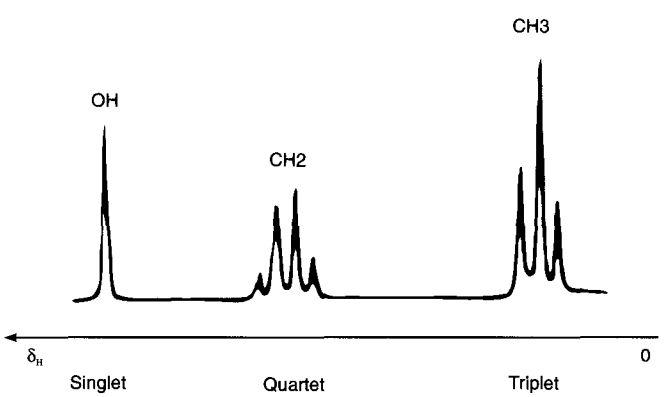


Figure 40: ^{1}H-NMR Spectrum of Ethanol.
See main text for interpretation.

a different NMR signal of the two hydrogen atoms of the CH_2-group. The same two hydrogen atoms *vice versa* split the signal of the methyl-group into three peaks as shown at the bottom in Figure 39 for the case $\delta/\mathcal{J} < 2$. The hydrogen of the hydroxyl-group (-OH) does not couple to the CH_2-group because this hydrogen atom is rapidly exchanged with hydrogen from other ethanol molecules or water. The area under the triplet is three units corresponding to the number of hydrogen

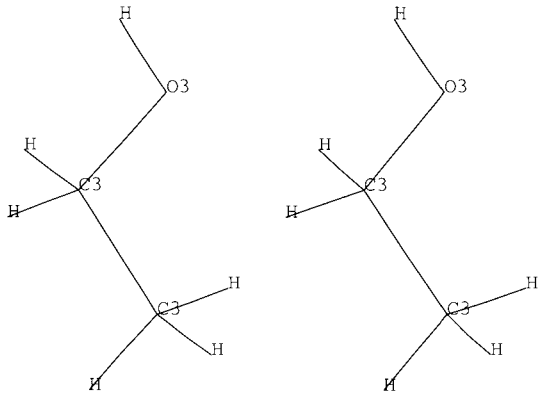


Figure 41: Stereoprojection of Wire Frame Model of Ethanol.
Look cross-eyed at the two halves of the stereo-pair to get a three-dimensional perception. Alternatively, stereo glasses can be used or a piece of paper to allow each eye to see only one half of the diagram. H is hydrogen; C3 is carbon and O3 is oxygen with sp3 orbitals.

atoms. The area under the quartet is two units and the one under the singlet is one unit, respectively.

The *relaxation times* **(4)** T_1 and **(5)** T_2 describe the rate of re-orientation of nuclear spins after application of an external magnetic field. T_2 is defined as the time constant of the decay of the M_{xy} components (Figure 37). T_1 is the time constant that describes the recovery of M_z after perturbation by the external magnetic field (Figure 36).

The techniques presented so far are referred to as "one-dimensional NMR spectroscopy". One-dimensional NMR spectra get very complicated for large molecules. Single peaks overlap extensively and can no longer be uniquely assigned to particular atoms. This led to the development of two-dimensional NMR techniques (2D-NMR). There are three basically different classes of 2D-NMR. (1) Combined spectra with different nuclei, e.g. ^{13}C and ^{1}H, with the signals of each atom species plotted on two orthogonal axes. (2) $\mathcal{J}$-resolved spectra and (3) correlated spectra. The most important techniques in the last class are correlation spectroscopy (COSY) and nuclear Overhauser effect spectroscopy (NOESY).

COSY. Correlation NMR spectroscopy allows the identification of hydrogen atoms which are separated by no more than three bonds in a molecule. The peaks on the main diagonal in the spectrum correspond to the signals from equivalent hydrogen atom groups whereas the cross-peaks off the diagonal denote interactions between atoms on the main diagonal with others of the same x or y coordinates, respectively. This allows identification of amino acid residues from their specific hydrogen interactions of NH-C_αH, C_αH-C_βH, C_βH-C_γH (etc.). These interactions give rise to individual patterns in a COSY spectrum. Figure 42 shows the case for alanine. The order of residues in a COSY spectrum is not related to the sequence of residues in a protein. Information from another type of NMR spectroscopy is needed to completely determine a structure.

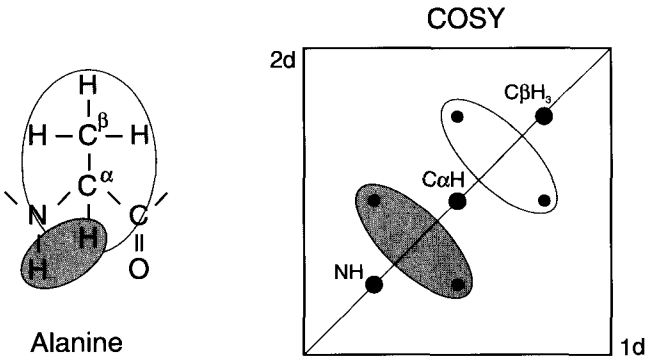


Figure 42: Schematic COSY Spectrum of Alanine.
The three thick black dots on the diagonal represent the main peaks of the three hydrogen classes ($C_\beta H_3$, $C_\alpha H$, NH) as they would appear in an ordinary 1D-NMR. Electronic interactions between shaded hydrogen atoms produce off-diagonal peaks.

NOESY. The so-called nuclear Overhauser effect describes the interaction of two nuclear spins through space. The nuclei must not be further apart than 5Å. Below 5Å distance electromagnetic interaction between two nuclei can affect each others' relaxation rates which in a 2D-NMR spectrum becomes visible as an additional off-diagonal peak. Only atoms with the maximal distance of about 5Å (or less) and which must not be covalently connected through three or fewer bonds give a signal. In a 2D-NOESY NMR experiment the hydrogens in NH, $C_\alpha H$ and $C_\beta H$ of one residue interact with the NH of the subsequent residue (Figure 43). The three corresponding distances have specific values if the residues occur in an α-helix (Figure 44) or in a β-sheet (Figure 45). This helps to identify immediately neighbouring residues and the presence of secondary structures. The information from NOESY-NMR spectra complement information from COSY spectra. Both can be combined to attempt reconstruction of the three-dimensional structure of a biopolymer. Using the method of sequential assignment it is possible first to identify single residues from COSY data and, second, to assign the location on the primary structure to individual residues with NOESY data[103].

One important limitation with NOESY-NMR is the fact that an average distance over time is measured. This means that the distance constraints may reflect only an "average" conformation of a compound which actually does not occur at all as a stable conformation. The distance constraints may also conflict with constraints from other parts of the molecule. In such cases individual examination of alternative models is required to resolve contradicting constraints.

The basic tasks of NMR spectra interpretation of biopolymers are first to identify and assign the peaks of spectra to single atoms and second, to predict a structure that satisfies all relevant distance constraints that were derived from the spectrum. Supplementary information in addition to a list of distance constraints from the NMR experiments are the sequence of components (amino acids, nucleotides), size

103 K. Wüthrich, **NMR of Proteins and Nucleic Acids**, Wiley, New York, 1986.

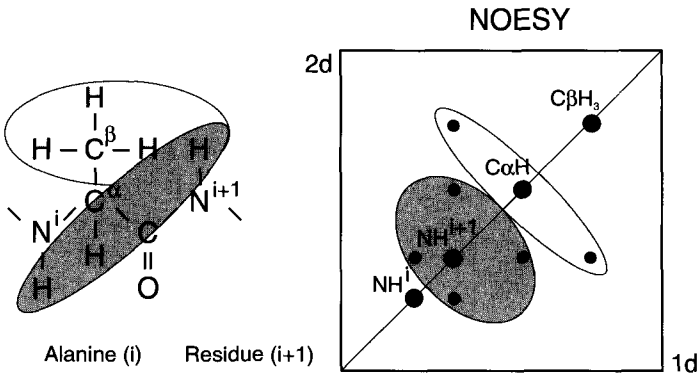


Figure 43: Schematic NOESY Spectrum of Adjacent Residues.
Interactions between hydrogen atoms that are separated by more than three bonds and less distant than 5Å give rise to a peak in the NOESY 2D-NMR spectrum. This information allows an assignment of neighbouring residues.

and shape of the molecule (measured e.g. by sedimentation) and the identity of peripheral residues (determined by the accessibility to chemical reagents). Given all this information, structure elucidation usually proceeds by (1) locating and refining local conformations (e.g. secondary structure) and (2) by combining local fragments to an overall tertiary structure. Further, the conformational space available to one particular fragment is of interest. In the past, a lot of work was done on the interpretation of NMR spectra using distance geometry algorithms[104,105,106,107] or molecular dynamics[108]. The remainder of this section looks at two approaches that are based on concepts from artificial intelligence and expert systems. The application of genetic algorithms to the problem of NMR spectra interpretation can be found elsewhere[109].

The standard way of interpreting a NMR spectrum of a protein is to consider long range distance constraints one by one and define a range of mutual orientations of two (and subsequently more) secondary structures which satisfy a given constraint. As a prerequisite, an accurate assignment of secondary structures is required which in some cases can be derived from a NOESY spectrum and circular dichroism data.

---

104 P. L. Easthope, T. F. Havel, *Computational experience with an algorithm for tetrangle inequality bound smoothing*, **Bull. Math. Biol.**, vol 51, no 1, pp. 173-94, 1989.

105 J. X. Yang, T. F. Havel, *SESAME: a least-squares approach to the evaluation of protein structures computed from NMR data*, **J. Biomol. NMR**, vol 3, no 3, pp. 355-60, 1993.

106 G. Wagner, S. G. Hyberts, T. F. Havel, *NMR structure determination in solution: a critique and comparison with X-ray crystallography*, **Annu. Rev. Biophys. Biomol. Struct.**, vol 21, pp. 167-98, 1992.

107 T. F. Havel, *An evaluation of computational strategies for use in the determination of protein structure from distance constraints obtained by nuclear magnetic resonance*, **Prog. Biophys. Mol. Biol.**, vol 56, no 1, pp. 43-78, 1991.

108 J. Hermans (Ed.), **Molecular Dynamics and Protein Structure**, Polycrystal Book Service, 1985.

109 C. B. Lucasius, M. J. J. Blommers, L. M. C. Buydens, G. Kateman, *A Genetic Algorithm for Conformational Analysis of DNA*, in: **Handbook of Genetic Algorithms**, (L. Davis Ed.), van Nostrand Reinhold, New York, pp. 251-281, 1991.

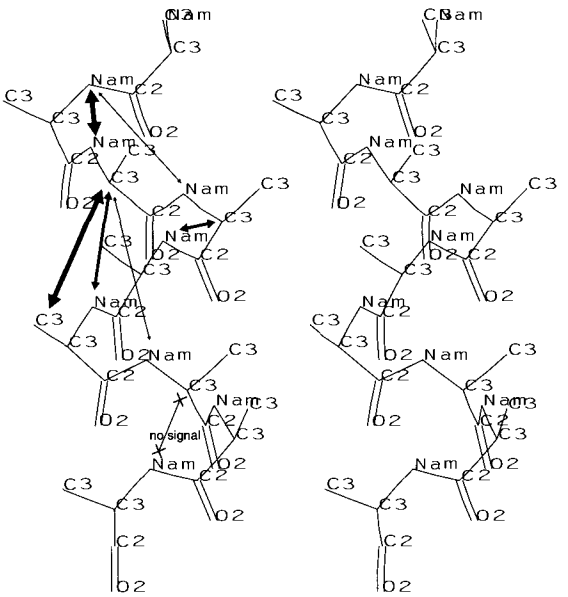


Figure 44: NMR-NOESY Interaction in α-Helices.
This is a stereoprojection of an α-helix with nine alanine residues. Hydrogen atoms are not shown. Specific nuclear Overhauser interactions occur between the hydrogen atoms bound to C_α^i–C_β^{i+3}, N^i–N^{i+1}, C_α^i–N^{i+3}, C_α^i–N^{i+1}, N^i–N^{i+2}, C_α^i–N^{i+4}. No hydrogen interaction is observed between C_α^i–N^{i+2}. Thickness of arrows approximately indicates strength of interaction. Strong, medium and weak NOE signals are taken to indicate distance limits of 2.5, 3.5 and 4.5Å respectively.

First one chooses an anchor to set up a coordinate system, e.g. an α-helix with its axis as the new z-axis. Then another secondary structure is anchored to the first one and the distance constraints are checked. If they are satisfied this assembly serves as a basis for the remaining secondary structures. To test the plausibility of the conformations generated one can compare their shape to the shape measured for the native protein and examine whether surface atoms are properly placed.

This approach relies on the minimisation of an error function that measures how well the constraints were satisfied. A principal difficulty with an error function is that it needs an *a priori* hierarchical ordering of the constraints that says which constraints are essential and which represent average distances that actually do not occur in one of the stable conformations. Due to the imprecise and incomplete nature of NMR data this question can not be answered in general and therefore individual choices have to be made on the order of domains to be anchored, the order of constraints to be satisfied and whether to accept the intersection or union of valid volumes for one secondary structure.

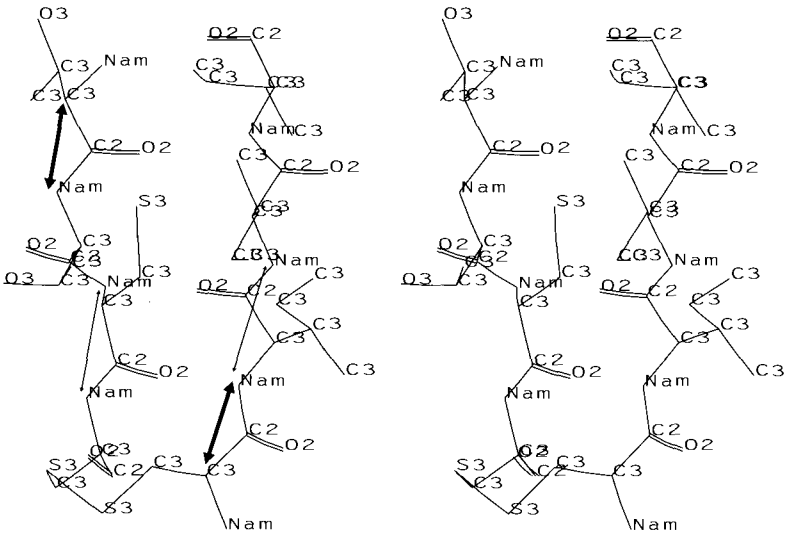


Figure 45: NMR-NOESY Interaction in β-Sheets.
This is a stereoprojection of a β-sheet with two β-strands, Thr-Thr-Cys-Cys and Cys-Ile-Ile-Ile. Hydrogen atoms are not shown. Specific nuclear Overhauser interactions occur between the hydrogen atoms bound to N^i–N^{i+1} (weak) and $C_\alpha{}^i$–N^{i+1} (strong). A three-dimensional impression can be perceived when both halves are superimposed by looking cross-eyed at the diagram.

2.2.7.2. Protean

Protean is a knowledge based system that applies several heuristics to automate the search process for valid protein conformations which satisfy a set of distance constraints derived from NMR experiments[110]. Protean is given an assignment of secondary structures with a list of distance constraints derived from NOESY data and produces a set of compatible folding patterns. Protean does not rely on the assumption that a single protein conformation gives rise to all observed NMR signals and it also does not require all distance constraints to be satisfied simultaneously. While a single structure may be compatible with only a subset of the distance constraints the ensemble of conformations accounts for all of them.

Like Molgen (section 2.2.4), Protean is based on hierarchically organised levels of abstraction to represent knowledge, in this case about protein structures. Figure 46 depicts the scope of the four main levels of abstraction.

Protean uses the so-called *blackboard architecture*[111] for problem solving. Figure 47 shows the major components. They are: (1) functional independent problem-solving knowledge sources which generate and refine parts of a solution;

110 O. Jardetzky, A. Lane, J. F. Lefevre, O. Lichtarge, B. Hayes-Roth, B. Buchanan, *Determination of macromolecular structure and dynamics by NMR*, Proceedings of the NATO Advanced Study Institute: **NMR in Life Sciences**, Plenum Publishing Corp., 1985.

111 B. Hayes-Roth, *A blackboard architecture for control*, **Artificial Intelligence Journal**, vol 26, pp. 251-321, 1985.

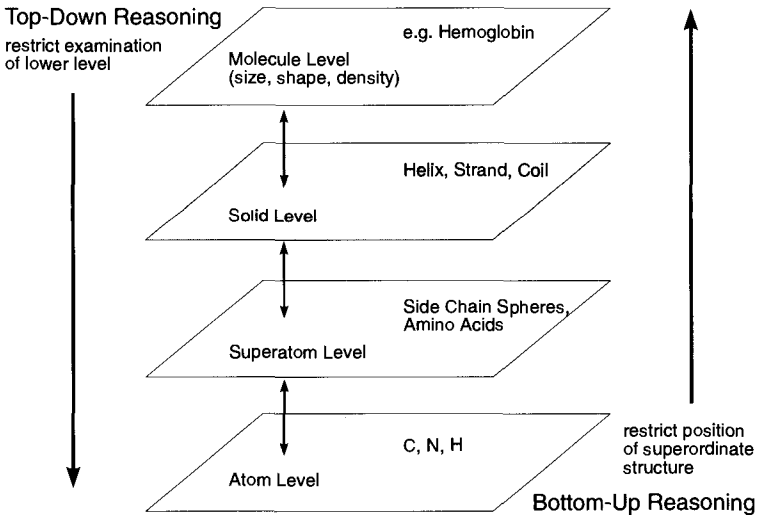


Figure 46: Protean Knowledge Representation Hierarchy.
The four levels of abstraction for representing knowledge on proteins in Protean and the two different reasoning strategies used are shown. The implementation of Protean focuses on the solid level.

(2) a multi-level solution blackboard on which different knowledge sources record their progress; (3) control knowledge sources to reason about problem solving strategy; (4) a control blackboard where actions of control knowledge sources are posted and (5) an adaptive scheduler that uses the current record on the control blackboard to determine the next action to be carried out. A blackboard system iterates through the following three steps. (1) A knowledge source activation record (KSAR) is selected from the agenda on the control blackboard and executed. Initially, this is the KSAR POST-THE-PROBLEM which retrieves the secondary structure description of a test protein and associated constraints from a data file and posts them on the blackboard in the proper format. (2) The KSAR selected in step (1) usually leads to the activation of one or more control knowledge sources which in turn place new KSARs on the agenda. For example, the initial KSAR POST-THE-PROBLEM triggers two knowledge sources: POST-SOLID-ANCHORS and DEVELOP-POSITION-OF-BEST-ANCHOR. (3) The adaptive scheduler chooses the KSAR that best satisfies the current control plan.

The representation language of Protean consists of the following concepts. An *anchor* is a secondary structure at a fixed location which defines the coordinate system relative to which all remaining secondary structures are placed. *To anchor* consequently means to position a secondary structure (called the *anchoree*) relative to an *anchor*. If a secondary structure has no explicit constraints relative to the *anchor* (i.e. it is not an *anchoree*) and it is positioned relative to an *anchoree* then this process is called *appending* an *appendage* to an *anchoree*. Completing the terminology, two *anchorees* or *appendages* are said to be *yoked* if constraints are applied between them. Figure 48 graphically describes a hypothetical situation in those terms that might arise during plan execution.

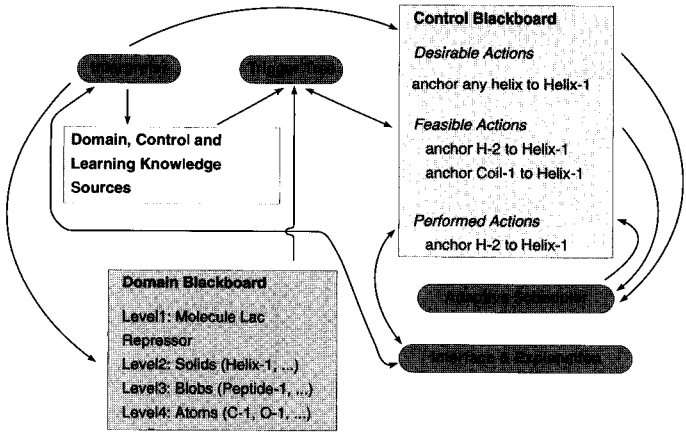


Figure 47: Blackboard Control Structure of Protean. See main text for explanation.

Protean uses 11 problem solving knowledge sources. These routines express the methods available to extend or modify a partial solution. (1) POST-THE-PROBLEM retrieves the description of a test protein and its associated constraints from a data file and posts them on the blackboard. (2) POST-SOLID-ANCHOR creates objects that describe secondary structures each one of which can serve as a potential *anchor*. (3) ACTIVATE-ANCHOR-SPACE chooses a particular *anchor* for a partial solution. (4) ADD-ANCHOREE-TO-ANCHOR-SPACE chooses a secondary structure to become an *anchoree*. (5) EXPRESS-NOE-CONSTRAINT identifies the range of positions compatible with a set of NOE constraints. (6) EXPRESS-COVALENT-CONSTRAINT identifies the range of positions available to a covalently bound secondary structure. (7) EXPRESS-TETHER-CONSTRAINT identifies the range of valid positions where two secondary structures are connected by a short coil. (8) ANCHOR-HELIX identifies all positions of a helix where it can satisfy all previously established constraints and those between itself and the *anchor*. (9) APPEND-COIL identifies the set of allowable positions of a coil where it can satisfy all previously established constraints including those between itself and the *anchor*. (10) APPEND-HELIX identifies the set of valid positions for a helix *appendage*. (11) YOKE-HELICES restricts the positions for two helices based on the constraints imposed between them.

Complementary to the 11 problem solving knowledge sources there are 16 control knowledge sources that determine what heuristics are available during program execution and which strategy to choose. Four generic control knowledge sources operate on the BB1 blackboard levels *strategy* and *focus*. (1) INITIALISE-FOCUS identifies the initial focus based on the most recently selected strategy while (2) UPDATE-FOCUS updates it. (3) If the goal of a focus has been satisfied the status of the focus is changed to "inoperative" by TERMINATE-FOCUS. (4) TERMINATE-STRATEGY concludes a strategy after its goal has been satisfied. Protean uses 12 *heuristics* as domain-specific control knowledge sources. These are: (5) develop the position of best *anchor*; (6) create the best anchor space; (7) position all secondary

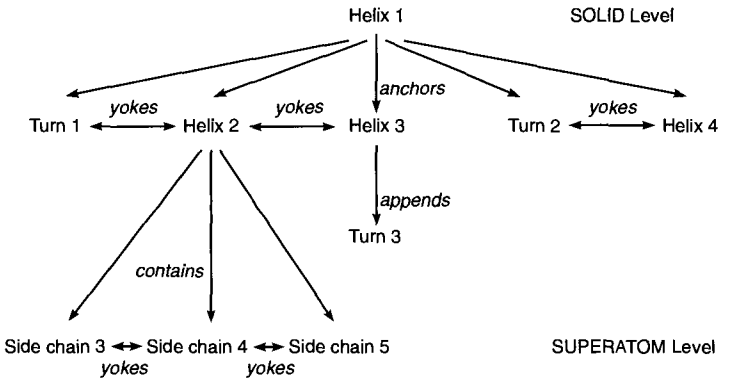


Figure 48: Protean Representation Language. See main text for explanation.

structures; (8) prefer helix over strand over coil as an *anchor* and (9) as an *anchoree*; (10) prefer long *anchors* and (11) long *anchorees* over short ones; (12) prefer *anchors* with more constraints to potential *anchorees*; (13) prefer *anchorees* strongly constrained to the *anchor* and (14) to other *anchorees*; (15) prefer strong NOE constraints over weak ones and finally (16) prefer KSARs which work on the strategically selected *anchor*. The essence of these heuristics is the general strategy to establish the longest and most constrained helix as the anchor and to add other secondary structures giving priority to long helices and secondary structures with the most constraints to other structures.

Protean was tested on myoglobin[112] where it produced a family of structures with the true known structure of myoglobin among them. Protean was also used to predict the *lac*-repressor headpiece which consists of 51 residues, three helices and four random coils. Figure 49 shows the control plan developed by Protean to complete the search.

Summary. Protean could capture the essential features in conformation prediction problems for small proteins made of only helices and coils. Its strength comes from the underlying blackboard architecture that allows easy incorporation of human expert heuristics for applying NOE derived distance constraints. The authors point out, however, that a more elaborate representation formalism will be required if global constraints on size, shape and density of the protein should be introduced. The blackboard architecture is considered flexible enough to incorporate more details on all levels of abstraction. The use of a single control strategy seems sufficient for small protein fragments but larger molecules will require the calculation of many independent partial solutions and their relations between each other. Also, alternative solutions that do not satisfy all constraints will have to be compared and examined independently. This can be difficult when at the same time one wants to avoid a computational explosion of the number of hypothesised conformations.

112 O. Lichtarge, B. Buchanan, C. Cornelius, J. Brinkley, O. Jardetzky, **Knowledge Systems Laboratory Technical Report KSL-86-12**, Stanford University, CA, 1986.

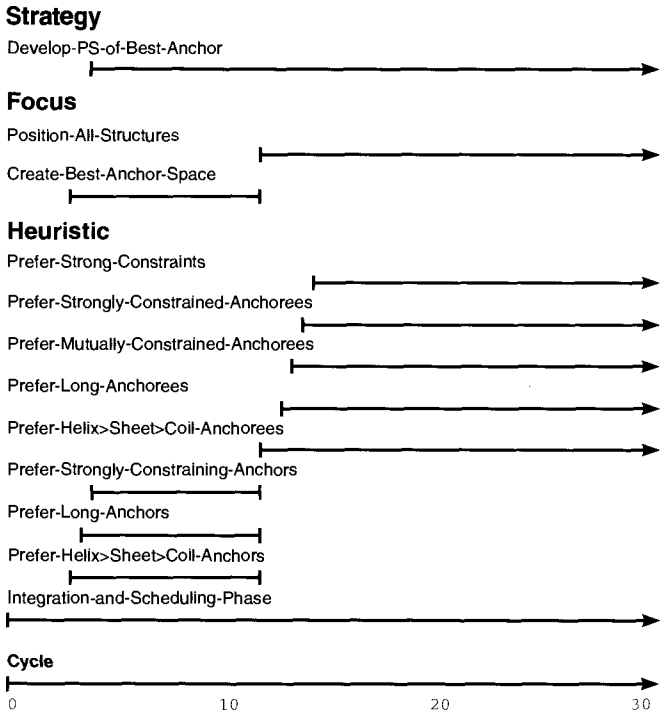


Figure 49: Control Plan developed by Protean to Solve the *lac*-Repressor Structure. This figure shows the control knowledge sources triggered in the course of the first 30 cycles.

2.2.7.3. Protein NMR Assistant

P. Edwards, D. Sleeman, G. C. K. Roberts and L. Y. Lian are developing a suit of programs to support the assignment of secondary structures in proteins from 2D-NMR spectral data[113,114]. The assembly of complete tertiary structures has not yet been attempted. Like Protean, the Protein NMR Assistant (PNA) is also based on a blackboard architecture which consists of the three basic components *blackboard*, *knowledge sources* and *control mechanism*. A blackboard system allows the solution space and domain knowledge to be partitioned in different ways and facilitates evaluation of different reasoning strategies. The PNA blackboard is divided into five levels of abstraction each of which is represented in an object oriented manner. (1) Raw spectroscopic data and the primary structure of the protein. (2) Hypotheses that explain the occurrence and appearance of residue spin-systems in a HOHAHA spectrum. (3) Continuous stretches of five to six residues derived

113 P. Edwards, D. Sleeman, G. C. K. Roberts, L. Y. Lian, *A Protein NMR Assistant*, in **Working notes Symposium Artificial Intelligence and Molecular Biology**, AAAI, Stanford University CA, pp. 41-45, 1990.

114 P. Edwards, D. Sleeman, G. C. K. Roberts, L. Y. Lian, *An AI Approach to the Interpretation of NMR Spectra of Proteins*, in **Artificial Intelligence and Molecular Biology** (L. Hunter Ed.), AAAI / MIT Press, pp. 396-432, 1993.

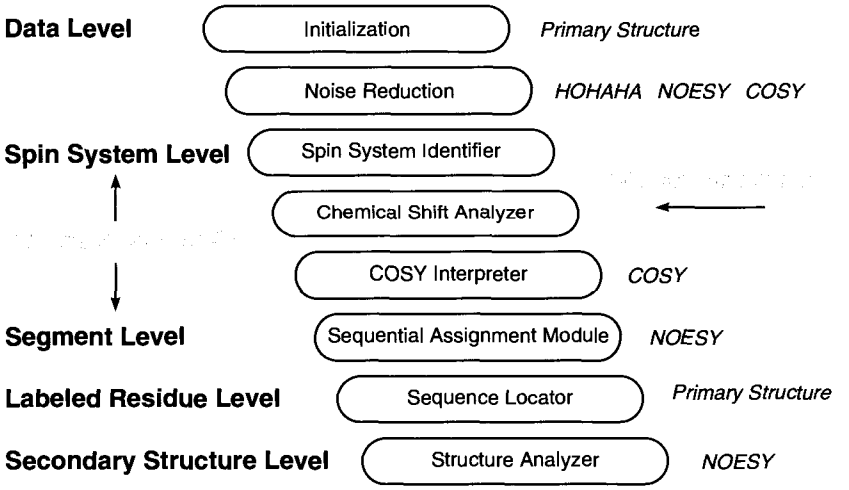


Figure 50: Levels of Abstraction and Main Modules of the Protein NMR Assistant.

from (2). (4) Complete identification of a residue and all NMR signals that originate from it. (5) Complete units of secondary structures in agreement with NOESY data. Figure 50 depicts the main knowledge sources in this hierarchy.

After initialisation, where raw COSY, NOESY and HOHAHA data are transformed into an object oriented representation with slots *id*, *xcoord*, *ycoord*, *xsize*, *ysize*, *peak-type*, and *infers*, and after the primary structure is read in, the *noise knowledge source* tries to remove those peaks from a HOHAHA spectrum that are likely to be of non-specific character. Noisy signals can be identified as groups of peaks that run parallel to the ω_1 axis. Their slot *peak-type* is then set to NOISE. The next knowledge source tries to identify *spin systems*. Therefore a data base of chemical shifts for each residue type is used. Starting in the amide proton region one finds a peak close to the diagonal and then follows a straight line parallel to the *x*-axis. Peaks on this line are then used to find corresponding peaks with the same *y*-coordinate. Such a set of peaks is then tentatively labelled as belonging to the same spin system and further compared to any admissible residue for the protein under investigation. If a match (+/- some scatter) is observed a spin system hypothesis is recorded to preliminary assign the peaks to a residue type. Next, the *chemical shift analyser knowledge source* goes over the hypotheses just created by the spin system identifier and completes the peak assignments which may have been incomplete or in error. It can identify ambiguous situations and labels them accordingly. The *COSY interpreter knowledge source* then uses COSY data and tries to verify one of the alternative hypotheses. If this can be done the hypothesis is marked complete. The next knowledge source, the *sequential assignment module*, uses the amino acid residue sequence of the protein and NOESY data to attempt identification of residues within the primary structure. First, unique residues or, if there are none, unique dipeptides within the protein are taken from the sequence. The peak of one of their $C_\alpha H$ atoms is located in the NOESY spectrum and the spin systems with an interaction with a nitrogen proton along the same *x*-axis is sought. If such an

interaction is found the corresponding residues are assumed to be adjacent in sequence. Alternatively, interactions with the N or C_β protons can be used to identify neighbouring residues. After short fragments have been assigned the *knowledge source sequence locator* tries to match the fragment to the known sequence. If there were uncertainties concerning the nature of a particular spin system, the sequence information can be used to resolve them. This leads to a fully labelled spin system hypothesis. The final step is carried out by the *knowledge source structure analyser*. Here, the expected intensities of proton interactions in secondary structures are compared to the intensities from the NOESY spectrum. If there is good agreement between the patterns and data the corresponding secondary structure is proposed. The control structure in the protein NMR assistant is kept flexible and the user is allowed to interrupt the reasoning process at different stages.

The approach shown here is remarkable because it tries to closely model the way of a human expert looking at 2D H-NMR spectra. It also allows forward and backward chaining, e.g. during comparison of spin systems and primary structure. However, the whole program is very dependent on the accuracy of NMR measurements which are not always sufficiently accurate for peak assignment. There is a recent development of 3D H-NMR spectra, which are still difficult and rather expensive to measure but which greatly simplify the process of peak assignment. It would be interesting to see how the protein NMR assistant can in the future be extended to include 3D NMR spectroscopy. Another approach using synthetic NOESY spectra for comparison with empirical measurements can be found elsewhere[115].

115 Beeson N.W., Weymouth T., Wagner G., Hyberts S.G., *A Workbench System for the Analysis of Two-dimensional NMR Spectra of Proteins*, **Working Notes of the AAAI / AIMB Workshop, Stanford**, pp. 26-27, 1990.

3. Predicate Logic, Prolog & Protein Structure

This chapter looks at the fundamental principles of predicate logic and how they can be applied to various problem solving tasks in protein structure analysis and drug design. *Predicate logic* is the basic mathematical theory and *Prolog* is a programming language that implements the key concepts of predicate logic and provides a deductive inference engine. In section 3.1 the mathematical background of predicate logic and its use in Prolog is presented. A few examples of knowledge based, symbolic programming using Prolog and the concepts of predicate logic follow in section 3.2.

3.1. Methodology

In many applications, the information to be encoded into a global database comes from qualitative and quantitative descriptive statements which are difficult in representing as simple structures, e.g. strings, sets of numbers or arrays. Complex queries to an expert system require the capability of representing, retrieving and manipulating sets of logical statements. *First order predicate calculus* is a formal language which can be used to express a variety of statements in unified syntax. Syntax rules define which expressions are legitimate in the language and how they may be combined with others to form more complex expressions. In predicate calculus, legitimate expressions are also called *well-formed formulas*.

3.1.1. Syntax

The basic entities in predicate calculus are *constants*, *variables*, *functions* and *predicates*. *Constants*, obviously, are used to refer to specific objects which can be uniquely identified by an unambiguous label, e.g. physical objects, numbers, people or abstract concepts. The first amino acid residue threonine THR-1 in a Crambin molecule is an example of a specific object. Strictly speaking, THR-1 refers to only one particular residue in one molecule. However, if one is interested to represent facts about all THR-1 residues which behave similarly and exhibit the same general properties in several Crambin molecules the THR-1 symbol can be used to refer to the *class* of all first threonine residues in Crambin proteins. *Variables* provide the means to remain indefinite about objects in the domain of discourse. For example, if one wants to represent a fact about each residue of a certain protein, a variable

can be used to refer to an unspecified residue. The variable can later be instantiated to each actual residue in turn. *Predicates* are used to express relations between objects. The convention is first to write the name of the predicate and then, enclosed in parentheses, one or more arguments to the predicate. The arguments can be *constants*, *variables* or *functions*. For example, to express the fact that threonine is the first residue of Crambin, one could invent a predicate with the name POSITION and then state POSITION (THREONINE, 1). This is also called an *atomic formula*. In general, atomic formulas are composed of a predicate and a number of (possibly nested) *terms*. A term can be a constant, a variable or a function. To express the fact that a residue exists at position 25 of unknown type, one can write POSITION (X, 25) with the variable X in place of the residue. The *arity* of a predicate is the number of arguments it takes. There can be predicates with the same name but different arity. *Functions* are used to denote the mapping or transformation of one object into another. For example, the function DIAMETER (P) can be defined to retrieve the diameter of a protein by returning the numerical measure in Ångstrom units. – As shown in the examples above, inventing and defining *constants*, *variables*, *functions* and *predicates* for a particular application is done with a specific interpretation of real-world objects in mind which determines the semantics of the predicate calculus expressions in that domain. Once an interpretation for an atomic clause has been defined, one says that this formula has the value of TRUE just when the corresponding statement about the domain of discourse is true, and FALSE if it isn't. Therefore, the well-formed formula RESIDUES (CRAMBIN, 46) has the value TRUE whereas RESIDUES (CRAMBIN, 47) has the value FALSE because Crambin is composed of exactly 46 residues.

3.1.2. Connectives

Atomic formulas can be combined to yield more complex statements. There are four types of connectives: "$\wedge$" (logical and), "$\vee$" (logical or), "$\rightarrow$" (implies) and "$\neg$" (negation). "$\wedge$" connects two atomic formulas in the way that *both of them* must evaluate to the value TRUE for the combination of both (the *conjunction*) to become true. In contrast, "$\vee$" requires that *at least one* of two atomic formulas is TRUE for the combination of both (the *disjunction*) to become true. Conjunctions and disjunctions also belong to the class of well-formed formulas. For example, the well-formed formula POSITION (THREONINE, CRAMBIN, 1) $\wedge$ HAS_HYDROXYL (THREONINE) is true because the semantic interpretation of both atomic formulas coincides with the empirically accepted knowledge about Crambin and threonine. *Implications* are used for representing "*if-then*" statements, e.g.

PROTEIN (CRAMBIN) $\rightarrow$ CONSISTS_OF (AMINO_ACIDS, CRAMBIN) $\wedge$
GREATER (NUMBER_OF_AMINO_ACIDS (CRAMBIN), 5)

says that because we know that Crambin is a protein, it follows that it consists of more than 5 amino acid residues. Otherwise it would have to be correctly identified as a small peptide. The left-hand side of an implication is called *antecedent*, the right-hand side is called the *consequent*. If both antecedent and consequent are

well-formed formulas, then the implication is also a well-formed formula. An implication has the value TRUE if either the consequent has the value TRUE (regardless of the antecedent) or if the antecedent has the value FALSE (regardless of the consequent). Otherwise, that is in the case of the consequent being FALSE and the antecedent being TRUE, the implication has the value FALSE. This definition of implication is sometimes at odds with our intuitive understanding of the meaning of "implies". For example, the following implication is logically correct and true: "If the sun is made of cheddar cheese then elephants can fly". From a false antecedent anything can follow, even a false consequence.

The negation connective "¬" is special in that it operates only on one atomic formula which is turned into its logical opposite. For example, the atomic formula

¬ PROTEIN (ASPIRIN)

is true because its interpretation of aspirin as a non-protein substance is correct. A negation is also a well-formed formula. An atomic formula and the negation of an atomic formula are both called *literals*. By tabulating all different combinations it can easily be shown that the truth values of $\neg A \vee B$ and of $A \rightarrow B$ are the same. However, in some contexts one version is much easier to comprehend. Without the use of variables, the syntax defined so far describes a subset of predicate calculus which is called *propositional calculus*.

3.1.3. Quantification and Inference

Sometimes an atomic formula with one or more uninstantiated variables always has the value TRUE, no matter what assignment is given to the variable(s), e.g. PROTEIN (X) → MADE_OF_AMINO_ACID_RESIDUES (X). This fact can be expressed by using the *universal quantifier* "∀" The formula:

∀X (PROTEIN (X) → MADE_OF_AMINO_ACID_RESIDUES (X))

can be read "all proteins are made of amino acid residues" or, alternatively, "for everything X, which is a protein, this X is also made of amino acid residues". In contrast, if one wants to express the fact that there is at least one instance known to satisfy a certain predicate, the *existential quantifier* "∃" can be used. For example, if it is known that there must exist a receptor for a certain hormone neurotensin but the identity of the receptor has not been found, this fact can be expressed as follows:

∃X (RECEPTOR (X, NEUROTENSIN)).

This formula leaves the identity of X unspecified but asserts that there is at least one type of receptor that can be activated by neurotensin. The *scope* of a quantifier extends over the entire formula that follows. A variable (here X) that is quantified is said to be a *bound variable*; otherwise it would be a *free variable*. Well-formed formulas with only bound variables are called *sentences*. *First order predicate calculus* does not allow quantification over *predicates* or *functions*, only over *variables*. Expressions that are not well-formed formulas are e.g. the negation of a function ¬F (X) or the function of a predicate FUNC (PRED (X)).

Two well-formed formulas are said to be *equivalent* ($\equiv$) if their truth values are identical for all admissible variable instantiations. Some fundamental logical equivalencies are listed below.

Identity:	$\neg(\neg A) \equiv A$
	$(A \vee B) \equiv (\neg A \rightarrow B)$
de Morgan's laws:	$\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
	$\neg(A \vee B) \equiv (\neg A \wedge \neg B)$.
Contrapositive law:	$(A \rightarrow B) \equiv (\neg B \rightarrow \neg A)$.
Distributive laws:	$(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee (A \wedge C))$
	$(A \vee (B \wedge C)) \equiv ((A \vee B) \wedge (A \vee C))$
Commutative laws:	$(A \wedge B) \equiv (B \wedge A)$
	$(A \vee B) \equiv (B \vee A)$
Quantification:	$(\forall X)\ P(X) \equiv (\forall Y)\ P(Y)$
	$(\exists X)\ P(X) \equiv (\exists Y)\ P(Y)$
	$\neg((\exists X)\ P(X)) \equiv (\forall X)\ \neg P(X)$
	$\neg((\forall X)\ P(X)) \equiv (\exists X)\ \neg P(X)$
	$(\forall X)\ (P(X) \wedge Q(X)) \equiv ((\forall X)\ P(X) \wedge (\forall Y)\ Q(Y))$
	$(\exists X)\ (P(X) \vee Q(X)) \equiv ((\exists X)\ P(X) \vee (\exists Y)\ Q(Y))$.

In predicate calculus new well-formed formulas can be produced by applying rules of inference to certain existing well-formed formulas. Two important *rules of inference* are *modus ponens* and *universal specialisation.* Given the fact A and the implication $A \rightarrow B$, the *modus ponens* rule infers B. *Universal specialisation* takes a quantified statement, for example $(\forall X)\ P(X)$, and derives from it the existence of a statement $P(A)$, where A is a constant symbol. Derived well-formed formulas are called *theorems* in predicate calculus and the sequence of inference rules which have to be applied to derive a theorem is called its *proof.* In predicate calculus one attempts to state a problem as a theorem. To solve the problem one has to find a proof for that theorem which in many cases can be automatically derived by an inference engine and theorem prover as in Prolog.

3.1.4. Unification

The process of substituting a variable in one term by a constant, another variable, a predicate or a function with the objective to make the first term identical to a second term is called *unification.* For example, to make the predicate statements P (X) and P (A) identical one has to simply substitute the variable X with the constant A. This substitution is written $s = \{A / X\}$ and the relation between the two terms is $P(A) = P(X)\ s$. The term P (A) is also called a *ground instance* because the expression does not contain any variables after substitution. Substituting X in P (X) by another variable, say Y, produces the substitution instance P (Y) which is called an *alphabetic variant.* When a variable occurs more than once in an expression, each occurrence of that variable must be substituted by the same term. Also, no variable can be replaced by a term which contains the same variable. Two (or more) substitutions can be successively applied. In general, the order of substitutions is relevant to the result, i.e. the composition of substitutions is not commutative but it is associative. A substitution *s* is called a *unifier of a set of expressions* $\{E_i\}$ if all sub-

stituted expressions are identical: $E_1 s = E_2 s = E_3 s = \ldots = E_n s$. A *most general unifier (mgu)* is a unifier g that includes all other valid substitutions s of a set $\{E_i\}$ such that $\{E_i\}\ s = \{E_i\}\ gs'$. This means that any other substitution s' can be derived from the most general unifier g by introducing additional variable / term substitution pairs. The most general unifier is unique except for alphabetic variants.

A unification algorithm to find the most general unifier of two expressions or to report failure if the two cannot be unified is the following.

1. If the two expressions E1 and E2 are identical, return the current list of substitution pairs (which is empty at first), because no substitution is necessary.
2. If E_1 is a variable, check whether E1 occurs in E2. If it does, the two expressions cannot be unified. Otherwise, if E1 does not occur in E2, add the substitution pair {E2 / E1} to the list of substitutions and return this list.
3. If E2 is a variable, check whether E2 occurs in E1. If it does, the two expressions cannot be unified. Otherwise, if E2 does not occur in E1, add the substitution pair {E1 / E2} to the list of substitutions and return this list.
4. If expression E1 is composed of other (nested) terms take the first elements of E1 and E2 and try to unify them proceeding with step 1. Then take the rest of E1 and E2 and proceed with step 1, always updating the list of the substitutions already made.

```
(defun unify (x y substitutions)
   (prog ((x (value x substitutions))
          (y (value y substitutions)))
       (cond ((variable-P x) (cons (list x y) substitutions))
             ((variable-P y) (cons (list y x) substitutions))
             ((or (atom x) (atom y)) (and (equal x y ) substitutions))
             (t (let ((new-substitutions
                        (unify (car x) (car y) substitutions)))
                    (and new-substitutions
                         (unify (cdr x) (cdr y) substitutions)))))))

(defun value (x substitutions)
    (cond ((variable-p x)
            (let (binding (assoc x substitutions)))
                 (cond ((null binding) x)
                       (t (value (cadr binding) substitutions)))))
          (t x)))

(defun variable-p (x)
    (and (listp x) (equal (car x) '?)))
```

Example 1: Unification in Lisp.
This is a simple, recursive unification procedure written in Lisp. At the beginning, the function UNIFY is given the values of two expressions X and Y and an empty list of substitutions. If one of X and Y is a variable (here implemented as a list of a question mark and the name of the variable) and has no prior binding in the SUBSTITUTIONS list, the other expression is bound to that variable. If both X and Y are constants (in Lisp: "atoms") and they are identical, then no further substitution is necessary, if they are not unification fails. Finally, sub-expressions are processed by recursive calls to UNIFY.

3.1.5. Resolution

In contrast to other programming languages and programming paradigms programming in logic with Prolog provides the programmer with an automatic deductive inference engine based on the *modus ponens*. This is implemented in the so-called *resolution algorithm*. Resolution can be applied to *clauses* (disjunctions of literals) to see whether a new clause can be derived from or is compatible with a set of given clauses. The direct approach to successively generate all compatible clauses by forward chaining and check whether the new clause appears among them is not feasible because of the large number of derivable clauses which increases exponentially with the number of clauses in a knowledge base. Even with small knowledge bases forward chaining would take too long to be practical. Hence, a different approach is required which goes as follows. Add the negation of a new clause to the existing set of clauses. Then remove all pairs of terms where one of them is the negated form of another and finally check whether this leads to a contradiction. In that case only the empty clause remains at the end. If there is any other clause left which is different from the empty clause then the original (positive) clause was not compatible with the given set of clauses. If only the empty clause remains this signifies a contradiction between the negated clause and the knowledge base. In that case resolution says that based on the closed world assumption the original (positive) clause can be derived from or is compatible with the knowledge base. For the resolution algorithm to work, first all *well-formed formulas* have to be converted into clauses. This involves the following nine steps.

1. Implication symbols have to be eliminated by substitution of $A \rightarrow B$ into $\neg A \vee B$.
2. Reduction of the scope of the negation operator ("$\neg$") by repeated use of de Morgan's laws.
3. Standardisation of variables by renaming dummy variables in quantified expressions to ensure that each quantifier has its own unique variables.
4. Elimination of existential quantifiers by replacing each occurrence of an existentially quantified variable by a *Skolem function*. The arguments of the Skolem function are universally quantified variables that are bound by other universal quantifiers whose scopes include the scope of the existential quantifier to be eliminated. A Skolem function returns a true instance of an existentially quantified variable depending on the value of other universally quantified variables whose scope includes the existential quantifier to be eliminated.
5. Move all universal quantifiers to the front of the well-formed formula and make the scope of each quantifier the entire formula.
6. Transform the matrix of expressions in conjunctive normal form which is a finite set of disjunctions of literals by application of the distributive rules.
7. Since all variables of the expression are global and must eventually be bound the universal quantifiers can be omitted. The new convention is then that all variables are universally quantified.
8. Next, all conjunction symbols ("$\wedge$") are eliminated by replacing each conjunction by a set of expressions of the terms previously connected by "$\wedge$". Now, each clause is only a disjunction of literals.

9. Finally, all variables are renamed so that no two clauses contain the same variable.

The remaining set of disjunctive expressions can now be used to find matching pairs of positive and negated terms. For expressions with variables the unification algorithm as described above is used. Resolution is said to be *refutation complete* which means that every set of inconsistent clauses produces the empty clause at the end. Resolution is also *correct* in the sense that the only case which produces an empty clause at the end is the case where the new clause was inconsistent with the knowledge base.

As an example, consider the following knowledge base of two universally quantified statements and one existential statement.

1. $(\forall X)\ A(X) \rightarrow B(X)$
2. $(\forall X)\ C(X) \rightarrow \neg B(X)$
3. $(\exists X)\ C(X) \wedge G(X)$
4. Goal statement: $(\exists X)\ G(X) \wedge \neg A(X)$

The following three clauses are equivalent to statements 1 – 3 with P as a Skolem constant. 4' is the negation of the goal statement 4.

1. $\neg A(X) \vee B(X)$
2. $\neg C(Y) \vee \neg B(Y)$

3a. $C(P)$

3b. $G(P)$

4'. $\neg G(Z) \vee A(Z)$.

Resolving 3b and 4' gives

5. $A(P)$.

Resolving 1 and 5 gives

6. $B(P)$.

Resolving 2 and 6 gives

7. $\neg C(P)$

Resolving 3a and 7 gives

8. NIL.

The empty set signifies a contradiction which means that the original goal $(\exists X)\ G(X) \wedge \neg A(X)$ could be proven over the given knowledge base and under the closed world assumption.

3.1.6. Reasoning by Analogy

This section illustrates how *reasoning by analogy* can be implemented with just a few Prolog clauses. The idea here is to give a feeling for symbolic manipulation of abstract concepts in Prolog, not to present a specialised application in molecular

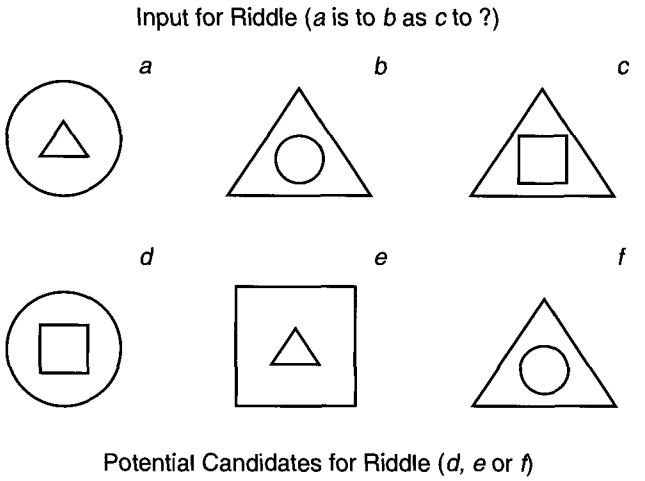


Figure 1: Geometric Analogy Problem.
Try to complete the analogy! Given *a* related to *b*, to which of *d*, *e* or *f* is *c* related by analogy? The solution appears in the trace of the corresponding Prolog program below.

biology. Applications on biology related subjects will follow in section 3.2. It is suggested that the basic algorithm presented here can be applied to more complex facts and relations, actually to anything that can be expressed in Prolog (Horn) clauses and thus can contribute to the concept of an "intelligent bio-workstation" as discussed earlier in Chapter 2. The reader is invited to try the algorithm on his / her domain of interest.

The task for this example is to answer questions of the form "A is to B as C to X?", where A, B and C are given and X to be identified from a set of potential candidates. In our case here, A, B and C are simple geometric objects as depicted in Figure 1.

The algorithm of this example was originally invented by T. Evans at MIT in the mid 1960's and later described in detail by M. Minsky[1]. The particular implementation here is derived from L. Sterlin and E. Shapiro[2]. Example 2 shows the complete Prolog program.

The first two clauses define the MEMBER predicate which returns TRUE if an element occurs at any level in a list and FALSE if not. The first clause is a fact that says "The member predicate is true for two arguments only then if the first argument (here ELEMENT) occurs at the first position in the second argument (a list)." In Prolog, the square brackets "[ ]" define a list and the "|" operator separates the first entry in a list from the remainder of the list. For example, if the second argument to MEMBER is a list of four objects, the first entry in that list is bound to ELEMENT and a list with the last three objects is bound to REST. Then the unifica-

1 M. Minsky, **Semantic Information Processing**, MIT Press, Cambridge, Massachusetts, 1968.
2 L. Sterlin, E. Shapiro, **The Art of Prolog**, MIT Press, Cambridge, Massachusetts, pp. 228, 1986.

tion algorithm tries to match ELEMENT from the first argument of MEMBER with the first element in the list. If this can be done the first clause of MEMBER succeeds, otherwise it fails. The first clause handles the case that the element to be looked up is the first in the list. If this is not the case, the second clause for MEMBER is tested. This is a recursive call of MEMBER and it means "If the given element is not the first in the list then call MEMBER again but with the first element of the list removed." That way, the list is recursively shortened from the beginning and another call to the first MEMBER clause will attempt to unify the second element of the original list with ELEMENT. This goes on until one match is found (success) or the end of the list is reached (failure).

```
member (Element, [Element | Rest]).
member (Element, [First | Rest]) :- member (Element, Rest).
analogy (A, B, C, Which, Alternatives):-
        match (A, B, Relation),
        match (C, Which, Relation),
        member (Which, Alternatives).
match (inside (Object1, Object2), inside (Object2, Object1), invert).
match (above (Object1, Object2), above (Object2, Object1), invert).
print (A, B, C) :-
        write (A), write (" is to "), write (B), write (" like "),
        nl, write (C), write (" is to ").
try (Name, Answer):-
        objects (Name, A, B, C),
        print (A, B, C),
        alternatives (Name, Candidates),
        analogy (A, B, C, Answer, Candidates).
objects (riddle, inside (triangle, circle),
                 inside (circle, triangle),
                 inside (square, triangle)).
alternatives (riddle, [inside (square, circle),
                       inside (triangle, square),
                       inside (circle, triangle)]).
run :- try (riddle, Answer), write (Answer).
```

Example 2: Reasoning by Analogy in Prolog.
This is the complete code for a Prolog program to solve the geometric analogy problem in Figure 1. See main text for a detailed explanation.

The next predicate, ANALOGY, defines an intuitive concept of analogy which goes as follows. The problem "A is to B as C to WHICH?" with a set of potential candidates for the variable WHICH can be solved by:

1) finding a relation that connects A and B (the value of that relation is bound to the variable RELATION)
2) applying RELATION to C and thereby generating an object WHICH that fulfils the analogy and
3) finally testing via MEMBER whether the object WHICH is among the available candidates.

The first call of the predicate MATCH in ANALOGY has the third argument uninstantiated and tries to establish a relation between A and B while the second call of

MATCH with the first and third arguments specified generates a new object. In that case, the second argument is unknown at the time of the call.

The following two clauses for the predicate MATCH express the fact that the predicates INSIDE and ABOVE can be related by inversion. The name of the relation is INVERT and its effect is specified as the exchange of the two arguments of INSIDE (or ABOVE, respectively).

The predicate PROBLEM does nothing other than print the current analogy problem to the screen.

The predicate TRY is the main problem solving routine. It first establishes the input data, then calls PROBLEM to print the problem to the screen, then identifies which set of potential candidates is available (ALTERNATIVES) and finally calls ANALOGY to solve the task.

The riddle itself is defined by the predicate OBJECTS. The first argument is the name of the riddle, the following three arguments specify A, B and C. The predicate ALTERNATIVES defines a list of potential candidates for a particular riddle. The first argument is the name of the riddle, the second a list of potential solutions. Finally, the predicate RUN has to be used to start the program. RUN specifies the name of the riddle to be solved and prints the solution to the screen.

Example 3 shows a trace of the program. First, program execution is initiated by a call of RUN. In this call the name of the problem is supplied (RIDDLE). Then, OBJECTS retrieves the definition of this problem and PRINT prints it onto the screen. (Additional problems could have been defined in different clauses of OBJECTS.) Both RUN and PRINT predicates succeed. A call of ALTERNATIVES retrieves the set of available candidates for this problem. Then ANALOGY is called. The first call of MATCH instantiates RELATION to INVERT, the second call of MATCH generates an object analogous to INSIDE (SQUARE, TRIANGLE) based on the analogy relation INVERT. Now the MEMBER predicate is called for the first time. As the first clause of MEMBER does not succeed, the second clause initiates a recursive call of MEMBER with the first element of the second argument removed. In the second call of MEMBER the first MEMBER clause succeeds and both calls return with TRUE. The success of MATCH and MEMBER lets ANALOGY succeed and so subsequently TRY and RUN. Below the trace is the symbolic output shown which describes the complete solution.

```
Trace:
------
>run
 >try (riddle,  Answer)
  >objects (riddle,  A,  B,  C)
  +objects (riddle, inside (triangle, circle), inside (circle, triangle),
                    inside (square, triangle))
  >print (inside (triangle, circle), inside (circle, triangle),
          inside (square, triangle))
  +print (inside (triangle, circle), inside (circle, triangle),
          inside (square, triangle))
  >alternatives (riddle,  Candidates)
  +alternatives (riddle, [inside (square, circle),
                          inside (triangle, square),
                          inside (circle, triangle)])
  >analogy (inside (triangle, circle), inside (circle, triangle),
```

```
            inside (square, triangle), Answer,
            [inside (square, circle), inside (triangle, square),
                                      inside (circle, triangle)])
   >match (inside (triangle, circle), inside (circle, triangle), Relation)
   +match (inside (triangle, circle), inside (circle, triangle), invert)
   >match (inside (square, triangle), Which, invert)
   +match (inside (square, triangle), inside (triangle, square), invert)
   >member (inside (triangle, square), [inside (square, circle),
             inside (triangle, square), inside (circle, triangle)])
    >member (inside (triangle, square), [inside (triangle, square),
                                         inside (circle, triangle)])
    +member (inside (triangle, square), [inside (triangle, square),
                                         inside (circle, triangle)])
   +member (inside (triangle, square), [inside (square, circle),
             inside (triangle, square), inside (circle, triangle)])
  +analogy (inside (triangle, circle), inside (circle, triangle),
            inside (square, triangle), inside (triangle, square),
            [inside (square, circle), inside (triangle, square),
             inside (circle, triangle)])
 +try (riddle, inside (triangle, square))
+run

Output:
-------
inside (triangle, circle) is to inside (circle, triangle) like
inside (square, triangle) is to inside (triangle, square)
```

Example 3: Trace Output for Geometric Analogy Problem.
This example shows the trace and output of the Prolog program in Example 2. Lines are indented according to nesting of the calling hierarchy. The call of a predicate is marked with ">" and its successful completion with "+". After completion all variables have been instantiated. No predicates fail in this example and therefore no backtracking is required. Predicate names and arguments are in lower case, except for uninstantiated variables that start with an upper case letter (A, B, C, CANDIDATES, WHICH, ANSWER). See text for more details.

Two parts of this program are application specific and must be adapted to accommodate a new domain.

1. Symbolic descriptions of the objects of interest have to be implemented in new clauses of OBJECTS and ALTERNATIVES.
2. Symbolic definitions of analogy relations (like INVERT) have to be encoded in new clauses of the MATCH predicate.

This example was intended to demonstrate the power and ease of symbolic programming in Prolog. With symbolic descriptions of semantic relations abstract concepts can be put to work in just a few keystrokes. Obviously, more work is required to create a practical system. However, the basic analogy reasoning strategy is completely contained in the short program above. One way to extend this version is to connect a graphical user interface and graphical representation of the objects in focus such that the user sees diagrams as in Figure 1, not only symbolic descriptions as in the output of Example 3.

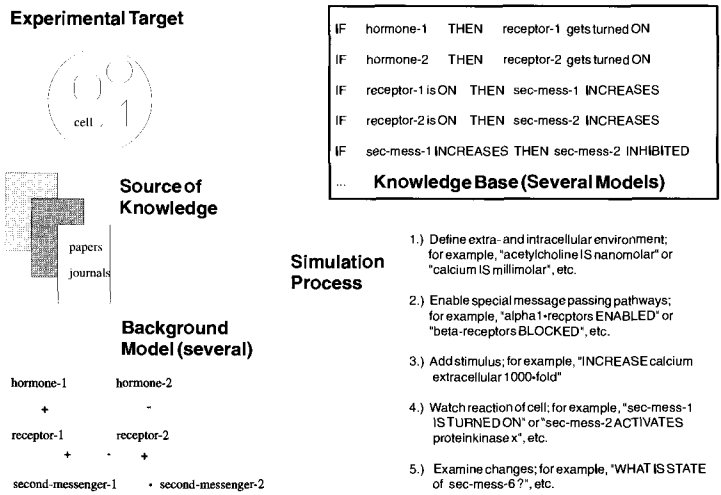


Figure 2: Qualitative-Quantitative Simulation Model.
This diagram schematically illustrates four conceptually different levels involved in developing a qualitative-quantitative simulation. The basis for model building is a collection of empirical measurements on biological objects. Those measurements were interpreted and published. The collection of publications on one subject contains much of the current knowledge on that subject or, in other words, the background model of established rules and facts. Knowledge engineering can then begin and encode facts and rules about the objects of interest in predicate logic. Finally, a knowledge based, predicate logic programming system like Prolog can be used to examine different scenarios.

3.2. Applications

This chapter examines several applications of logic programming in Prolog for molecular biological issues. In the first section an example of knowledge processing on the subject of virus transcription control is presented. After that, the use of Prolog for knowledge based encoding and prediction of protein topology will be discussed, and finally, there are sections on drug design and α-helix secondary structure prediction.

3.2.1. Example: Molecular Regulation of λ-Virus in Prolog

This section illustrates the use of declarative, logic programming with Prolog for the implementation of a qualitative and quantitative simulator for molecular regulation of expression in λ-phage[3]. In cases where clearly defined mathematical expressions are known to model the behaviour of the components of a system differential cal-

3 S. Schulze-Kremer, *CS Prolog – Parallel Programming with Transputers. Application: MolSIM, A Program for Simulation of Concurrent Molecularbiological Processes*, in **Tools and Techniques for Transputer Applications**, (Ed. S. Turner), IOS Press, Amsterdam, pp. 97-110, 1990.

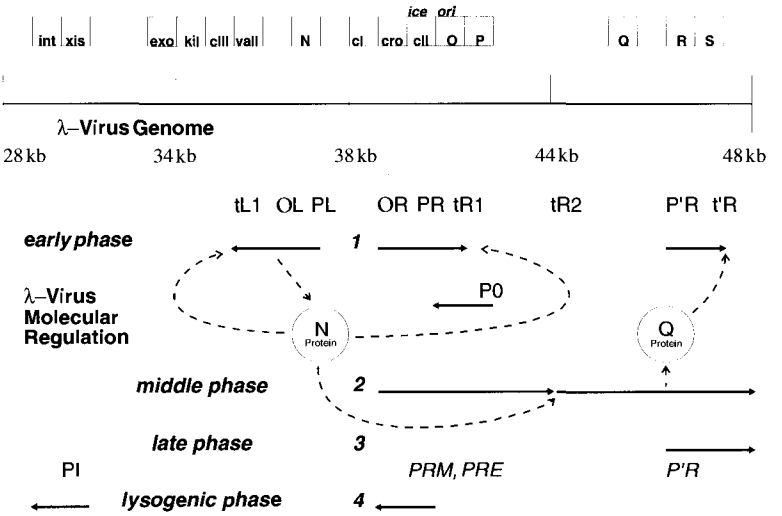


Figure 3: λ-Phage Genome and Molecular Regulation.
This diagram schematically shows the organisation of part of the λ-phage genome and the location of some important genes on it (at the top in boxes). Below there are two operators (OL, OR; R stands for "to the right on the genome", L for "to the left"), three promoters (PL, PR, P'R) and four transcription terminators (tL1, tR1, tR2, t'R) drawn. During lysogenic phase (4), only the gene cI is expressed which codes for the λ-repressor protein. That protein binds to the two operators OL and OR and thus blocks expression of other viral genes. For the lytic phase, when there is not enough λ-repressor protein to prevent transcription at OL and OR, OL begins to synthesise N-protein which acts as an anti-terminator for transcription. This leads to the expressions of genes behind the tR1 and tR2 termination sites. Another protein, called Q-protein, is produced which anti-terminates at site t'R.

culus should be applied. There are, however, areas which lack a complete mathematical description and which may also include semi-quantitative statements (like "A stimulates B") or qualitative statements (like "A initiates synthesis of B"). There, differential equations require empirically estimated coefficients which in many cases may not be obtained with satisfactory reliability. Instead, one would like to be able to see towards which state a simulation would develop if it is only guided by a model of possibly inaccurate or incomplete qualitative statements (*approximate semi-quantitative simulation*).

The λ-phage[4] is a virus particle that consists of 48 kb double-stranded DNA[5] surrounded by a protein coat. It has also a "tail" that allows the phage to dock to bacteria and to insert its genetic material into the host. After that is done, the virus can choose between two developmental pathways. It can destroy the bacterium immediately by taking over protein synthesis to produce about 100 new copies of itself

4 M. Ptashne, A. D. Johnson, C. O. Pabo, *A genetic switch in a bacterial virus*, **Scientific American**, 247, (5), pp. 128-140, 1982.
5 1 kb = 1000 nucleotide base pairs of DNA.

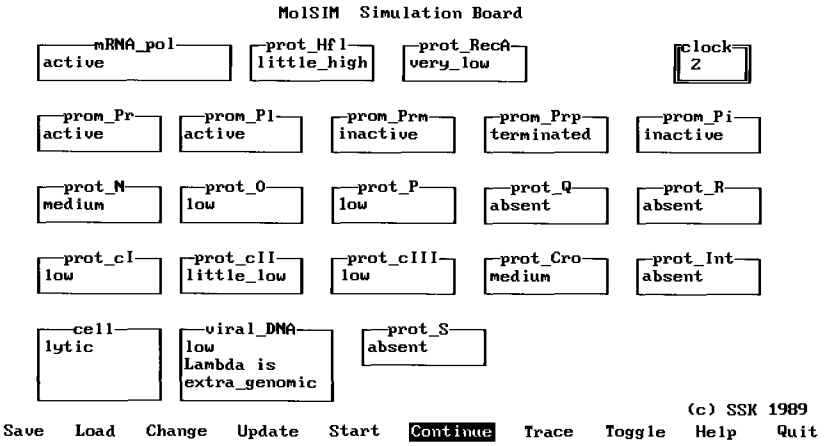


Figure 4: MolSIM Simulation Screen.
A program MolSIM was written in CS Prolog to execute and evaluate a predicate description of a model of λ-phage transcription control and replication. The screen shows symbolic gauges to quantitatively and qualitatively monitor the concentration and activity of various proteins, the state of the cell, of viral DNA and of several promoters. The values change during run time.

(*lytic pathway*) or it can remain silent and covalently insert its genetic material into the host chromosome (*lysogenic pathway*).

In the latter case the integrated DNA can be activated later in response to chemical stimuli or changes in metabolism. In the lysogenic pathway only very few functions of the viral DNA remain active which detect the concentration of some virus-related proteins. The silent DNA is called *prophage* and replicates as part of the bacterial host chromosome, usually for many generations. Figure 3 depicts the fundamental control mechanism of λ-phage and the mechanism to switch from lysogenic to lytic stage.

Example 4 is a listing of the basic source code required for MolSIM to simulate a simple model for λ-phage expression control. All statements are in standard Prolog syntax as defined by W. F. Clocksin and C. S. Mellish[6]. There are facts that start with the name of the predicate which is followed by a list of arguments enclosed in round brackets. Rules start with a predicate followed by a colon and a dash (":–" stands for the implication "follows from" or "←") and conditions following as a conjunction of predicates. The first predicate is the consequence and those following the implication sign (pointing to the left!) are conditions which must be fulfilled for the consequence to be drawn. All clauses are terminated with a period. Remember that in Prolog variables start with upper case letters and constants with lower case letters. The individual clauses are indexed and can be translated into English as follows.

```
[1] processes ([mRNA_pol, prom_Pl, prom_Pr, prom_Prm, prom_Prp, prot_cII, prot_RecA, prot_O, cell]).
[2] delta_time (1).
```

6 W. F. Clocksin, C. S. Mellish, **Programming in Prolog**, 2nd edition, Springer, 1984.

```
 [3] win (mRNA_pol, 2, 1, 3, 20, red, [[1, 1, active]]).
 [4] concentration (sigma, higher).
 [5] activity (prot_cII, low).
 [6] inhibition (prom_Pl, [disp (prom_Pl, inactive)]) :- conc (prot_cI >> low).
 [7] action (prom_Pl, [inc_d (prot_N, c, 1)]) :- conc (sigma >> medium).
 [8] action (prom_Pr, [send (high, [prot_O])]) :- not (terminator (tR1)).
 [9] action (prot_N, [anti_terminates ([tL1, tR1, tR2])]) :- conc (prot_N >> medium).
[10] action (prot_Int, [set (lambda (integrated))]) :- conc (prot_Int >> medium).
[11] action (prot_R, [set (state (cell, lyses))]) :- conc (prot_R >> high), conc (prot_S >> high).
[12] action (cell, [halt_simulation]) ;- state (cell, lyses).
[13] reaction (enz_ATPase, [[atp, 1]], [[adp, 1], [p, 1]], 20000).
[14] moi (medium).
[15] lambda (extra_genomic).
[16] state (cell, intact).
[17] terminator (tR1, prom_Pr).
[18] anti_terminates ([]).
[19] anti_terminates ([X | REST]) :- terminator (X, P),
     disp (P, anti_term), retract (terminator (X, P)), anti_terminates (REST).
[20] phase (lytic) :- (conc (prot_cII >> low) or act (prot_cII >> low)),
     conc (prot_cI << medium), disp (cell, lytic).
[21] phase (lysogenic) ;- conc (prot_cI >> medium), disp (cell, lysogenic).
```

Example 4: Source Code for λ-Phage Molecular Regulation Model.
This is the basic Prolog source code required to specify the simulation model for λ-phage expression control. See main text for detailed explanation.

1. There are nine processes in this simulation, each with its own properties and a gauge. The processes are listed in square brackets which is the Prolog syntax for sets: [MRNA_POL, PROM_PL, PROM_PR, PROM_PRM, PROM_PRP, PROT_CII, PROT_RECA, PROT_O, CELL].
2. This is a discrete simulation and the interval between two simulation steps is one time unit. The unit is arbitrary but has to be used uniformly throughout the model.
3. This is the specification of an output monitoring window (gauge or display) on the screen connected to the process MRNA_POL. The first four numbers determine location and size of the window followed by a colour code. The final argument, a set of three elements, specifies the value *active* to be initially printed in line 1, column 1. Window specifications for other displays have been omitted for brevity.
4. The concentration of the sigma factor is *higher*, a symbolic, semi-quantitative value between *high* and *very high*.
5. The activity of protein cII is *low*.
6. Promoter PL is inhibited if the concentration of protein from gene cI becomes larger than *low*. Then the message *inactive* appears on its display.
7. Promoter PL will increase the concentration of protein N by one unit per cycle if the concentration of sigma factor exceeds the value *medium*.
8. Promoter PR will cause the concentration of protein O to increase to level *high* if transcription is not terminated at terminator site tR1.
9. Protein N anti-terminates the terminators tL1, tR1 and tR2 if its concentration exceeds the value *medium*.
10. Insertion protein *Int* allows λ-phage DNA to integrate into the host genome if its concentration exceeds the value *medium*.
11. Protein R initiates lysis of the cell if both concentrations of protein R and protein S are greater than *high*.
12. The simulation should be halted when the cell lyses.

13. The enzyme ATPase converts one molecule ATP into one molecule ADP and P with a turnover of 20000 conversions / minute.
14. Multiplicity of infection is *medium* at the beginning of the simulation.
15. λ-Phage DNA is not inserted in the host genome at the beginning of the simulation.
16. The cell is intact at the beginning of the simulation.
17. Promoter PR is terminated at the site tR1 at the beginning of the simulation.
18. Nothing is anti-terminated at the beginning of the simulation.
19. Antitermination of a list of termination sites is defined recursively. The termination site is identified, taken from the list of active termination sites and a corresponding message is posted on the display of the termination site.
20. This is the definition of lytic stage. If either the concentration or activity of the protein encoded by gene cII is greater than *low*, and the concentration of protein product cI is lower than *medium* then post the message *lytic* in the display of *cell*.
21. The lysogenic state is defined by the presence of protein product cI in a concentration greater than *medium*.

As can be seen, quantitative and qualitative statements can be expressed and connected with each other. Now the simulator can be initiated. The displays and gauges show the current status and concentrations of the system as it changes over time. A transcript of all interactions is written to a file for off-line analysis of the sequence of events. MolSIM was implemented in CS Prolog, a Prolog dialect that can perform parallel evaluation of subgoals on a personal computer hosting several transputers. Depending on the number of transputers available some or all regulatory processes can be executed in parallel. One use of MolSIM is to evaluate a user-defined model. For example, if the simulation shows rapidly decreasing concentration of a vital protein then this could point to a potential malfunction in the bacteria in which case the model offers an explanation for the disturbed metabolism. Alternatively, if one expects perfect growth of the cells under the defined conditions, the result of the simulation is a hint that the model and in particular the statements on the respective protein must be checked for correctness and completeness.

3.2.2. Knowledge based Encoding of Protein Topology in Prolog

The first publication on the idea of using Prolog to describe and analyse protein structures appeared in the mid-80s[7]. Figure 5 shows the topology of the Flavodoxin domain and Example 5 has the corresponding Prolog code. This example shows how geometrical features and topological relations between secondary structures can be encoded symbolically.

7 C. J. Rawlings, W. R. Taylor, J. Nyakairo, J. Fox, M. J. E. Sternberg, *Reasoning about Protein Topology using the Logic Programming Language PROLOG*, **Journal of Molecular Graphics**, vol 3, no 4, pp. 151-157, 1985.

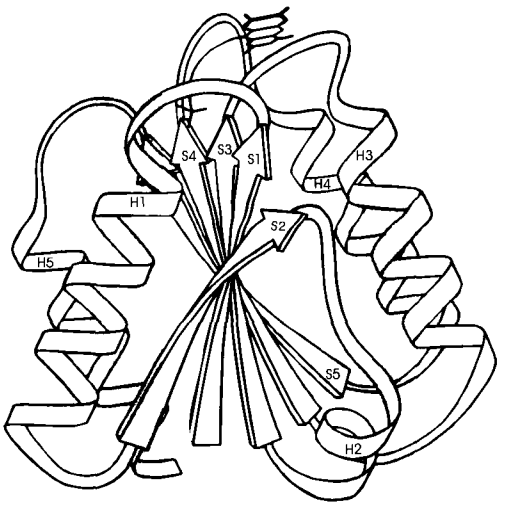


Figure 5: Ribbon Diagram of Flavodoxin.
Flavodoxin has one domain with five α-helices (H1 – H5) and five parallel β-strands (S1 – S5). Arrows indicate the orientation of β-strands. The topology of this domain is described in Prolog clauses in Example 5.

```
domain ([layer (1), layer (2), layer (3)]).
                                    # three layers in this domain
helical_layer ([layer (1), layer (3)]).
                                    # layer 1 and 3 have only helices
sheet_layer ([layer (2)]).
                                    # layer 2 has only β-strands
between (layer (1), layer (2), layer (3)).
                                    # layer 2 is in between layers 1 and 3
contains (layer (1), [helix (2), helix (3), helix (4)]).
contains (layer (2), [strand (1), strand (2), strand (3), strand (4),
                      strand (5)]).
contains (layer (3), [helix (1), helix (5)]).
                                    # which secondary structures in which layer
parallel ([strand (1), strand (2), strand (3), strand (4), strand (5)]).
                                    # strands 1 - 5 are parallel
right_twisted ([strand (1), strand (2), strand (3), strand (4), strand (5)]).
                                    # strands 1 - 5 are twisted right-handed
adjacent (helix (1), helix (5))
adjacent (helix (3), helix (4))
adjacent (strand (1), strand (2))
adjacent (strand (2), strand (3))
adjacent (strand (3), strand (4))
adjacent (strand (4), strand (5))
                                    # adjacent secondary structures
βαβ_units_between (direct_neighboor_strands)
βαβ_units_between (distant_neighboor_strands)
                                    # where the βαβ-units are located
positioned (helix (3), helix (4), parallel)
positioned (helix (1), helix (5), strongly_twisted)
positioned (strand (2), strand (5), orthogonal)
positioned (strand (1), strand (3), slightly_twisted)
positioned (strand (3), strand (4), slightly_twisted)
positioned (strand (4), strand (5), slightly_twisted)
                                    # relative positions of secondary structures
```

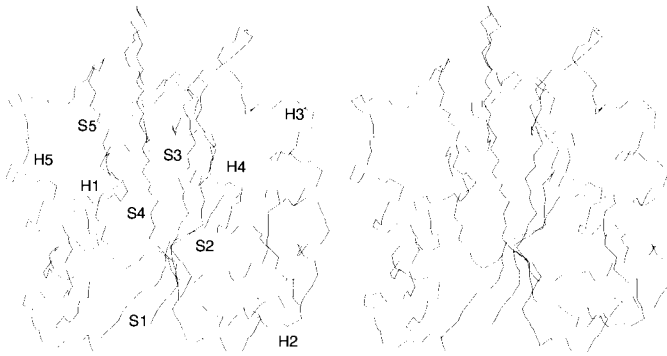
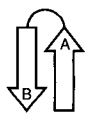


Figure 6: Stereoscopic View of Flavodoxin.
This is the backbone of Flavodoxin with its ten secondary structures.



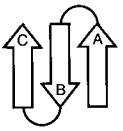

```
hairpin (A, B) :-
   strands ([A, B]),
   follows (A, B),
   adjacent (A, B),
   antiparallel (A, B).
```

Figure 7: Hairpin of β-Strands.
Schematic drawing of a two-stranded β-hairpin. On the right the corresponding Prolog clause is shown. Arrows indicate the orientation of β-strands.

```
cross_over (strand (1), strand (2), right)
cross_over (strand (2), strand (3), right)
cross_over (strand (3), strand (4), right)
cross_over (strand (4), strand (5), right)
                                    # cross overs in β-strand connections
bending (strand (1), slight)
bending (strand (2), medium)
bending (strand (3), medium)
bending (strand (4), medium)
bending (strand (5), medium)
                                    # bending of secondary structures
twisting (strand (1), slight)
twisting (strand (2), middle)
twisting (strand (3), middle)
twisting (strand (4), slight)
twisting (strand (5), middle)
                                    # twisting of secondary structures
```

Example 5: Flavodoxin in Prolog.
Various properties of secondary structures in the Flavodoxin domain are implemented as Prolog clauses. Lines with a "#" are comments and not part of the Prolog code.

C. J. Rawlings, W. R. Taylor, J. Nyakairo, J. Fox and M. J. E. Sternberg showed in the above cited paper how Prolog can be used to retrieve super-secondary structures from a data base of known structures. Their approach is elegant, first, because complex super-secondary structures of only β-strands can be nicely defined by a combination of secondary structures or simple super-secondary structures, and second, because in Prolog the source code of a super-secondary structure reads




```
meander (A, B, C):-
   hairpin (A, B),
   hairpin (B, C).
```

Figure 8: Meander of β-Strands.
Schematic drawing of a three-stranded β-*meander* with the corresponding Prolog clause.

almost like its ordinary, semantic description in English. Some examples will follow to illustrate the advantage of this approach. The most simple case of a super-secondary structure with only β-strands is a *hairpin*. A hairpin is defined as two directly adjacent β-strands connected by a short loop and which are oriented anti-parallel. Figure 7 shows a cartoon of a hairpin super-secondary structure projected on two dimensions and the corresponding Prolog clause. For the Prolog definition of a hairpin to be complete four other predicates have to be defined (remember, in Prolog variables start with a capital letter):

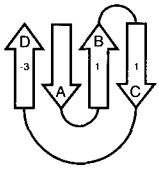
1. STRANDS ([A, B]) is a predicate that is true if all members of the argument list are known to be strands. This can be implemented as a collection of STRAND (S) facts and the STRANDS (X) predicate:

```
strand (s1).
strand (s2).
strand (s3).
strands ([X]) :- strand (X).
strands ([X|Y]) :- strand (X), strands (Y).
```

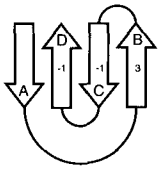
2. FOLLOWS (A, B) is true if two β-strands A and B follow each other directly in sequence with no other strands or helices on the primary structure between them. The FOLLOWS (A, B) clauses will have to be entered into the program before Prolog can "reason" about hairpins. Alternatively, strand and helix clauses can be automatically generated from a Brookhaven protein database file using inter-atomic distances in space and the secondary structure positions along the primary structure.
3. Similarly, ADJACENT (A, B) clauses have to be provided.
4. Facts of the form ANTIPARALLEL (A, B) must be entered which is more difficult to carry out automatically because one has to calculate the angle between idealised axes of the two strands. If the angle is close to -180.0° then two strands are defined to run anti-parallel.

If there is a knowledge base of proteins in terms of FOLLOWS, ADJACENT and ANTIPARALLEL then Prolog can be used to automatically derive all existing hairpins in that database based only on the hairpin definition of Figure 7. Similarly, one can ask the Prolog interpreter whether a particular pair of β-strands forms a hairpin. – Adjoining another strand to a hairpin gives a *meander* which can be elegantly defined by a combination of two hairpins (Figure 8).

Attaching one more β-strand to a *meander* yields the so-called *greek key* super-secondary structure which acquired its name from the similarity that this topology shares with one type of ornament on early Greek pottery. In proteins the *greek key*

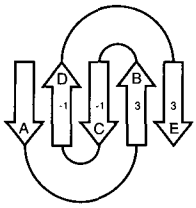



```
greek_key ([1, 1, -3], A, B, C, D):-
   meander (A, B, C),
   strand (D),
   follows (C, D),
   adjacent (A, D),
   antiparallel (A, D),
   D ≠ B.
```




```
greek_key ([3, -1, -1], A, B, C, D):-
   meander (B, C, D),
   strand (A),
   follows (A, B),
   adjacent (A, D),
   antiparallel (A, D),
   A ≠ C.
```

Figure 9: *Greek Key* Motifs (3, -1, -1) and (1, 1, -3) with their Prolog Definitions.



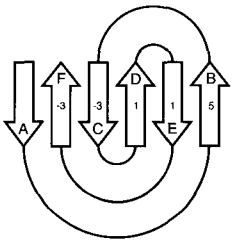

```
greek_key ([3, -1, -1, 3], A, B, C, D, E):-
   greek_key ([3, -1, -1], A, B, C, D),
   greek_key ([1, 1, -3], B, C, D, E).
```

Figure 10: *Greek Key* Motif (3, -1, -1, 3) and Prolog Definition.

motif can be found in different forms which is why a special notation was invented for *antiparallel* β-*sheets*. The first β-strand is defined to point downwards. For a *positive topological code* the second β-strand is placed *to the right* of the first β-strand. The topological code is the number of β-strands between the first and the second β-strand plus one. So, if the β-strands are immediately adjacent in space the topological code is ± 1. If there are more β-strands between two strands the topological code is greater than one. A strand located *on the right* from the strand preceding in sequence has a *positive* topological code (and vice versa). The sign of the topological code changes only when a change in direction of assembly of the β-sheet occurs. For antiparallel β-sheets adjacent strands have always alternating orientation. Figure 9 shows two different *greek key* super-secondary structures with their corresponding Prolog definitions.

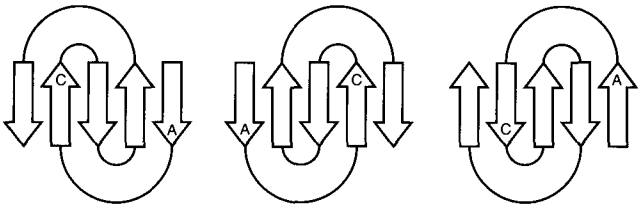
This building blocks approach to define super-secondary structures can be extended to more complicated *greek key* motifs (Figure 10). Another anti-parallel super-secondary structure made entirely of β-strands is the *jelly roll* motif (Figure 11). The corresponding Prolog clauses are intuitively understandable.

The definition of *handedness* of a *greek key* motif can be implemented by another predicate. To simply state that right-handedness is equivalent to the first topological code being positive would be misleading because, for example, GREEK_KEY ([3, -1, -1, 3)]) and GREEK_KEY ([-3, 1, 1, -3]) have in fact the same topolo-




```
jelly_roll ([5, -3, 1, 1, -3], A, B, C, D, E, F]):-
   greek_key ([3, -1, -1, 3], B, C, D, E, F),
   strand (A),
   follows (A, B),
   adjacent (A, F),
   antiparallel (A, F).
```

Figure 11: *Jelly Roll* Motif (5, -3, 1, 1, -3) and Prolog Definition.



Left Right (A down) Right (A up)

Figure 12: Handedness of *Greek Key* Motifs.
One left-handed *greek key* motif (figure on the left) and two identical, right-handed but by 180.0° rotationally translated *greek key* motifs (figures on the right and in the middle) are shown.

gy but are rotated by 180.0° against each other (Figure 12). Instead, a practicable definition for right-handedness in Prolog could be:

```
handed (right, strands (A, B, C | Rest])):-
    left_of (A, C),
    orientation (A, down).
handed (right, strands (A, B, C | Rest])):-
    right_of (A, C),
    orientation (A, up).
```

With these definitions (and complementary clauses for left-handed chirality) we can extend the GREEK_KEY predicate to include super-secondary structures with chirality:

```
greek_key (Hand, Type, Strands):-
    greek_key (Type, Strands),
    handed (Hand, Strands).
```

Having acquired the expertise to encode super-secondary structures and their properties in Prolog what can we do with it? There are three main directions to turn to:

```
meander (A, B, C) :-
     strand (A),
     strand (B),
     follows (A, B),
     adjacent (A, B),
     antiparallel (A, B),
     strand (C),
     follows (B, C),
     adjacent (B, C),
     antiparallel (B, C).
```

```
greek_key ([3, -1, -1], A, B, C, D) :-
     strand (B),
     strand (C),
     follows (B, C),
     adjacent (B, C),
     antiparallel (B, C),
     strand (D),
     follows (C, D),
     adjacent (C, D),
     antiparallel (C, D),
     strand (A),
     follows (A, B),
     adjacent (A, D),
     antiparallel (A, D),
     A \== C.
```

Example 6: Partial Evaluation of MEANDER and GREEK_KEY Predicates.
Partial Evaluation recursively replaces all predicates on the right (condition) side of a rule by their definitions until only unit clauses remain. Notice the absence of calls to the HAIRPIN and MEANDER predicates as used in the original definitions of Figures 8 and 9.

1. Use the implemented topological definitions to verify the compatibility of actual protein conformations with a new topological concept or constraint or post a query to the Prolog interpreter to search for instances of a particular topological concept. This is asking Prolog "Are there any objects with a certain combination of properties?" and "Which objects fulfil certain constraints?" The advantage compared to relational data base management systems is that, given a pre-defined set of objects and properties, queries can be made by concatenating and / or partially instantiating intuitively understandable topological concepts. No new tables or indices will have to be constructed. Symbolic descriptions can be used to define rather abstract concepts like *handedness* in a most concise and straight-forward way. To express the same concept in a traditional procedural programming language would be more complicated and requires more code than with Prolog.
2. Use *partial evaluation*[8] to automatically expand more complex predicates into their basic constituents (unit clauses). The resulting Prolog code is longer than the original version but it can be executed faster because it does not require the overhead for following nested predicate calls. Also, expanded clauses allow immediate identification of the fundamental concepts in a predicate. Example 6 shows the expanded clauses of the MEANDER and GREEK_KEY predicates. Partial evaluation can also be used for transformation and compression of a knowledge base of predicates for subsequent portation to another programming language or hardware platform.

8 A. Takeuchi, K. Furukawa, *Partial Evaluation of Prolog Programs and its Application to Meta Programming*, **ICOT Technical Report TR-126**, ICOT, Tokyo, Japan, 1985.

3. Invent new concepts and see if they are useful to describe existing or new protein classes, structures or functions. One way to proceed along this line is exemplified in the next section.

3.2.3. Protein Topology Prediction through Constraint Satisfaction

Protein topology describes the sequence, relative location and orientation of secondary structures. The previous section had some examples of β-strand super-secondary structures. Predicting the topology of super-secondary structures requires accurate information on the extent and location of secondary structures. There are a number of secondary structure prediction schemes, some of which will be discussed in later sections. Unfortunately, secondary structure prediction algorithms are not yet accurate enough to be used as a reliable basis for protein topology prediction. Fortunately, however, complete accuracy is not essential as long as the secondary structure of the protein is correct on a segmental level (i.e. predicted segments overlap with the true ones). Many topological folding rules e.g. those relating to handedness, orientation or strand position can still be applied to the predicted structure. With these limitations in mind we turn to an approach of D. A. Clark, J. Shirazi and C. J. Rawlings, which uses a Prolog implementation of folding constraints and a constraint based search algorithm to discover plausible super-secondary topologies[9].

The combinatorial analysis of α/β-sheets with n strands reveals that there are $\frac{n!4^{n-1}}{2}$ conceivable strand topologies, assuming left- or right-handed connections which can also be parallel or antiparallel for each pair of strands[10]. This term is the product of the number of rotationally invariant combinations for ordering n strands which is $\frac{n!}{2}$ and the number of different connections between n strands which is 4^{n-1} since there are four ways to connect two adjacent strands: parallel or antiparallel and right-handed or left-handed. This calculation includes left-handed antiparallel hairpin connections where the connection is supposed to encircle the whole β-sheet. However, those structures have not yet been observed in real proteins. Table 1 shows some numerical values for these expressions. One can see that for practical purposes even the number of topologies without left-handed hairpin connections is too large to be examined individually. It is therefore desirable to have a method that reduces the topological hypothesis space of β-sheets.

One way to automate the selection of reasonable β-strand topologies is given in the following three steps:

1) Define and implement β-strand topology folding constraints in Prolog.
2) Implement a constraint based breadth-first generate and test algorithm.

9 D. A. Clark, J. Shirazi, C. J. Rawlings, *Protein topology prediction through constraint-based search and the evaluation of topological folding rules*, **Protein Engineering**, vol 4, no 7, pp. 751-760, 1991.

10 J. Shirazi, D. A. Clark, C. J. Rawlings, **ICRF Biomedical Computing Unit Technical Report**, July, 1990.

Strands	2	3	4	5	6	7	8
$\frac{n!3^{n-1}}{2}$	3	27	324	4860	87480	$1.8 \cdot 10^6$	$4.4 \cdot 10^7$
$\frac{n!4^{n-1}}{2}$	4	48	768	15360	368640	$1.0 \cdot 10^7$	$3.3 \cdot 10^8$

Table 1: Number of β-Strand Topologies.
The counts of different topologies for two to eight β-strands are listed. The first set does not make a distinction between left- and right-handed hairpin connections but the second does. The values for seven and eight strands are rounded.

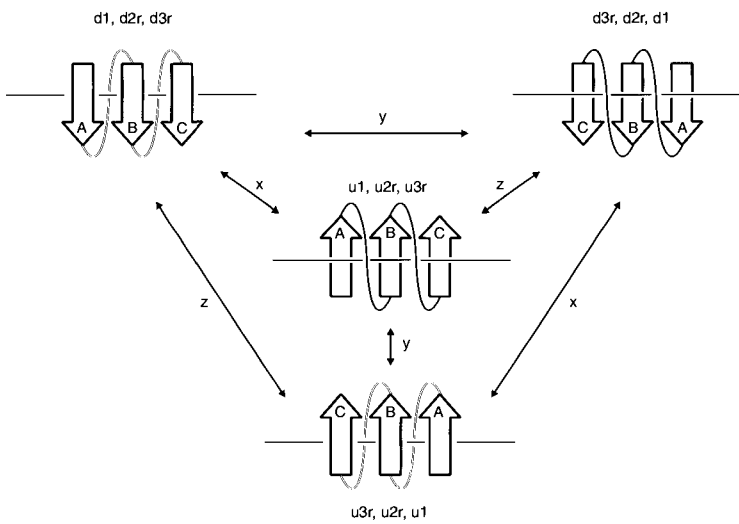


Figure 13: Rotational Invariance.
Four equivalent representations of one and the same β-sheet are shown with the notation of Clark *et al.* Rotation about the x, y, and z axis alters the orientation of the β-sheet. Large, open arrows indicate the direction of β-strands.

3) Use the programs from 1 and 2 to predict the topology of a set of new secondary structures.

In step 3, the template of an unknown α/β-sheet can be entered and valid instances which satisfy the given constraints are found. A number of constraints for α/β-sheets that can be used here have been described in the literature. For the remainder of this section only *parallel α/β-sheets* are considered. In the following paragraphs several constraints are collected together with their Prolog implementations. Constraints are implemented in their negated form, i.e. when a constraint rule succeeds the corresponding topological hypothesis is eliminated.

Constraint C1. *In parallel pairs of β-strands the β-α-β and β-coil-β connections are always right handed.* Figure 13 shows right-handed conformations. To determine handedness of a β-sheet using your right hand proceed as follows: let the index finger point along the first β-strand (extending from its N to C terminus), then

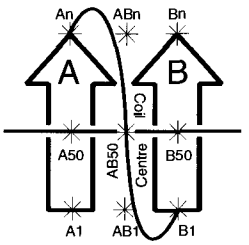


Figure 14: Auxiliary Vectors for the Calculation of Handedness in Parallel β-Sheets. Labels correspond to variable names in the Prolog source code for VECTOR_DOT_PRODUCT. In the idealised situation depicted here the points AB50, CENTRE and COIL coincide, which is not true for most real applications.

position the hand in such a way that the thumb points along the winding axis of the β-α-β or β-coil-β loop. If the remaining three fingers of the right hand follow the winding of the "super-helix" of β-α-β (or β-coil-β) loops the conformation is defined to be right-handed, or left-handed otherwise. A fragment of Prolog code for the definition of handedness in parallel pairs of β-strands is given below. The "#" sign denotes a comment and is not part of the source code. Figure 14 has a sketch with some of the points and vectors used in the calculation. If the predicate CONSTRAINT (C1, [S1, COIL, S2]) is fulfilled for two successive β-strands with one coil in between them these strands violate constraint C1 and the topology is excluded from the set of valid candidates.

```
constraint (c1, [StrandA, Coil, StrandB]) :-
     handed (left, StrandA, Coil, StrandB).

handed (Handedness, A, Coil, B) :-
     strands ([A, B]),
     vector_dot_product (A, Coil, B, Value),
     handedness (Value, Handedness).

handedness (Value, right) :-
     Value > 0.

handedness (Value, left) :-
     Value < 0.

vector_dot_product (A, Coil, B, Value) :-
     line (A, A1, An),      # fit a line for β_-strands A and B
     line (B, B1, Bn),      # with start A1, B1 and end An, Bn
     middle_of (A1, B1, AB1),      # middle points of / between strands
     middle_of (An, Bn, ABn),
     middle_of (AB1, ABn, Centre),
     middle_of (A1, An, A50),
     middle_of (B1, Bn, B50),
     vector (Centre, Coil, Centre_to_Coil),      # vector Centre to Coil
     vector (AB1, ABn, AB1n),      # various middle point vectors
     vector (A50, B50, AB50),
     vec_cross (AB50, AB1n, Normal),      # vector cross product
     vec_dot (Normal, Centre_to_Coil, Value).      # vector dot product
```

The predicates LINE (STRAND, START, END), MIDDLE_OF (POINT1, POINT2, MIDDLE), VECTOR (FROM_A, TO_B, VECTOR_DIFFERENCE), VEC_CROSS (VECTOR1, VECTOR2, PRODUCT) and VEC_DOT (VECTOR1, VECTOR2, RESULT) have to be implemented according to their standard mathematical definitions because they are not contained in the built-in set of Prolog predicates of most Prolog interpreters. For example, the code for VEC_CROSS (VECTOR1, VECTOR2, PRODUCT) with vectors represented as a list of three numbers could be:

```
vec_cross ([UX, UY, UZ], [VX, VY, VZ], [RX, RY, RZ]) :-
     RX is UY * VZ - VY * UZ,
     RY is VX * UZ - UX * VZ,
     RZ is UX * VY - VX * UY.
```

Constraint C2. *The initial β-strand in the sequence is not an edge strand in a β-sheet.* The Prolog definition for this constraint could be written as follows.

```
constraint (c2, [First_Strand | More_Strands]) :-
     position (1, First_Strand).

constraint (c2, [First_Strand | More_Strands]) :-
     number_of_strands ([First_Strand | More_Strands], N),
     position (N, First_Strand).
```

Suppose one of these clauses for constraint C2 would become true for a particular set of β-strands. In this case the calling program can exclude the current combination of β-strands because it violates constraint C2. It is not possible to represent constraint C2 in one clause because the body of a Prolog rule normally consists only of conjunctions whereas here a disjunction of two alternatives is required to capture the full meaning of constraint C2.

Constraint C3. *Only one change in winding direction occurs.* For example, the *greek key* motif (Figure 10, above) has 2 changes in winding direction and the *jelly roll* motif (Figure 11, above) has 3 changes. This does not conflict with constraint C3, however, because C3 applies only to *parallel* α/β-sheets and the *greek key* and *jelly roll* motives are *antiparallel* β-sheets. The Prolog code for constraint C3 could be:

```
constraint (c3, Strands, Connection_Matrix) :-
     direction_changes (Connection_Matrix, N),
     N > 1.

direction_changes ([K], 0).                # hairpin

direction_changes ([K, L], 0) :-           # 3 strands w/o change
     T is K * L,                           # determine, whether
     T > 0 .                               # both have the same sign

direction_changes ([K, L], 1) :-           # 3 strands with change
     T is K * L,
     T < 0 .

direction_changes ([K, L | Rest], X) :- # recursive decomposition
     direction_changes ([L | Rest], S),
     direction_changes ([K, L], T),
     X is T + S.
```

For illustration, a call to DIRECTION_CHANGES with the fictitious topological connection matrix [1, -2, -3, 4, -5] is shown which produces the following trace output with the correct solution of 3 winding changes in the last line. As before, ">" means entering a predicate call and "+" means leaving the call with success. Anonymous variables have been renamed to "X*n*" and nested calls are indented to emphasise the calling hierarchy.

```
>direction_changes ([1, -2, -3, 4, -5], X1)
 >direction_changes ([-2, -3, 4, -5], X2)
  >direction_changes ([-3, 4, -5], X3)
   >direction_changes ([4, -5], X4)
   +direction_changes ([4, -5], 1)
   >direction_changes ([-3, 4], X5)
   +direction_changes ([-3, 4], 1)
  +direction_changes ([-3, 4, -5], 2)
  >direction_changes ([-2, -3], X6)
  +direction_changes ([-2, -3], 0)
 +direction_changes ([-2, -3, 4, -5], 2)
 >direction_changes ([1, -2], X7)
 +direction_changes ([1, -2], 1)
+direction_changes ([1, -2, -3, 4, -5], 3)

Result: 3
```

Constraint C4. β-*Strands with conserved patterns in primary sequence lie adjacent in a sheet.* Conserved sequence patterns are constrained by their 3-dimensional structure, biochemical function or by surrounding fragments. For parallel β-sheets, the later case implies that there must be another structure that is geometrically constrained by the strand with the conserved pattern. A Prolog program to implement constraint C4 must first search for conserved β-strands and second, determine whether the strands lie next to each other. One solution for programming this task in Prolog is given below.

```
constraint (c4, Strands) :-
     conserved (Strands, Conserved_Strands),    # any conserved strands?
     not (adjacent_pairs (Conserved_Strands)). # all of them adjacent?

conserved ([Strand], [Strand]) :-
     has_conserved_pattern (Strand).
conserved ([Strand], []).
conserved ([Strand | Rest], [Strand | Conserved_Strands]) :-
     has_conserved_pattern (Strand),
     conserved (Rest, Conserved_Strands).
conserved ([Strand | Rest], Conserved_Strands) :-
     conserved (Rest, Conserved_Strands).

adjacent_pairs ([S1, S2]) :-                    # adjacent is reflexive
     adjacent (S1, S2).
adjacent_pairs ([S1, S2]) :-
     adjacent (S2, S1).
adjacent_pairs ([S1 | Rest]) :-
     member (X, Rest),
     adjacent (S1, X),
     remove (X, Rest, New_Rest),
     adjacent_pairs (New_Rest).
```

```
member (Element, [Element | Rest]).
member (Element, [First | Rest]) :- member (Element, Rest).
remove (X, [], []).
remove (X, [X | Rest], New_Rest) :- remove (X, Rest, New_Rest).
remove (X, [Y | Rest], [Y | New_Rest]) :- remove (X, Rest, New_Rest).
```

Auxiliary predicates are MEMBER, which checks whether an ELEMENT occurs in a list, and REMOVE, which removes every occurrence of an element X in a list. A trace of a call of REMOVE illustrates the recursive computation:

```
remove (a, [a, b, c, d, a, f, g], X).

>remove (a, [a, b, c, d, a, f, g], X1)
 >remove (a, [b, c, d, a, f, g], X2)
  >remove (a, [c, d, a, f, g], X3)
   >remove (a, [d, a, f, g], X4)
    >remove (a, [a, f, g], X5)
     >remove (a, [f, g], X6)
      >remove (a, [g], X7)
       >remove (a, [], X8)
       +remove (a, [], [])
      +remove (a, [g], [g])
     +remove (a, [f, g], [f, g])
    +remove (a, [a, f, g], [f, g])
   +remove (a, [d, a, f, g], [d, f, g])
  +remove (a, [c, d, a, f, g], [c, d, f, g])
 +remove (a, [b, c, d, a, f, g], [b, c, d, f, g])
+remove (a, [a, b, c, d, a, f, g], [b, c, d, f, g])

Result: [b, c, d, f, g]
```

The predicate CONSERVED defines which β-strands have conserved patterns. Assuming that the β-strands S1, S2, S4, and S5 show a conserved pattern this would be expressed in Prolog as:

```
has_conserved_pattern (s1).
has_conserved_pattern (s2).
has_conserved_pattern (s4).
has_conserved_pattern (s5).
```

The following call to CONSERVED produces the subsequent trace output and result:

```
conserved ([s1, s2, s3, s4, s5, s6], X).

>conserved ([s1, s2, s3, s4, s5, s6], X1)
 >conserved ([s2, s3, s4, s5, s6], X2)
  >conserved ([s3, s4, s5, s6], X3)
   >conserved ([s4, s5, s6], X4)
    >conserved ([s5, s6], X5)
     >conserved ([s6], X6)
     +conserved ([s6], [])
    +conserved ([s5, s6], [s5])
   +conserved ([s4, s5, s6], [s4, s5])
  +conserved ([s3, s4, s5, s6], [s4, s5])
 +conserved ([s2, s3, s4, s5, s6], [s2, s4, s5])
+conserved ([s1, s2, s3, s4, s5, s6], [s1, s2, s4, s5])
```

```
Result: [s1, s2, s4, s5]
```

Finally, the predicate ADJACENT_PAIRS tests whether conserved β-strands in a list are adjacent. Adjacency is a reflexive property, i.e. if β-strand S1 is adjacent to β-strand S2 then β-strand S2 is also adjacent to β-strand S1. One β-strand can be adjacent to maximally two other strands. In Prolog this is written as:

```
adjacent (s1, s2).
adjacent (s4, s5).
adjacent (s2, s6).
```

The following call to ADJACENT_PAIRS produces the subsequent trace output and result:

```
adjacent_pairs ([s1, s4, s2, s5, s6]).
>adjacent_pairs ([s1, s4, s2, s5, s6])
 >adjacent_pairs ([s4, s5, s6])
  >adjacent_pairs ([s6])
  -adjacent_pairs ([s6])
  >adjacent_pairs ([s5, s6])
  -adjacent_pairs ([s5, s6])

 -adjacent_pairs ([s4, s5, s6])
 >adjacent_pairs ([s4, s2, s5, s6])
  >adjacent_pairs ([s2, s6])
   +adjacent_pairs ([s2, s6])
 +adjacent_pairs ([s4, s2, s5, s6])
+adjacent_pairs ([s1, s4, s2, s5, s6])

Result: [s1, s4, s2, s5, s6]
```

A predicate exiting with "–" means failure of the inference engine to prove the corresponding predicate call and backtracking is attempted to find alternative solutions. The symbols ">" and "+" mean entering a predicate call and successful completion, respectively (as in the previous examples above).

Constraint C5. *In parallel α/β-sheets all β-strands lie parallel in the sheet.*

Constraint C6. *Unconserved β-strands are located at the edge of a β-sheet.* As was mentioned for constraint C4 less evolutionary pressure is exerted on outer β-strands than on inner β-strands (unless an outer β-strand is part of an active site). Therefore, outer β-strands need not be as highly conserved as inner β-strands. In Prolog this constraint can be implemented as follows:

```
constraint (c6, Strands) :-
     unconserved (Strands, Unconserved_Strands),
     not (outer_strands (Unconserved_Strands)).

unconserved ([Strand], [Strand]) :-
     not (has_conserved_pattern (Strand)).
unconserved ([Strand], [ ]) :-
     has_conserved_pattern (Strand).
unconserved ([Strand | Rest], [Strand | Unconserved_Strands]) :-
     not (has_conserved_pattern (Strand)),
     unconserved (Rest, Unconserved_Strands).
unconserved ([Strand | Rest], Unconserved_Strands) :-
     has_conserved_pattern (Strand),
```

```
     unconserved (Rest, Unconserved_Strands).

outer_strands (Strands, Size, Lower_Boundary) :-
     Upper_Boundary IS Size - Lower_Boundary,
     check_outer_strands (Strands, Lower_Boundary, Upper_Boundary).

check_outer_strands ([Strand], Lower_Boundary, Upper_Boundary) :-
     position (Strand, N),    # returns the position
     N ≤ Lower_Boundary,      # of a Strand
     N ≥ Upper_Boundary.

check_outer_strands ([Strand | Rest], L_Boundary, U_Boundary) :-
     check_outer_strands ([Strand], L_Boundary, U_Boundary),
     check_outer_strands (Rest, L_Boundary, U_Boundary).
```

Many other folding rules have been proposed in literature some of which are not completely independent from the constraints listed above. Here are four more constraints for general α/β-sheets. As an exercise for the interested the reader is invited to invent his or her own Prolog implementations.

Constraint C7. *β-Strands are ordered by hydrophobicity.* The most hydrophobic strands should be central with hydrophobicity decreasing towards the outside of the β-sheet.

Constraint C8. *Parallel β-coil-β connections must have at least 10 residues in the coil.* This number of residues is required to span the distance between the end of the first β-strand and the beginning of the second.

Constraint C9. *Insertion and deletion of amino acid residues tend to occur on outer β-strands.* This is a consequence of the need for co-ordinated changes in the central β-strands.

Constraint C10. *Long secondary structures (α-helices or β-strands) should pack approximately parallel or antiparallel.* Segments following each other directly in sequence are preferentially oriented anti-parallel. This constraint favours compact folding patterns.

The next step, as outlined above, is to devise an algorithm to search for valid conformations that observe some or all of the constraints listed above. D. A. Clark *et al* used a *breadth-first tree search* algorithm with *forward pruning.* In general, *constraint satisfaction algorithms* consist of the following four components:

1) *Objects* to be constrained, here the *n* β-strands of a β-sheet.
2) *Variables* to describe the objects, e.g. here *orientation*, *chirality*.
3) *Values* for those variables, e.g. here *up* or *down*, *left* or *right*.
4) *Constraints* on values of those variables.

There are two different approaches for generating a set of properly constrained objects:

- *Tree search* methods traverse the search space sequentially while instantiating the variables to be constrained in some order.
- *Arc consistency* methods generate consistent sets of values for each variable in an object.

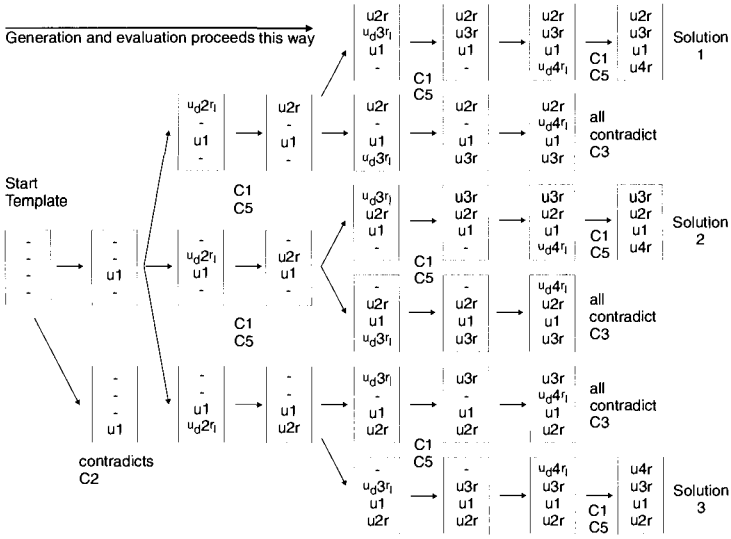


Figure 15: Example Trace of Constraint Satisfaction Algorithm.
Subscripts and superscripts denote alternate topologies (U, D for up and down; R, L for right and left). Extending and evaluating the patterns of β-strands proceeds from left to right. Constraints C1, C2, C3, and C5 are used to prune the search tree where noted. Altogether, 18 topologies have to be evaluated.

The constraint satisfaction algorithm CBS1 of D. A. Clark *et al* is a mixture of the two approaches. The main procedure is a recursive call to a function that takes as input a set of filled or partly instantiated templates and then adds one new strand to each of the empty positions in either orientation (up or down) and chirality (left or right). Each of the new templates is then tested against the set of active constraints and those templates that violate any constraint are eliminated. Constraints can include information on conservation of residues, hydrophobicity, topological patterns (etc.) as demonstrated in the examples above. To circumvent inspection of rotationally invariant topologies the first β-strand is by convention assigned to point upwards and to be located on the left side of the β-sheet (i.e. left from the central β-strand). Figure 15 shows an example trace of the CBS1 algorithm for a four-stranded β-sheet.

The three solutions for valid β-strand topologies in Figure 15 are [U4R, U1, U3R, U2R], [U4R, U1, U2R, U3R] and [U2R, U1, U3R, U4R]. CBS1 has to evaluate only 18 patterns in contrast to a potential 768 alternatives for an exhaustive search (or 324 for non-handed hairpin connections, respectively). This is a clear advantage over a simple, unconditional generate and test algorithm. The benefit becomes even larger for β-sheets with more β-strands. In one application on ATPase proteins CBS1 found one more topological hypothesis than was known from a previous, manual analysis which had been published earlier. The extra finding was produced because constraint C10 had been left out intentionally in that run of CBS1.

3.2.4. Inductive Logic Programming in Molecular Bioinformatics

Inductive logic programming (ILP) is a new way of automatically eliciting hidden knowledge with the help of a computer[11,12,13]. Having gained the attention of a wider audience only during the past few years, the roots of ILP had already been laid down in the 1950's[14]. The intriguing capability of ILP is the combination of logical reasoning and inductive rule generation, both in a symbolical manner. This means ILP can be used to discover new rules based on a set of observations *and* on background knowledge about the application domain. All logical implications that can be derived from information in the knowledge base are considered. The result of an ILP application is a set of new facts and rules about the domain in question.

Following the framework of Plotkin[15] the inductive learning task can be described using a background theory B, a set of positive examples E^+, a set of negative examples E^-, a hypothesis H and a partial ordering "$\leq$" to which the following conditions apply:

1. The background knowledge B should not *alone* explain all positive examples E^+, or mathematically (with the "$\mapsto$" symbol meaning "logically proves"): $\neg(B \mapsto E^+)$. Otherwise, the problem has already been solved. This is called the condition of *prior necessity*.
2. The background knowledge B should be consistent with all negative and positive examples E^+ and E^-. With "$\perp$" as the symbol for falsity this reads mathematically: $\neg(B \wedge E^+ \wedge E^- \mapsto \perp)$. This condition is called *prior satisfiability*.
3. The background knowledge B and the hypothesis H should *together* explain all positive examples E^+. This condition is called *posterior sufficiency*: $B \wedge H \mapsto E^+$.
4. The background knowledge B and the hypothesis H together should not contradict any of the negative examples E^-. This condition is called *strong posterior consistency:* $\neg(B \wedge H \wedge E^- \mapsto \perp)$. Alternatively, the condition of *weak posterior consistency* requires only the background knowledge B and hypothesis H to be logically consistent: $\neg(B \wedge H \mapsto \perp)$. Weak consistency is used for systems that have to deal with noise.
5. If there are more than one hypotheses that fulfil these requirements the result should be the most specific hypothesis. This assumes a measure of ordering for hypotheses $H1 \leq H2$ which says that $H2$ is more general than $H1$.

These five conditions capture all the logical requirements of an ILP system. The *prior necessity* and *posterior consistency* conditions can be checked with a theorem

11 S. Muggleton (Ed.), *Inductive Logic Programming*, **Proceedings of the ILP-91 International Workshop**, Viana de Castelo, Portugal, 2-4 March 1991.
12 S. Muggleton (Ed.), **Inductive Logic Programming**, Academic Press, London, 1992.
13 S. Muggleton (Ed.), *Inductive Logic Programming*, **Proceedings of the ILP-93 International Workshop**, Bled, Slovenia, 1.-3. April 1993.
14 C. Sammut, *The Origins of Inductive Logic Programming: A Prehistoric Tale*, in: **Inductive Logic Programming, Proceedings of the ILP-93 International Workshop**, S. Muggleton (Ed.), pp. 127-147, Bled, Slovenia, 1.-3. April 1993.
15 G. D. Plotkin, *A Note on Inductive Generalization*, in: **Machine Intelligence**, (Eds. B. Meltzer, D. Michie), chapter 8, pp. 153-163, American Elsevier, 1970.

Sufficiency: $B \wedge H \mapsto E^+ \Leftrightarrow$ Sufficiency*: $B \wedge \overline{E^+} \mapsto \overline{H}$
Syntactic Proof:
Given: $A \wedge B \mapsto C$
Deduction Theorem:
From $A \wedge B \mapsto C$ follows $A \mapsto (B \rightarrow C)$.
Remember: $X \rightarrow Y \Leftrightarrow \overline{X} \vee Y$
Hence, $A \mapsto \overline{B} \vee C \quad | \wedge \overline{C}$
$\Rightarrow A \wedge \overline{C} \mapsto \overline{B} \wedge \overline{C} \vee C \wedge \overline{C}$
$\Rightarrow A \wedge \overline{C} \mapsto \overline{B}$

Example 7: Revised Sufficiency Condition.
The condition of posterior sufficiency is transformed to allow explicit computation of $\neg H$. A, B and C are general sets of clauses.

A	B	A ∧ B	C	A ∧ B → C	¬C	A ∧ ¬C	¬B	A ∧ ¬C → ¬B
0	0	0	0	**1**	1	0	1	**1**
0	0	0	1	**1**	0	0	1	**1**
0	1	0	0	**1**	1	0	0	**1**
0	1	0	1	**1**	0	0	0	**1**
1	0	0	0	**1**	1	1	1	**1**
1	0	0	1	**1**	0	0	1	**1**
1	1	1	0	**0**	1	1	0	**0**
1	1	1	1	**1**	0	0	0	**1**

Table 2: Semantic Proof of Revised Posterior Sufficiency Condition.
The two versions of the sufficiency condition correspond to the fifth and ninth column, respectively. Both expressions have identical truth values.

prover. If all formulae are written as Horn clauses a Prolog interpreter can carry out this task. Only some minor changes (e.g. to provide iterative deepening) are needed in the program to ensure logical completeness.

A theorem prover can not directly derive the desired hypothesis H from B and E^+ because it would have to generate and test a huge number of hypotheses growing exponentially with the number of background clauses and positive examples. Because this is impractical and not efficient a different representation of the *posterior sufficiency* condition is used (marked below with an "*"), which is: $B \wedge \neg E^+ \mapsto \neg H$. A syntactic proof of the logical equivalency is given in Example 7.

Table 2 shows a semantic proof for the equivalence of the expressions (A ∧ B → C) and (A ∧ ¬C → ¬B). The revised posterior sufficiency condition allows the generation of new hypotheses by iterating through all positive examples of an application. Example 8 shows this for the case of a single positive example and a single hypothesis clause.

For illustration we will apply the principle of Example 8 to some facts about proteins in Example 9.

Sufficiency: $B \wedge H \mapsto E^+$

Sufficiency*: $B \wedge \overline{E^+} \mapsto \overline{H}$

Single example clause, single hypothesis clause:

$E^+ = h \leftarrow b_1, b_2 \ldots = h \vee \overline{b_1} \vee \overline{b_2} \ldots$

$H = h' \leftarrow b_1', b_2' \ldots = h' \vee \overline{b_1'} \vee \overline{b_2'} \ldots$

Substitute into Sufficiency*:

$B \wedge \overline{(h \vee \overline{b_1} \vee \overline{b_2} \ldots)} \mapsto \overline{(h' \vee \overline{b_1'} \vee \overline{b_2'} \ldots)} \Leftrightarrow$

$B \wedge \overline{h} \wedge b_1 \wedge b_2 \ldots \mapsto \overline{h'} \wedge b_1' \wedge b_2' \ldots$

Example 8: Inductive Inference of a Single Hypothesis from One Positive Example.
The hypothesis H cannot be directly derived from the original formula $B \wedge H \mapsto E^+$ because H is yet an unknown variable on the left side. Hence the revised sufficiency condition is applied. E^+ and H are decomposed into head (h, h') and body (b_i, b_i'). Using de Morgan's laws an explicit form for $\neg H$ can be derived. Negation of $\neg H$ then gives H.

$$B = \begin{cases} has_peptide_bonds(X) \leftarrow protein(X) \\ protein(X) \leftarrow serine_protease(X) \end{cases}$$

$E^+ = has_peptide_bonds(crambin) \leftarrow$

The ground unit consequences of $B \wedge \overline{E^+}$ are:

$C = \overline{protein(crambin)} \wedge \overline{serine_protease(crambin)} \wedge \overline{has_peptide_bonds(crambin)}$

This leads to the following three most specific clauses:

$$H = \begin{cases} protein(crambin) \leftarrow \\ serine_protease(crambin) \leftarrow \\ has_peptide_bonds(crambin) \leftarrow \end{cases}$$

Add any of these clauses to B and E^+ becomes a consequence of the new theory.

Example 9: Inductive Logic Reasoning on Proteins.
Two background rules and one positive example E^+ are to be processed. First, the ground unit consequence C = $B \wedge \neg E^+$ is formed. The "$\leftarrow$" sign means implication. If there is no condition then "$\leftarrow$" denotes a fact, or otherwise a rule. Negation of C gives two new hypotheses plus the original positive example itself. Any of these three facts can now be added to the background knowledge to explain or derive E^+.

If the positive example E^+ is a set of ground atoms (i.e. a conjunction), H can be derived according to the formulae in Example 10.

Example 11 shows an application of the principle in Example 10 to the well-known family relationships between Harry, John, Fred and Jane.

ILP can also be used to invent new predicates. Naturally, the program has no capability to propose proper English names for the new concepts invented but it can instantiate new predicates. This is done either by *and-introduction* or *or-introduction*. The general idea for both *and-* and *or-introduction* is to expand a statement with a tautology. Example 12 shows the mathematical background.

In Example 13 the method of predicate invention is used to invent the predicate LESS_THAN. Two facts are given as input to ILP: the background knowledge that

Set of ground atoms: $E^+ = e_1 \wedge e_2 \wedge e_3 \ldots$
$= B \wedge \overline{E^+}$
From Sufficiency* follows:
$B \wedge \overline{E^+} \mapsto \overline{E^+} \wedge C$
Substituting for $\overline{E^+}$ and rearranging gives:
$B \wedge \overline{E^+} \mapsto (\overline{e_1} \vee \overline{e_2} \vee \ldots) \wedge C \Leftrightarrow$
$B \wedge \overline{E^+} \mapsto (\overline{e_1} \wedge C) \vee (\overline{e_2} \wedge C) \vee \ldots$
Therefore is $H = (e_1 \vee \overline{C}) \wedge (e_2 \vee \overline{C}) \wedge \ldots$
Subsume set of clauses by most specific clause.

Example 10: Inductive Logic Inference with Multiple Examples.

$$B = \begin{cases} father(harry, john) \leftarrow \\ father(john, fred) \leftarrow \\ uncle(harry, jane) \leftarrow \end{cases}$$

$$E^+ = \begin{cases} parent(harry, john) \leftarrow \\ parent(john, fred) \leftarrow \end{cases}$$

The ground unit consequences of $B \wedge \overline{E^+}$ are:

$$C = father(harry, john) \wedge father(john, fred) \wedge uncle(harry, jane) \wedge \overline{parent(harry, john)} \wedge \overline{parent(john, fred)}$$

Remember: $P \leftarrow Q \Leftrightarrow P \vee \overline{Q}$
This leads to the following most specific cases:
$e_1 \vee \overline{C} = parent(harry, john) \leftarrow father(harry, john), father(john, fred), uncle(harry, jane)$
$e_2 \vee \overline{C} = parent(john, fred) \leftarrow father(harry, john), father(john, fred), uncle(harry, jane)$
The least general generalisation (*lgg*) is then:
$lgg(e_1 \vee \overline{C}, e_2 \vee \overline{C}) = parent(X, Y) \leftarrow father(X, Y), father(Z, V)$

Example 11: Illustration of Inductive Logic Inference with Multiple Examples E^+. Three background facts and two new positive examples are processed. ILP creates one new rule to accommodate the new examples. An algorithm called *least-general generalisation* subsumes the two resulting rules into one. See reference 15 for details on least-general generalisation.

the smallest number in a list with one element is always that element; and the positive example that "2" is the smallest number in the list "[2, 3]" (Prolog list syntax). First, the ground unit consequences are computed (remember: "ground" means predicates with fully instantiated arguments). Next, a new predicate NEW_PRED is assumed, which at first gets as arguments a combination of all arguments in C. Then, *and-introduction* is used to combine the negation of C and NEW_PRED according to the formula for H in Example 12. This leads to two most specific clauses for H. The first clause can be generalised to give the first clause of H'. The interpretation of this clause is "If a number X is the minimum of a list Z and Y is smaller that X, then Y is the minimum of a list that has Y plus all elements from list Z." Generalisation can be done by dropping arguments and substituting variables

Observational vocabulary: $O = V(E^+ \cup E^-)$
Theoretical vocabulary: $T = V(B) - O$
Invented vocabulary: $I = V(H) - (T \cup O)$
Predicate invention is successful if $I \neq \emptyset$.
AND-introduction: $X \equiv (X \wedge Y) \vee (X \wedge \overline{Y})$
OR-introduction: $X \equiv (X \vee Y) \wedge (X \vee \overline{Y})$

Let $C = B \wedge \overline{E^+}$ and p is a ground atom whose predicate symbol is not in $(T \cup O)$, then $B \wedge \overline{E^+} \mapsto C$ and $B \wedge \overline{E^+} \mapsto (p \wedge C) \vee (\overline{p} \wedge C)$
$H = (\overline{p} \vee \overline{C}) \wedge (p \vee \overline{C})$

Example 12: Principle of Predicate Invention in ILP.

Invention of a new predicate *less_than* to find the minimum element of a list:
$B = \min(X, [X]) \leftarrow$
$E^+ = \min(2, [2,3]) \leftarrow$
The gound unit consequences of $B \wedge \overline{E^+}$ are:
$C = \overline{\min(2, [2,3])} \wedge \min(2, [2]) \wedge \min(3, [3])$
Let $p = \mathit{new_pred}(2, 3, [2], [3], [2,3])$.
This leads to two most specific starting clauses for predicate invention:

$$H = \begin{cases} \min(2, [2,3]) \leftarrow \min(2, [2]), \min(3, [3]), \mathit{new_pred}(2, 3, [2], [3], [2,3]) \\ \mathit{new_pred}(2, 3, [2], [3], [2,3]) \vee \min(2, [2,3]) \leftarrow \min(2, [2]), \min(3, [3]) \end{cases}$$

Generalising and renaming the predicate symbols gives:

$$H' = \begin{cases} \min(Y, [Y|Z]) \leftarrow \min(X, Z), \mathit{less_than}(Y, X) \\ \mathit{less_than}(2, 3) \leftarrow \end{cases}$$

Example 13: Invention of the LESS_THAN Predicate.
See main text for details.

for constants. It remains for the human expert to recognise the similarity between NEW_PRED and the relation *less than* and to rename NEW_PRED correspondingly.

One implementation of ILP is Golem[16,17] of S. Muggleton and C. Feng. Golem takes as its input background knowledge B, negative (E^-) and positive examples (E^+) implemented as Prolog facts and performs a random sampling of the positive examples E^+ in order to build new hypotheses. Golem produces Prolog rules that are valid and consistent generalisations of the positive examples E^+ and which do not conflict with any of the negative examples E^-. All induction can be carried out in an interactive mode by entering all the examples and background knowledge online and then writing the rules to a file, or in batch mode, where Golem reads a background knowledge file, a file with positive examples and another file with neg-

16 S. Muggleton, C. Feng, in: **Proceedings of the First Conference on Algorithmic Learning Theory**, (Eds. S. Arikawa, S. Goto, S. Ohsuga, T. Yokomori), Jpn. Soc. Artificial Intelligence, pp. 368-381, 1990.

17 S. Muggleton, C. Feng, *Efficient induction of logic programs*, in: **Inductive Logic Programming**, (Ed. S. Muggleton), pp. 281-297, Academic Press, email steve@prg.oxford.ac.uk, 1992.

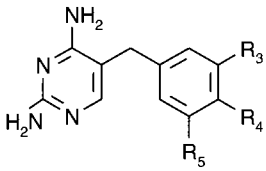


Figure 16: Trimethoprim Analogues.
Various substituents can be added at positions R_3, R_4 and R_5.

ative examples each containing ground instances in Edinburgh Prolog syntax. The basic algorithm of Golem proceeds as follows:

1) Get two positive examples E^+.
2) Compute properties common to both examples from E^+.
3) Generalise the common properties to a rule that covers both examples. If different rules are feasible try all of them successively.
4) Do this for other pairs of examples from E^+.
5) Evaluate the rules: how many positive (negative) examples are covered by each rule? The best rule covers the most positive examples and the least negative examples. Here, a trade-off between *accuracy* and *coverage* must be made.
6) Take the best rule and remove all positive examples from E^+ that are covered by that rule. Go back to step 1 until either all positive examples are covered or no further rules can be generated anymore.

Other well-known ILP systems are Foil[18] and Mobal[19].

3.2.4.1. Trimethoprim Analogues

Golem was used by R. D. King, S. Muggleton, R. A. Lewis and M. J. E. Sternberg[20] to analyse the quantitative structure / activity relationship (QSAR) of trimethoprim analogues (Figure 16). The training data in their application are chemical structures of 44 trimethoprim analogues. These substances have been observed to inhibit *E. coli* dihydrofolate reductase (Figure 17) and the questions are "What is the reaction mechanism?" and "Which parts of the molecule are responsible for inhibition?" Golem is used to find underlying common principles in their chemical composition that might be relevant and significant for their biochemical activity. After training Golem, the authors analyse the performance of the program with control data of 11 unseen compounds. The result is an improved accuracy over the Hansch linear regression model and a set of symbolic rules that are immediately understandable to a chemist. The rules describe polarity, flexibility, hydrogen-bonding (etc.) of the compounds to explain their reactivity.

18 R. Quinlan, *Learning logical definitions from relations*, **Machine Learning**, vol 5, pp. 239-266, 1990.
19 K. Morik, S. Wrobel, J. U. Kietz, W. Emde, **Knowledge Acquisition and Machine Learning: Theory, Methods, and Applications**, Academic Press, London, 1993.
20 R. D. King, S. Muggleton, R. A. Lewis, M. J. E. Sternberg, *Drug design by machine learning*, **Proc. Natl. Acad. Sci.**, 89, pp. 11322-11326, 1992.

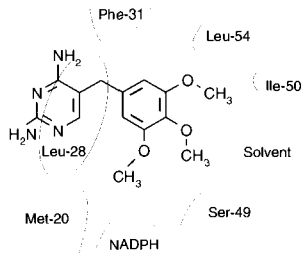


Figure 17: Location of Trimethoprim in Dihydrofolate Reductase and Neighbouring Amino Acid Residues.

Chemical compounds in Golem are implemented as Prolog clauses. A positive example for Golem is GREAT (D20, D15) which states that drug no. 20 has a higher activity than drug no. 15. This ordered pair of drugs is one positive example. The paired comparisons must later be converted into an absolute ranking which can be then be compared with other methods. A negative fact would be the opposite statement, GREAT (D15, D20). Background information is also encoded as Prolog predicates. For example, STRUCT (D35, NO2, NHCOCH3, H) says that drug no. 35 has NO_2 substituted in position 3, $NHCOCH_3$ in position 4 and no substitute in position 5. The substituents are further characterised by the following properties:

- *polarity* of α and β atoms of the substituent,
- *size* as measured by the volume of atom groups,
- *flexibility* as given by the number of rotatable bonds,
- *hydrogen donors* and *acceptors* expressed by their number and strength,
- presence and strength of σ-*electron acceptors*,
- presence and strength of π-*electron acceptors* and *donors*,
- *polarisability* of molecular orbitals.

One example for the syntax of these properties is POLAR (BR, POLAR3). This says that bromide has a polarity of 3. Golem had no explicit concept of numbers. Therefore, new symbols with an explicit *greater-less* relation had to be defined such as GREAT0_POLAR (POLAR1) which says that POLAR1 is greater than zero. These symbols must be used instead of ordinary numbers. This is certainly a shortcoming of Golem arising from this particular implementation.

With Prolog definitions like those presented above, the authors had 871 positive and 871 negative examples and 2976 background facts as the input to Golem. The program derived nine Prolog rules to predict the relative activities of two drugs. One of them is the following:

```
great (A, B) :- struc (A, D, E, F),
                struc (B, h, C, h),
                h_donor (D, h_don0),
                pi_donor (D, pi_don1),
                flex (D, G),
                less4_flex (G).
```

In common English this rule says that drug A is better than drug B if drug B has no substitution in positions R3 and R5; if drug A has a hydrogen donor with strength 0 in position R3; if drug A has a π-donor with strength 1 in position R3; and if drug A has a flexibility of less than 4 in position R3. This rule covered 119 positive training examples (pairs of activity relations) and none of the negative examples.

The call to the first STRUC (A, D, E, F) predicate establishes the compound A with substituents D, E and F. At this moment the substituents are yet uninstantiated variables. The second STRUC (B, H, C, H) predicate establishes compound B and states that it has hydrogen ("H") in positions R3 and R5 and a yet unspecified substituent at R4. The H_DONOR (D, H_DON0) predicate says that substituent D of compound A is a hydrogen donor of strength 0. It was necessary to write "H_DON0" instead of simply "0" because, as mentioned above, Golem had no explicit concept of integer numbers. Therefore, numbers had to be substituted by symbols with an explicitly defined ordering. The same holds for the π-donor and flexibility predicates. The strength of substituent D as a π-donor is constrained by predicate PI_DONOR (D, PI_DON1). Finally, the flexibility of substituent D is constrained to be less than four. This is expressed by LESS4_FLEX (G) in the same symbolical / numerical manner as explained above for H_DONOR.

On examination of the properties of the better drug A in all nine rules the authors make the observation that according to Golem's results the substituents at R3 and R5 should be of low hydrogen donor strength, have a small π-donor potential, be moderately flexible and of comparatively small size and have detectable polarity and polarisability. All these requirements are best fulfilled by the methoxy-group ($-OCH_3$) which is in perfect agreement with traditional chemical reasoning and the fact that this substituent is buried in a hydrophobic environment.

3.2.4.2. Drug Design of Thermolysin Inhibitors

An earlier work that applied the same type of learning from observation for the drug design of thermolysin inhibitors was done by L. di Pace, F. Fabrocini and G. Bolis[21]. Again, the problem is to derive a general theory that implies all positive examples, no negative examples and which is compatible with the background knowledge.

In this case, positive examples were pairs of thermolysin inhibitors ranked by their activity as measured by the thermodynamic equilibrium constant K_i. The algorithm of their system is divided into two steps, a *specialisation* step and a *generalisation* step. In the first step the system learns from differences between an inactive compound and a structurally similar but active compound. This allows identification of relevant features in the active compound. In the second step, the system tries to generalise a set of relevant features in active compounds to produce a more general model for structure related activity. Properties of the fragments and substituents are e.g. volume, hydrophobicity, hydrogen bond donor and spatial extension. The authors report that the model derived by their system is in good agree-

21 L. di Pace, F. Fabrocini, G. Bolis, *Shift of Bias in Learning from Drug Compounds: The Flemming Project*, in **Lecture Notes in Artificial Intelligence, Machine Learning - EWSL 1991**, (Ed. Y. Kodratoff), pp. 482-493, Springer, 1991.

ment with experimental data of the crystal structure of a thermolysin complex with the hydroxamic acid derivative inhibitor.

3.2.4.3. α-Helix Prediction

In another application, Golem was used for the prediction of α-helices in proteins[22]. Here, positive examples are random pairs of residues that participate in an α-helix of the protein training set. Each residue is specified in a corresponding ALPHA predicate, e.g. ALPHA (P155C, 105) which says that residue 105 in the protein P155C (cytochrome) occurs in a helix. Negative examples are pairs of non-helical residues. The background knowledge in Golem for this application includes the primary structure, e.g. encoded as POSITION (P155C, 119, P); the residues' preference for acting as a hydrogen-donor or acceptor; whether a residue is aromatic or not; its hydrophobicity, size and polarity and whether it is a lysine or a proline. Golem produced rules of the following kind.

```
alpha (PROTEIN, B) :- sequence (D, E, F, G, B, H, I, J, K),
    position (PROTEIN, D, N), not_aromatic (N), not_k (N),
    position (PROTEIN, F, L), hydrophobic (L),
    position (PROTEIN, G, O), not_aromatic (O), not_p (O),
    position (PROTEIN, B, C), not_aromatic (C), not_p (C),
    position (PROTEIN, H, Q), not_p (Q), not_k (Q),
    position (PROTEIN, I, M), hydrophobic (M), less_hydro (N, M),
                              less_volume (M, O),
    position (PROTEIN, K, P), not_aromatic (P), less_hydro (P, L).
```

The SEQUENCE predicate establishes a sequence of nine residues with D, E, F, G, B, H, I, J and K as the positions in primary structure of the residues N, L, O, C, Q, M and P of a protein named in the variable PROTEIN. Here, the letters N, L, O, ... are *not* one-letter codes for amino acids but simply one-letter variables in Prolog. The properties of the nine residues are determined by the predicates used in this rule. The first of the nine residues must not be aromatic nor lysine. The third residue must be hydrophobic, the fourth and fifth should not be aromatic or proline. The sixth residue can be anything but proline or lysine. Residue seven must have a hydrophobicity value lower than that of the first residue and be less voluminous than residue four. Residue nine must be not aromatic and have a lower hydrophobicity value than residue three.

Of all secondary structures, α-helices are the easiest to predict because they are formed by internal hydrogen bridges which can be detected as *local* patterns. Local intra-structural hydrogen bridges as in helices are not found in β-strands (in turns only to a smaller extent). These secondary structures are much more determined by surrounding residues. α-Helices are often amphiphilic, i.e. they are additionally constrained by local interactions with neighbouring residues or solvent. In this context it is not surprising that the authors prediction accuracy (*acc* = 80%) is higher than for a prediction over all three states of secondary structure. What actually is disturbing is that the authors use the label "Q3" to quote their performance values. This is incorrect and misleading because no three state prediction was done.

22 S. Muggleton, R.D. King, M.J.E. Sternberg, *Protein secondary structure prediction using logic-based machine learning*, **Protein Engineering**, vol 5, no 7, pp. 647-657, 1992.

Instead, only the positions correctly predicted to be helical (true positive predictions; *pos*) and the positions correctly predicted to be not helical (true negative predictions; *neg*) were counted: $acc = \frac{pos+neg}{all\ residues}$. Other and more appropriate measures for secondary structure prediction will be discussed later in more detail.

4. Machine Learning of Concepts in Molecular Biology

4.1. Methodology

This chapter examines several machine learning algorithms which are primarily not based on predicate logic and their application to the analysis of protein super-secondary structures and signal pattern recognition in DNA and protein sequences.

4.1.1. Learning Hierarchical Classifications: Cobweb/3

Cobweb/3[1] of K. McKusick and K. Thompson is an implementation of the Cobweb[2] and Classit[3] algorithms. These algorithms perform incremental, unsupervised concept formation using an information-theoretical evaluation function to constrain the search for a concept hierarchy that maximises the ability to make predictions about missing attributes of new objects. Input data consists of a set of objects described as vectors of either discrete or real-valued attributes. The output is a hierarchy of concepts. Cobweb/3 has a prediction mode for evaluating the predictive accuracy of a concept hierarchy on a set of objects that were previously not known to the algorithm. Cobweb/3 is written in (rather portable) Common Lisp and has been tested using Franz Lisp, Lucid Common Lisp, Harlequin LispWorks and AKCL on Sun 4s. For Franz Lisp, there is an optional user interface for visualising classification results and for building a concept hierarchy manually.

The type of classification and learning in Cobweb/3 is known as *conceptual clustering*[4]. There are five main characteristics of the Cobweb approach.

- *Hierarchical Organisation of Concepts*. Instead of intervals or sets of related examples Cobweb/3 forms a concept hierarchy where nodes represent concepts that are partially ordered by generality.

1 K. McKusick, K. Thompson, *COBWEB/3 – A Portable Implementation*, **Technical Report FIA-90-6-18-2**, NASA Ames Research Center, Artificial Intelligence Research Branch, June 20, 1990.

2 D. H. Fischer, *Knowledge Acquisition Via Incremental Conceptual Clustering*, Machine Learning, 2, 139-172, 1987.

3 J. H. Gennari, *Focused Concept Formation*, **Proceedings of the Sixth International Workshop on Machine Learning**, pp. 379-382, Ithaca, NY, Morgan Kaufmann, 1989.

4 R. S. Michalsky, R. Stepp, *Learning from Observation: Conceptual Clustering*, in **Machine Learning, An Artificial Intelligence Approach**, (Eds. R. S. Michalsky, J. G. Carbonell, T. M. Mitchell), San Mateo, CA, Morgan Kaufmann, 1983.

- *Hill Climbing*. Each new hierarchy learned is a function of a previous hierarchy. This means adding a new instance is inexpensive as it does not require recalculation of the entire classification set up so far.
- *Incremental Learning*. Cobweb/3 processes one example entirely after the other. It therefore does not require all examples to be present in memory at the same time and a classification can be extended to accommodate new examples at any time. This makes the program run relatively fast (at least for a Lisp program of that size and complexity) and with comparatively small memory requirements. The disadvantage is that the order in which the examples are presented to Cobweb/3 is significant. Hierarchies may be constructed which are too complicated. This can be tested by doing multiple runs with random selection of the order of examples and comparison of the resulting classifications.
- *Unsupervised Learning*. Cobweb/3 does not need any kind of prior classification or grouping of the input data. It uses the description of the examples itself to determine useful hierarchies. This is similar to statistical clustering.
- *Top-Down Classification*. Cobweb/3 inserts a new example into an existing hierarchy by sorting it down through the present concept hierarchy.

Once Cobweb/3 has been trained on a set of (positive) examples the program and the concept hierarchy generated can be used in three ways.

- *Establish a model* to describe relations among the training examples. The descriptions of intermediate concepts that embrace individual examples are available for model building.
- *Classify new examples* into one of the classes of a concept hierarchy.
- *Predict missing properties* of new examples based on their similarity to existing classes.

Cobweb/3 accepts as input a series of instance descriptions which are ordered sets of attribute-value pairs. Values can be of nominal or numerical type. An example of a protein description with nominal values for the attributes DIAMETER (radius of gyration in Å for a folded conformation), RESIDUES (number of residues determining the size of a protein) and enzymatic ACTIVITY, (expressed as turnover number per second) is the following:

```
diameter   small
residues     few
activity    high
```

The same instance with numerical values reads:

```
diameter    15.0
residues      58
activity   12647
```

Figure 1 shows a small hierarchy of four hypothetical proteins which could have been generated by Cobweb/3. The properties of the simplified protein objects are DIAMETER, RESIDUES and ACTIVITY. The same classification could be expressed numerically. Instead of MEDIUM, for example, there would be a numerical value. For class nodes the mean of all attribute values of the instances covered by this class is recorded together with their standard deviation.

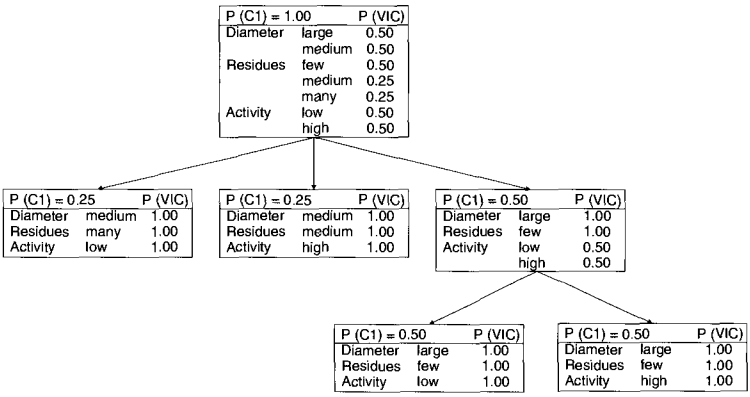


Figure 1: Simple Concept Hierarchy in Cobweb/3.

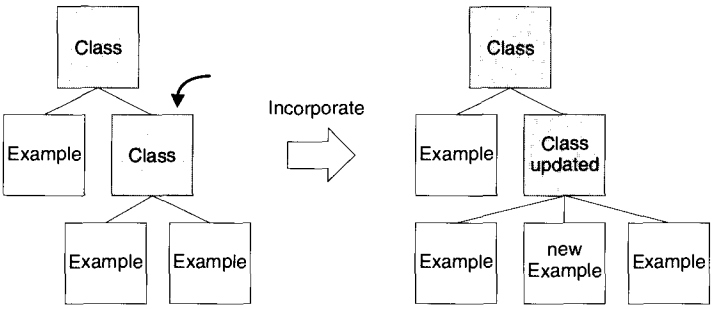


Figure 2: Cobweb/3 Class Incorporate Operator.

Cobweb/3 starts with the most general hierarchy of only one class that contains all examples. Similarity between examples and classes is determined by the *category utility* measure (see below). There are *five operators* that are used to incorporate new examples into an existing hierarchy.

1. *Incorporate to class.* If a new example fits well into an existing class this class is extended to include the new example. The class probabilities are updated to reflect the new attribute-value distribution (Figure 2).
2. *Incorporate into singleton.* If the new example is very similar to another example that is a terminal node (singleton) then a new class is created which covers both examples (Figure 3).
3. *Create a new disjunct.* If an example has very different characteristics from any existing concept at the current level it is placed in a new category by itself (Figure 4).
4. *Merge.* If the classification tree has grown broad the MERGE operator is used. MERGE combines two neighbouring classes and attempts to include an example in the newly combined class (Figure 5).
5. *Split.* If a node becomes too general to be useful for classification the SPLIT operator is used to replace such node by its children (Figure 6).

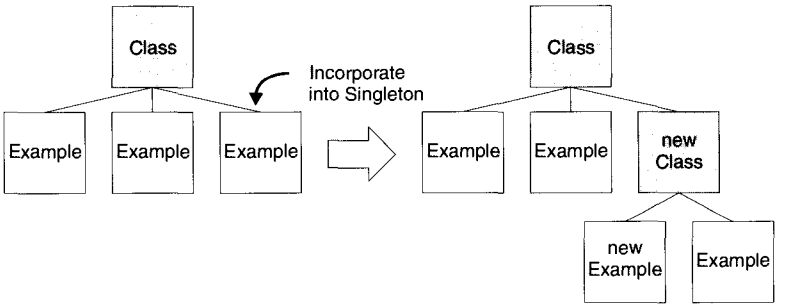


Figure 3: Cobweb/3 Singleton Incorporate Operator.

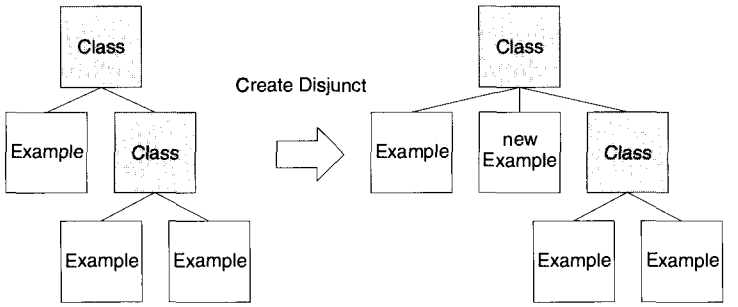


Figure 4: Cobweb/3 Create Disjunct Operator.

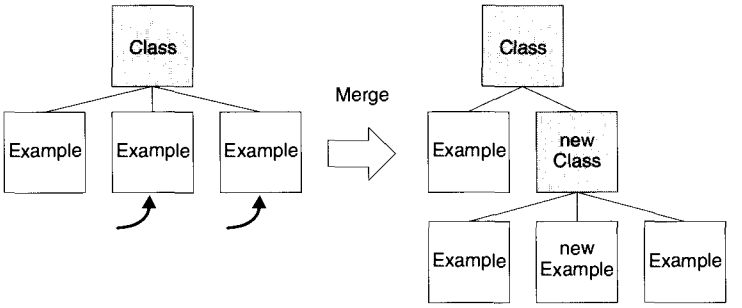


Figure 5: Cobweb/3 Merge Operator.

The application of different operators to incorporate a new example produces several different classification trees. The measure to evaluate the relative quality of different concept hierarchies in Cobweb/3 is *category utility*, u. Category utility gives a high score to classifications that maximise *intra-class similarities* and *inter-class differences*. Thereby, category utility trades off the *predictiveness* $P(C_n \mid A_i = V_{ij})$ of each attribute value (the probability of an instance's membership in a class, given its attribute value) and the *predictability* $P(A_i = V_{ij} \mid C_n)$ of an attribute value (the probability of a value given the membership in a class). *Predictive attribute values* are near-

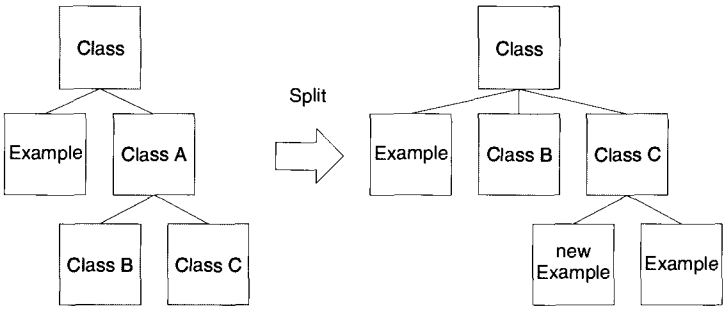


Figure 6: Cobweb/3 Split Operator.

ly unique for a certain class and therefore used to establish the class membership while *predictable attribute values* are easy to guess accurately if only the class membership is known.

The basic strategy behind Cobweb/3 is *probability matching* (in contrast to *probability maximising*). Probability matching means that a value is guessed with the probability $P(A_i{=}V_{ij} \mid C_n)$ and that this guess is correct with the same probability. Probability maximising implies that the value with the highest probability is always guessed. Although the latter approach seems superior at a cursory level probability maximising is not as sensitive to the distribution of values as probability matching.

Category utility u can be expressed as:

$$u = \frac{X - Y}{N}$$

with X as the expected total number of attribute values that can be guessed correctly using N classes, and Y the expected number of attribute values that can be guessed correctly without any classification. Division by N normalises category utility with respect to the number of classes. X can be calculated using Bayes theorem as follows:

$$\text{Bayes Theorem:} \qquad P(A_i = V_{ij})P(C_n|A_i = V_{ij}) = P(C_n)P(A_i = V_{ij}|C_n).$$

$$X = \sum_{n=1}^{N}\sum_{i=1}^{I}\sum_{j=1}^{J} P(A_i = V_{ij})P(C_n|A_i = V_{ij})P(A_i = V_{ij}|C_n) \Leftrightarrow$$

$$X = \sum_{n=1}^{N} P(C_n)\sum_{i=1}^{I}\sum_{j=1}^{J} P(A_i = V_{ij}|C_n)^2.$$

This is the probability sum across N classes with I attributes and J values. The original definition of X tries to maximise and balance intra-class similarity $P(A_i{=}V_{ij} \mid C_n)$ and inter-class difference $P(C_n \mid A_i{=}V_{ij})$ and weights the product of both by the "importance" of individual values $P(A_i{=}V_{ij})$. $P(A_i{=}V_{ij})$ is the *a priori* probability of a particular value V_{ij}. $P(C_n)$ is the probability of the occurrence of a particular class C_n and $P(A_i{=}V_{ij} \mid C_n)$ is the conditional probability of a particular value V_{ij} given the membership of class C_n. $P(C_n \mid A_i{=}V_{ij})$ is the probability of an instance's

membership in class C_n, given its attribute value. Correspondingly, Y is determined by the following equation:

$$Y = \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij})^2 .$$

Substituting X and Y into the equation for u, we obtain:

$$u = \frac{\sum_{n=1}^{N} P(C_n) \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij}|C_n)^2 - \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij})^2}{N} .$$

This version of category utility is used by Cobweb/3 for nominal attribute values. Numerical values must be treated differently because using the above equation would cause numbers like 7.123, 7.234 (etc.) to be considered as different objects without taking their similarity into account. Category utility can be adapted to deal with numerical attributes by representing attribute values as mean and standard deviation. Thus, the innermost summation in the nominal category utility measure can be replaced with the integral of the normal Gaussian distribution:

$$\sum_{j}^{\#\ of\ values} P(A_i = V_{ij})^2 = \int_{-\infty}^{+\infty} \frac{1}{\sigma^2 2\pi} e^{-\left(\frac{x-\mu}{\sigma}\right)^2} dx = \frac{1}{\sigma} \frac{1}{4\sqrt{\pi}} .$$

The transformed category utility for numerical values is then:

$$u = \frac{\sum_{n=1}^{N} P(C_n) \sum_{i=1}^{I} \frac{1}{\sigma_{in}} - \sum_{i=1}^{I} \frac{1}{\sigma_{ip}}}{4N\sqrt{\pi}}$$

with I attributes, N classes, σ_{in} as the standard deviation for attribute i in class n and σ_{ip} as the standard deviation for attribute i in the unclassified set of examples, i.e. before any class information was generated.

A problem arises for $\sigma = 0$ where $1 / \sigma \rightarrow \infty$ in the case of a class with one single instance. Cobweb/3 must then rely on a user supplied *acuity parameter* that defines the minimum detectable difference between two instances. To accommodate missing attributes one can divide the sums by I, the actual number of attributes present.

The Cobweb/3 algorithm is divided into a *learning mode* and a *prediction mode*. In learning mode Cobweb/3 classifies each incoming instance and incorporates it permanently into the hierarchy, changing the hierarchy's structure and thus affecting future classification and prediction. In prediction mode Cobweb/3 also classifies an incoming instance but does not change the class hierarchy nor incorporate the instance. Cobweb/3 merely returns a description of the class found most suitable to accommodate the new instance.

In learning mode Cobweb/3 performs the following steps for each new example:

1. initialise the root node,
2. preview the next level,
3. incorporate an instance into the next level,
4. go back to step 2 until the example would be placed in a class by itself.

Four Objects under Root Node

P (R) = 1.00		P (VIC)
Diameter	large	0.50
	medium	0.50
Residues	few	0.50
	medium	0.25
	many	0.25
Activity	low	0.50
	high	0.50

+

New Object

P (E1) = 1.00		P (VIC)
Diameter	small	1.00
Residues	many	1.00
Activity	medium	1.00

→

Five Objects under Root Node

P (R) = 1.00		P (VIC)
Diameter	large	0.40
	medium	0.40
	small	0.20
Residues	few	0.40
	medium	0.20
	many	0.40
Activity	low	0.40
	high	0.40
	medium	0.20

Figure 7: Incorporation of a New Instance into the Root Node.
At the beginning, there are four objects under the root node. Adding another object shifts the attribute-value probabilities. For example, previously the probability for a LARGE DIAMETER was 2 / 4 = 0.5 and now becomes 2 / 5 = 0.4.

Initialisation is done by calculating the *a priori* probabilities of the attribute values for the first example and storing them in the root node. Initially the probabilities for each datum are 1.0 because they are the only ones seen so far. Then, the next example is incorporated into the root node and the attribute-value probabilities are updated to reflect their new distribution. An example is shown in Figure 7.

During the preview step Cobweb/3 looks ahead one-step to determine the best place to incorporate the instance at the current level of the hierarchy. This involves only the current node (the node where an instance was incorporated at last; initially this is the root node) and its children. Cobweb/3 also considers the application of the MERGE or SPLIT operators on this level. The action which led to the greatest increase in category utility is then made permanent in the incorporation step. Then recursion to the next lower level of the hierarchy follows until the new instance would be placed in a class by itself.

In prediction mode Cobweb/3 descends top-down from the root node analogous to the learning node. The only differences are that the SPLIT and MERGE operators are not used and that the instance is not actually incorporated into the hierarchy. If the simulated incorporation ends at another individual example (singleton) the values of that node are taken for prediction of any missing attribute values. If the new instance create a new disjunct under an existing class the attribute values of that class are used for prediction.

4.1.2. Learning Partitional Classifications: AutoClass III

AutoClass[5] of P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor and D. Freeman is an unsupervised Bayesian classification system for independent data[6]. AutoClass seeks a maximum posterior probability classification. The input to AutoClass is a database of attribute vectors and a class model defined by a parametric class probability function and corresponding parameter priors. Models are constructed from a set of terms for both discrete and real valued attributes. AutoClass attempts to find the set of classes that is maximally probable with respect to the data and model. The output is a set of classes given as instances of the model with specific parameters. AutoClass has facilities for reporting on class membership, influence of an attributes on a class, and on probability weighting of the data over classes. AutoClass runs in the Common Lisp environment of Symbolics and Explorer Lisp machines, and in the Franz and Lucid Lisp implementations on Sun and similar Unix machines[7]. For the Symbolics there is an optional user interface for visualising classification results.

This section describes AutoClass III. The Bayesian approach to pattern recognition uses the probability distribution over attributes of the objects to be classified. This is in contrast to statistical cluster analysis which produces subsets that are interpreted as the definitions of classes. AutoClass does not merely list the objects belonging to one class but generates an explicit model description of each class. AutoClass III does *partitional clustering* as opposed to *hierarchical clustering* as in Cobweb/3. The program has a number of interesting features.

- *The optimal number of classes is determined automatically.* There is a trade-off between a prior probability term that prefers fewer classes and the likelihood of data which drives the system to a classification with more classes. These two effects balance each other and AutoClass III arrives at a single solution with an optimal number of classes.
- *Objects may belong to more than one class.* Probabilistic descriptions of classes allow objects to be, say, 90% in class A and 10% in class B.
- *All attributes are considered significant.* In contrast to manual or visual inspection of a database, Bayesian classification can take many different attributes into account simultaneously.
- *Input can contain both real valued or discrete data.* This is of practical value as one does not have to convert discrete data into numerical format or vice versa. Sometimes these conversions mean a loss of information.

5 P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, D. Freeman, *AutoClass, a Bayesian Classification System*, **Proceedings of the Fifth International Conference on Machine Learning**, Ann Arbor, MI, Morgan Kaufman, 1989.

6 A new version, AutoClass IV, is supposed to also process not completely independent data.

7 Effective from January 1991, AutoClass III will no longer be available by anonymous ftp (Internet address 128.102.16.8) from NASA. The authors are now required to maintain a record of all users and thus will send the code to interested persons by ftp, e-mail, or tape. Foreign requestors are required to obtain formal approval from NASA. To get a copy of AutoClass III send an e-mail to Will Taylor (email address taylor@pluto.arc.nasa.gov).

AutoClass III is based on Bayes theorem. Here the symbol D is used for the observed data and H for a hypothesis to explain the data. For classification purposes H will be the model of class descriptions that is most likely to explain a particular set of instances D. The following probabilities can be discerned:

- $p(D)$ is the *per se* probability of data D. This term does not need to be calculated explicitly as long as we compare alternative hypotheses for one and the same data set.
- $p(D|H)$ is the conditional probability of data D given the hypothesis H. It is also called the *likelihood* of data D. For a specific classification hypothesis calculation of $p(D|H)$ involves a straightforward application of statistics.
- $p(H)$ is the *prior* probability of H, i.e. how likely is the classification model (number and description of classes) H among other, competing hypotheses H_i'. Calculation of $p(H)$ requires a rather complicate mathematical treatment of finite mixtures[8].
- $p(H|D)$ is the probability that the hypothesis H explains the data D. This is also called the *posterior* probability of the hypothesis H given the data D.

For the classification application Bayes theorem states that:

$$p(H|D) = \frac{p(H)\,p(D|H)}{p(D)}.$$

AutoClass III tries to find the best class model $p(H|D)$. Input data is assumed to be in attribute-value vector form. Let the database D contain I objects x_i, each described by K attribute values x_{ik}, $k \in \{1, \ldots, K\}$ in one of $\mathcal{J}$ classes. Attributes may be either real or discrete values but they must be independent in AutoClass III. The class distribution function $p(x_i|x_i \in C_j, \vec{\theta}_j)$ gives the probability distribution of the attributes of an object in class j. The class parameter vector $\vec{\theta}_j$ takes for Gaussian distributed real valued attributes the form:

$$\vec{\theta}_{jk} = \begin{pmatrix} \mu_{jk} \\ \sigma_{jk} \end{pmatrix},$$

where $\vec{\theta}_{jk}$ is the parameter vector describing attribute k in class j by its arithmetic mean and standard deviation. In this particular case, the class distribution function can be expressed as:

$$p\left(x_i|x_i \in C_j, \vec{\theta}_j\right) = \prod_{k=1}^{K} p\left(x_{ik}|x_{ik} \in C_j, \vec{\theta}_{jk}\right).$$

The class distribution of one attribute becomes:

$$p(x_{ik}|x_{ik} \in C_j, \mu_{jk}, \sigma_{jk}) = \frac{1}{\sqrt{2\pi}\,\sigma_{jk}}\, e^{-\frac{(x_{ik}-\mu_{jk})^2}{2\sigma_{jk}^2}}.$$

For discrete attribute values the elements of $\vec{\theta}_{jk}$ are the probabilities for each individual value. The probability of object i coming from a set of classes is the sum of

8 B. S. Everitt, D. J. Hand, *Finite Mixture Distributions*, **Monographs on Applied Probability and Statistics**, Chapman and Hall, London, UK, 1981.

probabilities that it came from each class separately, weighted by the class probabilities:

$$p\left(x_i|\vec{\theta},\vec{\pi},\mathcal{J}\right) = \sum_{i=1}^{\mathcal{J}} \pi_j \, p\left(x_i|x_i \in C_j,\vec{\theta}\right).$$

with π_j as the class probability of class j. The likelihood of measuring the whole data base D equals the product of the probabilities of measuring each individual object:

$$p\left(\vec{x}|\vec{\theta},\vec{\pi},\mathcal{J}\right) = \prod_{i=1}^{I} p\left(x_i|\vec{\theta},\vec{\pi},\mathcal{J}\right).$$

Using Bayes theorem the probability that object i belongs to class j can be calculated as:

$$p\left(x_i \in C_j|x_i,\vec{\theta},\vec{\pi},\mathcal{J}\right) = \frac{\pi_j p\left(x_i|x_i \in C_j,\vec{\theta}_j\right)}{p\left(x_i|\vec{\theta},\vec{\pi},\mathcal{J}\right)}.$$

AutoClass III performs a search over π_i and $\vec{\theta}_j$. The user is asked for a small number that is certainly larger than the expected number of classes. AutoClass III starts with this number of classes which is then successively decremented together with the number of significantly contributing attributes until no further improvement is seen. Typically, the procedure converges after about 20 iterations. The computational cost of a single iteration is of the order $I \cdot \mathcal{J} \cdot K$.

AutoClass III repeatedly creates a random classification and tries to transform it into a high probability classification through local changes until convergence to a local optimum is reached. AutoClass then remembers what it found and starts over again until being told to stop. Each effort is called a "try" and the computed probability is expected to cover the whole volume around the optimum in parameter space rather than just the peak. The standard approach to transform one classification into a better one is:

1. Compute the probabilistic class memberships of all cases based on the class parameters and the implied relative likelihoods.
2. Using the new class members, compute class statistics (mean, standard deviation) and revise the class parameters. Repeat this process until the class statistics stop changing. More sophisticated strategies use additional operators during the converge cycle, such as merging and splitting classes.

One can halt AutoClass III after a certain amount of time (expressed in seconds) or by typing a "q" during program execution. Deciding when to stop is a judgement call and is therefore up to the user. Since the search includes a random component, there is always the chance that if one lets AutoClass keep going it will find a better solution. So one needs to trade off between expected additional improvement and the minimum time in which that improvement is likely to happen. Search status reports are printed for each improved classification. The reports include an estimation of the next improvement and by when it is expected to occur.

One indication that one should stop AutoClass is when most of the classifications found are duplicates of previous ones (flagged by "dup" as they are found). This should only happen for very small sets of data or when fixing a very small number of classes, like two. Each new try begins with a certain number of classes and ends with a slightly smaller number, as some classes may drop out. Compute time is of the order (number of examples) · (number of attributes) · (number of classes) · (number of tries) · (number of converge cycles per try). The major uncertainties in the estimation of run-time are the number of basic back and forth cycles until each try converges, and, of course, the number of tries which was found to vary between 10 to 100.

The influence value for discrete valued attributes is calculated as:

$$I_{jk} = \sum_l P_{jkl} \ln \frac{P_{jkl}}{P_{kl}}.$$

The influence value for real valued attributes is given by:

$$I_{jk} = \ln \frac{s_k}{s_{jk}} + \frac{1}{2} s_k^2 \left(s_{jk}^2 - s_k^2 + (m_{jk} - m_k)^2 \right),$$

where j is the class index, k the attribute index, l the discrete attribute value index, I_{jk} the influence value of attribute k in class j, P_{jkl} the probability of value l of attribute k for class j, P_{kl} the global probability of value l of attribute k, s_k the global standard deviation of attribute k, m_k the global mean of attribute k, s_{jk} the standard deviation of attribute k of class j, and m_{jk} the mean of attribute k in class j. Class strength is a measure of information density given by:

$$\frac{classL_t}{classW_t}$$

where t is the class number. $classL$ is the logarithmic approximate likelihood of the class statistics with respect to the class hypothesis and parameters. $classW$ is the sum of attribute weights for this class.

4.2. Applications

The following sections illustrate the application of Cobweb/3, AutoClass III and ID3 (section 2.1.3) to the analysis of super-secondary structures and to signal pattern prediction in protein and DNA sequences.

4.2.1. Inductive Analysis of Protein Super-Secondary Structure

A new method for investigating protein super-secondary structure was developed[9] in the Inductive Protein Structure Analysis project (IPSA) by S. Schulze-

9 S. Schulze-Kremer, R. King, *IPSA – Inductive Protein Structure Analysis*, **Protein Engineering**, vol 5, no 5, pp. 377-390, 1992.

Kremer[10,11,12]. IPSA includes the creation of a new database which is designed specifically for the analysis of protein structure by statistics and machine learning. The Protein Representation Language (PRL) database was created using a suite of programs to convert raw atomic coordinates from the Brookhaven protein database (PDB) into higher-level chemically meaningful terms. These terms include explicit symbolic representations of geometrical, topological and chemo-physical information about secondary structures and relationships between secondary structures. The IPSA methodology was applied to the PRL database to discover unrecognised types of protein super-secondary structure. The method consists of:

1. The use of PRL information to produce a new database of secondary structures which associate together (potential super-secondary structures).
2. The use of various clustering techniques to produce a consensus clustering of these putative examples of super-secondary structure.
3. The classified super-secondary structures are finally examined to uncover biological significant features.

This method was applied to find simple super-secondary structures consisting of pairs of α-helices and pairs of one helix and one β-strand. For the α-helix pairs four well-defined super-secondary structures were found, for the helix/strand pairs 12 classes can be discerned. Tests were carried out using homologous pairs and conformational fits which confirm the clustering.

Automatic inductive inference methods are of special interest to crystallographers because structural databases are becoming so large that it is impossible to analyse the abundance of protein conformations by individual, manual inspection. Computational methods for data analysis can also be more systematic and objective than human perception. The PRL database was formed to make automatic inductive analysis of the Brookhaven protein database easier. This was necessary because PDB was designed with human crystallographers in mind and consists basically of the 3-dimensional coordinates of atoms. It does not contain an explicit symbolic representation of rigorously defined topological or chemo-physical properties of proteins proven to be useful to protein engineers. Humans can perceive patterns in such data (secondary, super-secondary structures) when it is projected on a computer screen using a molecular modelling program but such visual-spatial reasoning is difficult by computer. For a wider application of inductive methods, a richer symbolic description of proteins is needed.

The IPSA method aims to automatically derive regularities in protein structure from looking at a set of known structures. As this is a form of inductive reasoning the approach was termed *inductive protein structure analysis*. IPSA consists of the following four steps:

10 S. Schulze-Kremer, *IPSA – Induktive Proteinstruktureanalyse,* **diploma thesis**, Free University Berlin, Department of Chemistry, 1990.

11 S. Schulze-Kremer, *IPSA – Induktive Proteinstruktureanalyse,* **PhD thesis**, Free University Berlin, Department of Chemistry, 1992.

12 S. Schulze-Kremer, *IPSA – Inductive Protein Structure Analysis* in **Symbols versus Neurons? – Advanced Applications in Artificial Intelligence**, (Eds. J. Stender, T. Addis), IOS Press, Amsterdam, pp. 118-151, 1990.

1. Formation of a database.
   a) Define a collection of relevant attributes for describing secondary structures and the relation between pairs of secondary structure. These attributes include geometrical, topological and chemo-physical properties.
   b) Write a series of programs to calculate those attributes for a set of secondary structures. The input to these programs consists of 3-dimensional atomic coordinates of proteins from PDB; the output is the PRL database.
   c) From the PRL database form a database of elements of secondary structure which join together (potential examples of super-secondary structures) for use in clustering. This produces a multi-dimensional attribute description of the aggregated elements. In IPSA, pairs of two α-helices and pairs of one helix and one β-strand are considered.
2. Clustering.
   a) Cluster the data on aggregated elements using a number of clustering methods. Here, clustering is performed by the programs Cobweb/3, AutoClass III and Clustan (statistical cluster analysis).
   b) Combine the results of the different clustering methods to form a consensus clustering. The clusters (classes) of aggregated pairs of secondary structure are considered to be simple super-secondary structures.
3. Analysis.
   a) For each super-secondary structure, use Monte-Carlo statistics to find which attributes differ significantly between classes of super-secondary structure.
   b) Examine the conformations of the discovered super-secondary structures. The r.m.s. fits of conformations within each class are compared with the conformations of super-secondary structures from other classes.
   c) Study the super-secondary structures by molecular modelling techniques to examine their structural similarity and biological significance.
4. Prediction.
   a) Use existing methods for correlating primary structure to secondary structure to detect amino acid residue patterns in super-secondary structures.
   b) Use the patterns learned in step 4.a to predict super-secondary structures.
   c) Pack predicted super-secondary structures to build the overall conformation of a whole protein. This approach reduces the size of the conformation space for *ab initio* predictions.

4.2.1.1. Properties of Secondary Structures

A suite of programs was written to transform information about atomic coordinates from PDB into chemically meaningful terms in the PRL database. The properties calculated by the programs are classified into three groups: *basics*, *attributes of secondary structure* and *attributes of pairs of secondary structure*. The attributes in turn can be classified by their type: *geometrical*, *topological* or *chemo-physical*. The output of the programs are PRL database records in several ASCII files. Each file contains

information on a particular secondary structure property of a protein. Information in the PRL database is implemented as facts in Prolog syntax. This form of information storage has the advantage that the logical reasoning power of Prolog can be used on the database, e.g. to carry out complicated searches or to perform a simulation[13].

Basics

Primary Structure. The primary structure designations were taken directly from PDB.

Secondary Structure. The secondary structure designations used are found using the Kabsch-Sander program DSSP[14,15]. This uses the atomic coordinates from PDB. The Kabsch-Sander algorithm gives seven possible types of secondary structure: H = α-helix, G = 3_{10}-helix, I = π-helix, B = isolated β-bridge, E = extended strand, S = bend and T = H-bond turn. As well as this type of secondary structure, a three state designation was used: H, G and I give H = helix; E and B give B = β-strand; S and T give T = turn. Regions not designated to a secondary structure type are considered unstructured. In this work the standard secondary structure alphabet as defined by the DSSP program was used. One could, however, have chosen a completely different definition of secondary structures for IPSA.

Conformation. The 3-dimensional coordinates of the Cα atoms of the residues are used. This information did not go directly into the learning step. Only derived information was used for clustering.

Attributes of Secondary Structures

For each relevant secondary structure certain attributes were calculated and recorded.

Geometrical attributes of secondary structure.

Distance to the centre of gravity of protein. This is the distance from the centre of gravity of each secondary structure to the overall centre of gravity of the protein. The centre of gravity of a set of residues is defined as the vector sum of their Cα coordinate positions divided by the number of residues. As an approximation, all residues are assumed to have the same mass.

Bending. The amount of bending is defined as the angle between two connecting lines, one extending from the start to the middle of a secondary structure and the other from the middle to the end. For β-strands and turns, the start, middle and end are the appropriate Cα atoms (if there is no middle Cα, then it is the next in sequence). For helices, the start, middle and end are the centres of gravity of three consecutive atoms adjusted for each helix type.

---

13 S. Schulze-Kremer, *CSProlog – Parallel Programming with Transputers. Application: MolSIM, A Program for Simulation of Concurrent Molecularbiological Processes*, in **Tools and Techniques for Transputer Applications**, (Ed. S. Turner), IOS Press, Amsterdam, pp. 97- 110, 1990.

14 W. Kabsch, C. Sander, *Dictionary of protein structure: Pattern recognition of hydrogen-bonded and geometrical features*, **Biopolymers**, vol 22, pp. 2577-2637, 1983.

15 World Wide Web Server of the EMBL at http://www.embl-heidelberg.de. DSSP files are available in file://ftp.embl-heidelberg.de/pub/databases/protein_extras/dssp.

Twisting of strands. Twisting of a β-strand is defined as the angle between the plane formed by the first three Cα atoms and the plane formed by the last three Cα atoms.

Chemo-physical attributes of secondary structure.

Internal hydrogen bonding. The number of internal H-bonds with energy less than -0.5 Kcal mol^{-1} is counted and the total energy of these bonds found. This is especially relevant for helices and H-bond turns. The information about H-bonds was obtained using the DSSP program.

Relative solvent accessibility. Solvent accessibility of a secondary structure is recorded as a percentage of the total accessibility. This information was obtained using the DSSP program.

Internal helix salt bridges. For each helix, the number of intra-helical salt bridges is found along with the orientation of the bridge. A salt bridge is defined to be positively oriented when there is a negatively charged residue at position i and a positively charged residue at position i+3 or i+4. Conversely, a salt bridge is defined to be negatively oriented when there is a positively charged residue at position i and a negatively charged residue at position $i + 3$ or $i + 4$. The negatively charged residues are defined to be Asp and Glu and the positively charged ones His, Lys and Arg.

Helix hydrophobic moment. To calculate the hydrophobic moment of a helix it is first necessary to calculate the axis of the helix. This is done by linear regression of the corrected centres of gravity of all triples of three consecutive Cα atoms. The hydrophobic moment is then calculated as the sum of the hydrophobic vectors of all Cα atoms of the helix. The hydrophobic vector of a Cα atom is defined to have the length corresponding to the hydrophobicity of its residue on the consensus scale[16], to be perpendicular to the helix axis and to point from the helix axis to the Cα atom. This method of calculating the helix hydrophobic moment is more accurate than that of Eisenberg[17] (1984) as it does not assume a perfectly regular helix. However, it can be used only when atomic coordinates of the Cα atoms are known.

Strand hydrophobic moment. Strand hydrophobic moment is calculated by assuming that each alternate residue lies on the opposite side of a plane. The two sums of the hydrophobicity values on either side of the plane are calculated, and the difference of both is defined to be the hydrophobic moment of the strand.

Helix polarity. Helix polarity is calculated in an analogous manner to the hydrophobic moment of helices. Instead of using a hydrophobicity scale, a value of 1 is given to a charged residue and 0 to all other residues. Helix polarity is calculated twice, once for negatively charged residues and once for positively charged residues.

Strand polarity. Strand polarity is calculated in an analogous manner to the hydrophobic moment of strands. Instead of using a hydrophobicity scale, a value of 1 is given to a charged residue and 0 to all other residues. Strand polarity is calculated twice, once for negatively charged residues and once for positively charged residues.

16 D. Eisenberg, R. M. Weiss, T. C. Terwillinger, W. Wilcox, **Farady Symp. Chem. Soc.**, 17, pp. 109-120, 1982.

17 D. Eisenberg, **Ann. Rev. Biochem.**, vol 53, pp. 595-623, 1984.

Relations between Secondary Structure

For each pair of secondary structures, the following properties were calculated and recorded:

Geometrical attributes of pairs of secondary structure.

Angle between secondary structures. The angle between two secondary structures is defined to be the angle between their axes projected on a common plane. It is calculated as the arcus cosine of the scalar product of the unit vectors of the axes. The values range from 0° to 180°. The axis of a helix is defined as above, for β-strands and turns their axes are defined to go through their first and last Cα atoms.

Distance between centres. The centre of a helix is defined to be its centre of gravity, for β-strand and turn secondary structures, the centre is defined to be the middle Cα (if there is no middle Cα, then it is the next in sequence).

Closest distance. The closest distance between two secondary structures is defined to be the minimum distance between a Cα atom in one secondary structure and a Cα atom in the other.

Distance between axes. The axis of a helix is defined as above, for β-strands and turns their axes are defined to go through their first and last Cα atoms. This attribute is defined to be the closest distance between the two axes. Note: axes are taken to be infinite lines.

Topological attributes of pairs of secondary structure.

Strand partner and orientation. Each pair of β-strands (and their orientation, parallel or antiparallel) which are connected by H or S bonds is recorded. This information was found using the program DSSP.

Disulphide bonds. Each pair of secondary structures connected by a disulphide bond is recorded. The information about these bonds was obtained using the DSSP program.

External hydrogen bonding. The number of H-bonds with energies less than -0.5 Kcal mol^{-1} was counted and the total energy of these bonds added; this property is especially relevant for pairs of β-strands. The information about H-bonds was obtained using the DSSP program.

Distance in residues. The distance in residues between a pair of secondary structures is the number of residues between them along the polypeptide chain. This measure is not defined if the two secondary structures reside on separate chains.

Distance in secondary structures of the same type. This is defined to be, for each pair of secondary structures of one type (i.e. pairs of helices, β-strands or turns), the number of secondary structures of the same type separating them along the polypeptide chain. For example, for a pair of helices the number of helices in between them along the chain is found. This measure is not defined if the two secondary structures reside on separate chains.

Distance in secondary structures. The distance in secondary structures is the number of other secondary structures between them along the polypeptide chain. This measure is not defined if the two secondary structures reside on separate chains.

4.2.1.2. PRL Database

The following 129 proteins were selected from the Brookhaven database (release January 1990) for use in PRL: 1ABP, 1ACX, 1BDS, 1BP2, 1CBH, 1CC5, 1CCR, 1CMS, 1CPV, 1CRN, 1CTF, 1CTX, 1CY3, 1ECD, 1ETU, 1GCR, 1GOX, 1GP1, 1GPD, 1HIP, 1HMQ, 1HOE, 1INS, 1LH3, 1LZ1, 1MBD, 1MLT, 1NXB, 1PAZ, 1PCY, 1PHH, 1PPT, 1PRC, 1PSG, 1PYP, 1RHD, 1RNS, 1SBT, 1SN3, 1TEC, 1TIM, 1TNF, 1TON, 1TPP, 1UBQ, 1WSY, 2AAT, 2ABX, 2ACT, 2ALP, 2AZA, 2B5C, 2CAB, 2CCY, 2CDV, 2CI2, 2CNA, 2CPP, 2CRO, 2CTS, 2CYP, 2FB4, 2GN5, 2HHB, 2LBP, 2LH4, 2LHB, 2LIV, 2LZM, 2MHR, 2MT2, 2OVO, 2PAB, 2PFK, 2PKA, 2PRK, 2RNT, 2RSP, 2SGA, 2SNS, 2SOD, 2SSI, 2STV, 2TAA, 2TBV, 2TS1, 2WRP, 351C, 3ADK, 3BCL, 3C2C, 3CLN, 3DFR, 3EBX, 3EST, 3FXC, 3GAP, 3GRS, 3ICB, 3LZM, 3PGK, 3PGM, 3RP2, 3SGB, 3TLN, 3WGA, 3XIA, 451C, 4APE, 4FD1, 4FXN, 4HHB, 4MDH, 4RXN, 4SBV, 5ATC, 5CHA, 5CPA, 5HIR, 5PTI, 5TNC, 6API, 6LDH, 7CAT, 7RSA, 8ADH, 8CAT, 8DFR, 9PAP. The proteins were selected to have a good resolution and to be broadly non-homologous[18]. It should be noted that a great deal of effort was put in to selecting "clean" data from PDB; e.g. to exclude data with missing coordinates or breaks in the chain. Several homologous proteins were deliberately left in this set (e.g. 2LH4, 2LHB and 4HHB are all haemoglobins). This was done to provide an internal check for the clustering programs. For applications where low homology between proteins is needed it is possible to use a sub-set of the database[19].

There are the 16 types of entries in the PRL database, termed P1 – P16.

P1 – Secondary Structure Listing

The parameters are:

1. Protein ID as in PDB. A leading number in PDB is put at the end and lowercase letters are used to implement names of proteins as Prolog constants.
2. Chain ID.
3. Running number of secondary structure. This is a unique identifier for each secondary structure within a protein.
4. Secondary structure number, referring to helices (h), strands (b) or turns (t).
5. Residue number where secondary structure starts. This is in the PRL rational numbering scheme, not in the PDB numbering (see below). The first residue in a chain is always numbered 1.

18 An updated version of a selection of non-homologous PDB protein files can be obtained from the EMBL World Wide Web Server at http://www.embl-heidelberg.de and there in file://ftp.embl-heidelberg.de/pub/databases/protein_extras/pdb_select. Families of homologous proteins are obtainable from file://ftp.embl-heidelberg.de/pub/databases/protein_extras/fssp.

19 The PRL database is distributed at nominal cost by Brookhaven National Laboratories as an unsupported part of their Protein Database. Address: Brookaven National Laboratory, Chemistry Department, Building 555, PO Box 5000, Upton, NY 11973-5000, USA. Email pdb@bnl.gov, Internet ftp server at pdb.pdb.bnl.gov or 130.199.144.1. World Wide Web Server at http://www.pdb.bnl.gov.

6. Residue number where secondary structure ends. This is in the PRL rational numbering scheme, not in the PDB numbering (see below). The first residue in a chain is always numbered 1.
7. Length of secondary structure in amino acid residues. Only segments of at least two residues are considered.
8. Condensed secondary structure type. H, G, I (from Kabsch-Sander) give h (helix); E, B give b (β-strand); S, T give t (turn).
9. Relative accessibility of secondary structure.
10. Original Kabsch-Sander secondary structure type.
11. Residue number where secondary structure starts. This is in the PDB numbering scheme (for reference to PDB). This numbering is often based on a homologous protein and may therefore start with a negative number, a number greater than 1, or miss numbers in between.
12. PDB insertion code at the start of a secondary structure. This indicates insertion of a residue in the PDB numbering scheme. If no letter is given in PDB the letter "z" is used.
13. Residue number where secondary structure ends. This is in the PDB numbering scheme (for reference to PDB; see also item 11).
14. PDB insertion code at the end of a secondary structure (see also item 12).

Examples:

```
sec_struc (atc5, a,  1,  1,   3,  6, 4, t, 0.57, t,   3, z,   6, z).
sec_struc (atc5, a,  2,  2,  13, 14, 2, t, 0.32, t,  13, z,  14, z).
```

P2 – Cα Coordinates and Amino Acid Residues

The parameters are:

1. PRL residue number (see P1).
2. Cα atom x coordinate.
3. Cα atom y coordinate.
4. Cα atom z coordinate.
5. Amino acid type (three letter code).
6. Chain ID.
7. Protein ID (permutated PDB ID; see P1).
8. PDB residue number (see P1).
9. PDB residue insertion code (see P1).

Examples:

```
residue (1,  38.92,  41.52,  -5.16, ala, a, atc5, 1, z).
residue (2,  36.38,  43.97,  -6.71, asn, a, atc5, 2, z).
```

P3 – Strand Neighbours and Orientation

The parameters are:

1. Secondary structure number of first strand.
2. Secondary structure number of second strand.

3. Orientation of first strand versus second strand (+ is parallel, – anti-parallel).
4. Ladder ID (two strands form one ladder).
5. Sheet ID (more than two strands can form a sheet).
6. Chain ID.

Examples:

```
strand_partner (1,  2, +, b, a, a).
strand_partner (1,  3, +, c, a, a).
```

P4 – Hydrogen Bonds between Secondary Structures

The parameters are:

1. Condensed secondary structure type of first secondary structure.
2. Secondary structure number of first secondary structure.
3. Condensed secondary structure type of second secondary structure.
4. Secondary structure number of second secondary structure.
5. Number of hydrogen bonds with energy less than -0.5 Kcal mol^{-1}.
6. Total energy of hydrogen bonds (only bonds with less than -0.5 Kcal mol^{-1} are counted).

Examples:

```
h_bond (h,  1, h,  1, 14,  -28.3).
h_bond (t,  3, t,  6,  2,   -2.6).
```

P5 – Disulphide Bonds between Secondary Structures

The parameters are:

1. Condensed secondary structure type of first secondary structure.
2. Secondary structure number of first secondary structure.
3. Condensed secondary structure type of second secondary structure.
4. Secondary structure number of second secondary structure.

Examples:

```
s_bond (t, 25, b, 20).
s_bond (b, 20, t, 25).
```

P6 – Helix Salt Bridges

The parameters are:

1. Number of helix.
2. Number of intra-helical salt bridges.
3. Orientation of salt bridges. A salt bridge is defined to be positively oriented when there is a negatively charged residue at position i and a positively charged residue at position i+3 or i+4. Conversely, a salt bridge is defined to be negatively oriented when there is a positively charged residue at position i and a negatively charged residue at position i+3 or i+4. The negatively charged residues

are defined to be Asp and Glu and the positively charged ones His, Lys and Arg.

Examples:

```
salt_intra (1, 0, +).
salt_intra (1, 0, -).
```

P7 – Helix Hydrophobic Moment

The parameters are:

1. Number of helix.
2. Hydrophobic moment of helix. To calculate the hydrophobic moment of a helix it is first necessary to calculate the helix axis. This is done by linear regression over the corrected centres of gravity of all triples of three consecutive Cα atoms. The hydrophobic moment is then calculated as the sum of the hydrophobic vectors of all Cα atoms of the helix. The hydrophobic vector of a Cα atom is defined to have the length corresponding to the hydrophobicity of its residue on the consensus scale, to be perpendicular to the helix axis and to point from the helix axis to the Cα atom.

Examples:

```
helical_hydroph_moment (1, 4.77).
helical_hydroph_moment (2, 2.65).
```

P8 – Strand Hydrophobic Moment

The parameters are:

1. Number of strand.
2. Hydrophobic moment of strand. Strand hydrophobic moment was calculated by assuming that each alternate residue lies on the opposite side of a plane. The sums of the hydrophobicity values on either side of the plane are calculated and the difference is the hydrophobic moment of the strand.

Examples:

```
strand_hydroph_moment (1, 0.04).
strand_hydroph_moment (2, 0.07).
```

P9 – Helix Polarity

The parameters are:

1. Number of helix.
2. Polarity value of helix. Helix polarity was calculated in an analogous manner to the hydrophobic moment of helices. Instead of using a hydrophobicity scale, a value of 1 is given to a charged residue and 0 to all other residues.
3. Polarity type of helix. Helix polarity is calculated twice, once for positive residues (+) and once for negative residues (-).

Examples:

```
helix_polarity (1, 3.07, +).
helix_polarity (1, 2.65, -).
```

P10 – Strand Polarity

The parameters are:

1. Number of strand.
2. Polarity value of strand. Strand polarity was calculated in an analogous manner to the hydrophobic moment of strands. Instead of using a hydrophobicity scale a value of 1 is given to a charged residue and 0 to all other residues.
3. Polarity type of strand. Strand polarity is calculated twice, once for positive residues (+) and once for negative residues (-).

Examples:

```
strand_polarity (1, 0.05, +).
strand_polarity (1, 0.40, -).
```

P11 – Distances between Secondary Structures

There are six different predicates: HELICAL_SEGMENT_DISTANCE, STRAND_HELIX_DISTANCE, STRAND_SEGMENT_DISTANCE, TURN_HELIX_DISTANCE, TURN_SEGMENT_DISTANCE, TURN_STRAND_DISTANCE. They hold information about several distance measures between secondary structures.

The parameters are:

1. Secondary structure number of first secondary structure in the name of the predicate.
2. Chain ID of first secondary structure in the name of the predicate.
3. Secondary structure number of second secondary structure in the name of the predicate.
4. Chain ID of second secondary structure in the name of the predicate.
5. Distance between the centres of secondary structures in Å (centre of gravity of the middle three Cα atoms for helix and middle Cα atom for strand and turn).
6. Closest distance between the secondary structures in Å. Each distance between a Cα atom in one secondary structure and a Cα atom in the other is calculated and the smallest one selected.
7. Count how many secondary structures of the same type are between those two referred to in this predicate (only for HELICAL_SEGMENT_DISTANCE, STRAND_SEGMENT_DISTANCE, TURN_SEGMENT_DISTANCE; zero otherwise).
8. Count how many secondary structures of any type are between those two referred to in this predicate (for all six distance predicates).
9. Count how many amino acid residues are in between the two secondary structures referred to in this predicate.

10. Closest distance in Å between the (infinite!) axis through secondary structures (helix axis goes through the middle of a helix; strand and turn axes go through first and last Cα atom).

Examples:

```
turn_segment_distance   (1, a, 2, a, 14.64, 11.51, 1, 1,  7,  6.62).
turn_helix_distance     (1, a, 1, a, 11.81,  7.39, 0, 2, 11,  6.24).
turn_strand_distance    (1, a, 1, a, 19.10, 13.92, 0, 5, 37,  6.73).
strand_helix_distance   (1, a, 1, a, 23.29, 19.33, 0, 3, 11, 20.77).
helical_segment_distance(1, a, 2, a, 18.19,  9.71, 1, 5, 21, 11.88).
strand_segment_distance (1, a, 2, a,  5.31,  4.51, 1, 4, 21,  0.90).
```

P12 – Angles between Secondary Structures

There are six different predicates: HELICAL_SEGMENT_ANGLE, STRAND_HELIX_ANGLE, STRAND_SEGMENT_ANGLE, TURN_HELIX_ANGLE, TURN_STRAND_ANGLE, TURN_TURN_ANGLE. They hold information about the angle between secondary structures.

The parameters are:

1. Secondary structure number of first secondary structure in the name of the predicate.
2. Chain ID of first secondary structure in the name of the predicate.
3. Secondary structure number of second secondary structure in the name of the predicate.
4. Chain ID of second secondary structure in the name of the predicate.
5. Angle between the axis through secondary structures in radian.

Examples:

```
helical_segment_angle (2, a,  5, b, 1.23).
strand_helix_angle    (3, a,  1, a, 1.80).
strand_segment_angle  (1, a,  3, a, 0.20).
turn_helix_angle      (1, a,  1, a, 1.36).
turn_strand_angle     (1, a,  1, a, 1.43).
turn_turn_angle       (1, a,  2, a, 1.27).
```

P13 – Bending of Secondary Structures

There are three predicates: HELICAL_SEGMENT_BEND, STRAND_SEGMENT_BEND, TURN_SEGMENT_BEND. They hold information on how much a secondary structure is bent. Bending γ is defined as:

$$\gamma = \frac{180^\circ - angle(\text{first C}\alpha, \text{middle C}\alpha, \text{last C}\alpha)}{2}$$

Instead of a single Cα atom for β-strands and turns, the centre of gravity of three consecutive Cα atoms is taken for helices.

The parameters are:

1. Secondary structure number of the secondary structure.
2. Bending angle in degrees.

Examples:

```
helical_segment_bend (1,  2.24).
strand_segment_bend  (2, 20.69).
turn_segment_bend    (8, 37.26).
```

P14 – Twisting of Strands

The twist of a strand is defined as the angle between the planes defined by the first and last three Cα atoms, respectively.

The parameters are:

1. Secondary structure number of the strand.
2. Twist of the strand in radian.

Examples:

```
strand_segment_twist (1, 0.85).
strand_segment_twist (2, 0.23).
```

P15 – Relative Solvent Accessibility of Secondary Structures

The parameters are:

1. Secondary structure number of the secondary structure.
2. Condensed type of secondary structure.
3. Relative solvent accessibility of that secondary structure, calculated as percent of total accessibility.

Examples:

```
solvent_rel_accessibility (1, t, 0.57).
solvent_rel_accessibility (1, h, 1.61).
```

P16 – Distance between Secondary Structure and Centre of Gravity

There are three predicates: HELICAL_GRAV_DISTANCE, STRAND_GRAV_DISTANCE, TURN_GRAV_DISTANCE. They give the distance from the centre of gravity of a secondary structure to the overall centre of gravity of a protein. Centres of gravity are calculated only from Cα atoms.

The parameters are:

1. Secondary structure number of the secondary structure.
2. Distance to protein gravity centre in Å.

Examples:

```
helical_grav_distance (1, 44.84).
strand_grav_distance  (1, 29.70).
turn_grav_distance    (6, 43.81).
```

1. Structure1 length in residues
2. Structure1 solvent accessibility
3. Structure1 bending
4. Structure1 distance to gravity centre
5. Structure1 positive polarity
6. Structure1 negative polarity
7. Structure1 twisting
8. Structure1 hydrophobicity
9. Structure1 positive salt bridges
10. Structure1 negative salt bridges
11. Structure2 length in residues
12. Structure2 solvent accessibility
13. Structure2 bending
14. Structure2 distance to gravity centre
15. Structure2 positive polarity
16. Structure2 negative polarity
17. Structure2 twisting
18. Structure2 hydrophobicity
19. Structure2 positive salt bridges
20. Structure2 negative salt bridges
21. Angle between structures
22. Orientation of structures
23. Number of H-bonds between structures
24. Strength of H-bonds between structures
25. Number of S-bonds between structures
26. Turn connection between structures
27. Distance between centres of structures
28. Distance between nearest points of structures
29. Distance in same secondary structures
30. Distance in all kinds of secondary structures
31. Distance in residues between structures
32. Distances between axes of structures.

Table 1: 32 Properties for Clustering.
"Structure1" of an α-helix pair is the helix closer in sequence to the N-terminus.

4.2.1.3. α-Helix/α-Helix Pairs

Pairs of α-helices (potential super-secondary structures) were selected from the PRL database. α-Helices are only helices with type H as determined by the DSSP program. A pair of α-helices was selected if they were considered to be associated, i.e. if their closest distance was less than 3 Å or the distance between their centres was less than 10 Å or if they were connected by hydrogen or S-S bonds. Each α-helix pair has 32 features (Table 1). This definition of aggregation produced 145 examples of pairs of α-helices from the selected 129 proteins of PDB (Table 2).

The following is the parameter control file for the application of Cobweb/3 to the classification of α-helix pairs. Example 1 shows the result of Cobweb/3.

```
(load "sysdcl.cl")
(use-package :labyrinth)
(load-it)
(run "data/hh.dat" :acuity 0.1 :outfile t :print-each t :att-names
      '(a1_length a1_access a1_bending a1_gravity_distance a1_polar_pos
      a1_polar_neg a1_twisting a1_hydrophobic a1_salt_bridge_pos
      a1_salt_bridge_neg a2_length a2_access a2_bending
      a2_gravity_distance a2_polar_pos a2_polar_neg a2_twisting
      a2_hydrophobic a2_salt_bridge_pos a2_salt_bridge_neg r_angle
      r_orientation r_h_bond r_h_bond_str r_s_bond r_turn_con_both
      r_dist_centre r_dist_near r_dist_same_ss r_dist_all_ss r_dist_aa
      r_dist_lines))
(progn (rec-complete) (print-tree))
(write-tree-to-file "data/hh.tree")
(quit)
```

```
Node{145 144 143 142 141 140 139 138 137 138 135 134
    133 132 131 130 129 128 127 128 125 124 123 122
    121 120 119 118 117 118 115 114 113 112 111 110
    109 108 107 108 105 104 103 102 101 100 99 98 97
    98 95 94 93 92 91 90 89 88 87 88 85 84 83 82 81 80
    79 78 77 78 75 74 73 72 71 70 89 88 87 88 65 84 63
    82 81 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46
    45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29
    28 27 28 25 24 23 22 21 20 19 18 17 18 15 14 13 12
    11 10 9 8 7 8 5 4 3 2 1}
Node{145 144 143 142 139 137 128 127 125 124 121 120
    119 118 117 113 112 109 108 107 106 105 100 99 98 98
```

No.	PDB ID	Chain	Start1	End1	Start2	End2	Class	No.	PDB ID	Chain	Start1	End1	Start2	End2	Class
1	5ATC	a	17	32	292	304	C	74	4HHB	c	76	79	81	89	D
2	5ATC	a	135	149	285	290	C	75	4HHB	c	96	112	119	137	B
3	5ATC	a	285	290	292	304		76	4HHB	d	20	34	58	75	A
4	3ADK	a	99	108	146	156		77	4HHB	d	20	34	101	118	A
5	3ADK	a	146	156	158	164		78	2LHB	a	13	28	30	44	
6	8ADH	a	175	184	202	213		79	2LHB	a	30	44	61	65	
7	2TAA	a	182	195	215	222		80	1PHH	a	12	24	299	317	C
8	1ABP	a	43	55	73	81		81	1PHH	a	88	91	375	386	C
9	1ABP	a	111	128	149	162	A	82	6LDH	a	30	40	247	261	C
10	1ABP	a	180	192	210	217		83	1LH3	a	5	19	22	36	B
11	2AAT	a	62	65	67	70	D	84	1LH3	a	22	36	107	122	
12	2AAT	a	62	65	301	310	C	85	2LH4	a	5	19	22	36	B
13	1CPV	a	8	17	26	32	B	86	2LH4	a	22	36	107	122	
14	1CPV	a	40	50	99	107	A	87	2LH4	a	58	81	128	152	A
15	1CPV	a	60	64	79	88		88	2LBP	a	16	35	275	287	C
16	5CPA	a	216	231	286	306		89	2LBP	a	55	66	79	91	
17	7CAT	a	192	198	440	448	C	90	2LBP	a	203	215	232	237	
18	7CAT	a	453	466	486	499	A	91	2LIV	a	55	67	79	91	
19	8CAT	a	453	466	486	499	A	92	2LZM	a	60	79	84	90	B
20	1CMS	a	50	53	112	116	D	93	2LZM	a	115	123	126	134	
21	1CMS	a	272	275	300	303	D	94	2LZM	a	143	155	159	163	B
22	2CTS	a	89	98	393	414	C	95	3LZM	a	115	122	126	133	B
23	2CTS	a	137	147	258	269		96	1LZ1	a	25	36	110	115	
24	2CTS	a	243	252	258	269	B	97	1CTF	a	65	76	80	88	
25	1CRN	a	7	17	23	30	B	98	4MDH	a	215	219	221	236	
26	1CCR	a	11	21	96	109	A	99	4MDH	b	105	117	132	141	
27	2CCY	a	40	53	79	102		100	4MDH	b	215	218	221	237	B
28	2CCY	a	79	102	104	125	A	101	1MLT	a	2	10	13	25	B
29	2CCY	b	40	53	79	102		102	1MLT	b	2	10	13	25	B
30	2CCY	b	79	102	104	125	A	103	1MBD	a	21	35	101	118	A
31	2CYP	a	16	32	104	118	A	104	1MBD	a	52	57	59	76	
32	2CYP	a	43	54	104	118		105	1PSG	a	272	275	304	307	D
33	2CYP	a	165	172	255	271		106	2PFK	a	16	29	258	276	C
34	2CYP	a	201	208	233	240		107	2PFK	a	40	46	79	92	
35	2CYP	a	233	240	242	252		108	2PFK	a	103	114	296	299	C
36	3C2C	a	4	10	98	108		109	3PGK	a	237	243	246	251	B
37	3C2C	a	64	72	98	108		110	3PGK	a	315	325	348	359	A
38	2CDV	a	65	70	79	87		111	3PGM	a	29	45	58	71	A
39	2CDV	a	79	87	91	98	B	112	3PGM	a	99	103	107	111	
40	1CC5	a	12	17	20	23	B	113	3PGM	a	154	164	166	169	D
41	1CC5	a	12	17	74	85		114	1BP2	a	2	12	90	107	
42	351C	a	40	50	68	79	B	115	1BP2	a	40	55	90	107	
43	451C	a	40	50	68	79	B	116	1BP2	a	59	63	90	107	
44	2CPP	a	157	167	219	224		117	1PRC	c	102	120	122	125	D
45	1ETU	a	24	39	174	179		118	1PRC	c	122	125	262	280	C
46	1ECD	a	20	30	94	111		119	1PRC	c	132	136	305	309	C
47	4FXN	a	11	25	125	136	A	120	1PRC	h	12	30	32	35	D
48	3GRS	a	30	41	339	354	C	121	1PRC	h	107	110	232	248	C
49	3GRS	a	56	61	63	80		122	2PRK	a	104	117	139	150	
50	3GRS	a	56	61	96	120		123	2SNS	a	99	105	122	134	
51	1GPD	r	130	133	149	165	D	124	1SBT	a	6	11	13	17	
52	1GOX	a	19	26	165	169	C	125	1SBT	a	13	17	270	274	C
53	1GOX	a	88	98	309	341	C	126	1SBT	a	104	116	133	145	
54	1HMQ	cd	19	37	19	37	C	127	1TEC	e	14	17	20	23	B
55	2HHB	a	21	35	96	112	A	128	1TEC	e	20	23	273	277	C
56	2HHB	a	76	79	81	89	D	129	1TEC	e	112	124	141	152	
57	2HHB	a	96	112	119	136	B	130	1TEC	e	245	254	273	277	
58	2HHB	b	20	34	51	56	A	131	3TLN	a	137	151	159	180	A
59	2HHB	b	20	34	58	75	A	132	3TLN	a	159	180	225	229	
60	2HHB	b	20	34	101	118	A	133	3TLN	a	260	273	281	296	B
61	2HHB	c	21	35	96	112	A	134	1TIM	a	131	136	139	153	B
62	2HHB	c	76	79	81	89	D	135	1WSY	b	88	97	114	126	
63	2HHB	c	96	112	119	137	B	136	1WSY	b	114	126	146	151	
64	2HHB	d	20	34	51	56	A	137	1WSY	b	207	219	235	241	
65	2HHB	d	20	34	58	75	A	138	1WSY	b	329	343	349	364	A
66	2HHB	d	20	34	101	118	A	139	2TS1	a	145	149	152	155	D
67	4HHB	a	21	35	96	112	A	140	2TS1	a	164	184	196	210	A
68	4HHB	a	76	79	81	89	D	141	2TS1	a	248	256	293	307	
69	4HHB	a	96	112	119	136	B	142	2STV	a	102	105	117	121	D
70	4HHB	b	20	34	51	56	A	143	4SBV	b	93	96	98	104	D
71	4HHB	b	20	34	58	75	A	144	4SBV	c	93	96	100	104	B
72	4HHB	b	20	34	101	118	A	145	4SBV	c	202	211	216	219	B
73	4HHB	c	21	35	96	112	A								

Table 2: 145 Pairs of α-Helix Super-Secondary Structures.
This table gives the protein ID from PDB, the chain ID, and start and end locations of 145 α-helix pairs for clustering.

```
 93 90 88 82 81 80 74 68 62 56 44 40 38 38 35 34 24 8
 7 5 23 21 10 6 20 11 3 54 53 52 51 48 22 2 17 12 1}
Node{145 144 143 142 139 137 127 124 120 113 112 109
    105 100 99 98 96 93 90 74 68 62 56 44 40 38 36 35
    34 24 8 7 5 23 21 10 6 20 11 3}
  Node{144 143 127 74 88 62 56 20 11}
    Node{144}
    Node{143 20 74 68 62 56}
      Node{143}
      Node{20}
      Node{74 88 62 56}
        Node{74}
        Node{88}
        Node{62}
        Node{56}
    Node{127 11}
      Node{127}
      Node{11}
  Node{145 142 139 124 112 105 98 40 36 35 23
     10 6 34 21 24}
    Node{96 34 38}
      Node{98}
      Node{34}
      Node{36}
    Node{142 139 105 21}
      Node{142}
      Node{139}
      Node{105}
      Node{21}
    Node{112 35 23 10 6}
      Node{23}
      Node{10}
      Node{112 35 6}
        Node{112}
        Node{35}
        Node{6}
    Node{145 124 40 24}
      Node{145}
      Node{124}
      Node{40}
      Node{24}
  Node{120 109 100 98 38 3}
    Node{120}
    Node{109 38 3}
      Node{109}
      Node{38}
      Node{3}
    Node{100 98}
      Node{100}
      Node{98}
  Node{137 113 99 93 90 44 8 7 5}
    Node{93 8}
      Node{93}
      Node{8}
    Node{137 99 90 7}
      Node{137}
      Node{99}
      Node{90}
      Node{7}
    Node{113 44 5}
      Node{113}
      Node{44}
      Node{5}
Node{128 125 121 119 118 117 108 107 106 88 82 81
    80 54 53 52 51 48 22 2 17 12 1}
  Node{117}
  Node{128 125 119 108 81 52 2 17 12 1}
    Node{128 125 119 17 1 81 12}
      Node{119}
      Node{128 125 17}
        Node{128}
        Node{125}
        Node{17}
      Node{1}
      Node{81 12}
        Node{81}
        Node{12}
    Node{108 2 52}
      Node{108 2}
        Node{108}
        Node{2}
      Node{52}
  Node{121 118 107 106 54 51}
    Node{107}
    Node{106}
      Node{54}
      Node{121 118 51}
        Node{121}
        Node{118}
        Node{51}
    Node{88 82 80 53 48 22}
      Node{82}
      Node{53}
      Node{88 80 48}
        Node{88}
        Node{80}
        Node{48}
      Node{22}
Node{141 140 138 136 135 134 133 132 131 130 129 126
    123 122 116 115 114 111 110 104 103 102 101 97
    95 94 92 91 89 87 86 85 84 83 79 78 77 76 75
    73 72 71 70 69 67 68 85 64 83 61 60 59 58 57
    55 50 49 47 46 45 43 42 41 39 37 33 32 31 30
    29 28 27 26 25 19 18 16 15 14 13 9 4}
  Node{141 140 138 132 131 123 111 110 104 103 91 87
      86 84 79 78 76 71 65 59 50 47 37 33 31 30 29 28
      27 26 18 9 4 77 73 72 70 67 66 64 61 60 58 55 41
      19 18 15 14}
    Node{141 140 132 111 104 103 86 84 79 37 31 26
        16 9 78 4 131 123 91 87 30 28 50 33 29 27}
      Node{140 111 9 31}
        Node{140}
        Node{111 9}
          Node{111}
          Node{9}
        Node{31}
      Node{103 86 84 79 26}
        Node{103 86 84}
          Node{103}
          Node{86}
          Node{84}
        Node{79}
        Node{26}
      Node{132 37 16}
        Node{132}
        Node{37}
        Node{16}
      Node{104 78 4}
        Node{104 78}
          Node{104}
          Node{78}
        Node{4}
      Node{141 131 123 91 87 30 28 50 33 29 27}
        Node{123 91 27 29}
          Node{123 29 27}
            Node{123}
            Node{29}
            Node{27}
          Node{91}
        Node{50}
        Node{141 33}
          Node{141}
          Node{33}
        Node{131 87}
          Node{131}
          Node{87}
        Node{30 28}
          Node{30}
          Node{28}
    Node{138 110 76 71 65 59 47}
      Node{138}
      Node{76 71 65 59}
        Node{76 71 65}
          Node{76}
          Node{71}
          Node{65}
        Node{59}
      Node{110 47}
        Node{110}
        Node{47}
    Node{77 73 72 70 67 68 64 61 60
        58 55 41 19 18 15 14}
      Node{70 64 58 18 19}
        Node{70 64 56}
          Node{70}
          Node{64}
          Node{58}
        Node{18}
        Node{19}
      Node{77 73 72 67 66 61 60 55}
```

```
Node{77 72 68 60}
   Node{77 66}
      Node{77}
      Node{66}
   Node{72 60}
      Node{72}
      Node{60}
Node{73 67 61 55}
   Node{73 81}
      Node{73}
      Node{61}
   Node{67 55}
      Node{87}
      Node{55}
Node{41 15 14}
   Node{41}
   Node{15}
   Node{14}
Node{136 135 134 133 130 129 128 122 118 115 114 102
    101 97 95 94 92 89 85 83 75 69 63 57 49 46 45
    43 42 39 32 25 13}
   Node{135 133 130 129 126 122 115 114 97 94 89 46
       45 32 43 42 25}
      Node{129 128 122 115}
         Node{129 126 122}
            Node{129}
            Node{126}
            Node{122}
         Node{115}
      Node{97 43 42}
         Node{97}
         Node{43}
         Node{42}
      Node{25}
      Node{135 133 130 114 94 89 46 45 32}
         Node{135 133 130 89 32 94 45}
            Node{133 130}
               Node{133}
               Node{130}
            Node{89}
            Node{135 32}
               Node{135}
               Node{32}
            Node{94 45}
               Node{94}
               Node{45}
         Node{114 46}
            Node{114}
            Node{46}
Node{134 116 102 101 92 85 83 49}
   Node{102 101 92}
      Node{102 101}
         Node{102}
         Node{101}
      Node{92}
   Node{134 85 83}
      Node{134}
      Node{85}
      Node{83}
   Node{116 49}
      Node{116}
      Node{49}
Node{136 95 75 69 63 57 39 13}
   Node{136 95 39 13}
      Node{136}
      Node{95}
      Node{39}
      Node{13}
   Node{75 69 63 57}
      Node{75}
      Node{69}
      Node{83}
      Node{57}
```

Example 1: Cobweb/3 Classification Tree of 145 α-Helix Pairs.
A node is either a class node that contains several α-helix pairs or a terminal node with only a single pair.

The following are the parameter control files for AutoClass III for the classification of α-helix pairs. Example 2 shows the classification result of AutoClass III.

```
AutoClass HD2 File
;;; Mode: LISP; Syntax: Common-lisp; Package: AUTOCLASS; Base: 10

(DEFINE-DATA-FILE-FORMAT
  :number-of-attributes 32
  :separator-chars '(#\space)
  :unknown-tokens '(?)
  :data-syntax :line)

(DEFINE-ATTRIBUTE-DEFINITIONS
'(
  ( 0 real scalar "a1_length" (zero-point 0 rel-error 0.02))
  ( 1 real scalar "a1_access" (zero-point 0 rel-error 0.02))
  ( 2 real scalar "a1_bending" (zero-point 0 rel-error 0.02))
  ( 3 real scalar "a1_gravity_distance" (zero-point 0 rel-error 0.02))
  ( 4 real scalar "a1_polar_pos" (zero-point 0 rel-error 0.02))
  ( 5 real scalar "a1_polar_neg" (zero-point 0 rel-error 0.02))
  ( 6 real scalar "a1_twisting" (zero-point 0 rel-error 0.02))
  ( 7 real scalar "a1_hydrophobic" (zero-point 0 rel-error 0.02))
  ( 8 real scalar "a1_salt_bridge_pos" (zero-point 0 rel-error 0.02))
  ( 9 real scalar "a1_salt_bridge_neg" (zero-point 0 rel-error 0.02))
  ( 10 real scalar "a2_length" (zero-point 0 rel-error 0.02))
  ( 11 real scalar "a2_access" (zero-point 0 rel-error 0.02))
  ( 12 real scalar "a2_bending" (zero-point 0 rel-error 0.02))
```

```
  ( 13 real scalar "a2_gravity_distance" (zero-point 0 rel-error 0.02))
  ( 14 real scalar "a2_polar_pos" (zero-point 0 rel-error 0.02))
  ( 15 real scalar "a2_polar_neg" (zero-point 0 rel-error 0.02))
  ( 16 real scalar "a2_twisting" (zero-point 0 rel-error 0.02))
  ( 17 real scalar "a2_hydrophobic" (zero-point 0 rel-error 0.02))
  ( 18 real scalar "a2_salt_bridge_pos" (zero-point 0 rel-error 0.02))
  ( 19 real scalar "a2_salt_bridge_neg" (zero-point 0 rel-error 0.02))
  ( 20 real scalar "r_angle" (zero-point 0 rel-error 0.02))
  ( 21 real scalar "r_orientation" (zero-point 0 rel-error 0.02))
  ( 22 real scalar "r_h_bond" (zero-point 0 rel-error 0.02))
  ( 23 real scalar "r_h_bond_str" (zero-point 0 rel-error 0.02))
  ( 24 real scalar "r_s_bond" (zero-point 0 rel-error 0.02))
  ( 25 real scalar "r_turn_con_both" (zero-point 0 rel-error 0.02))
  ( 26 real scalar "r_dist_centre" (zero-point 0 rel-error 0.02))
  ( 27 real scalar "r_dist_near" (zero-point 0 rel-error 0.02))
  ( 28 real scalar "r_dist_same_ss" (zero-point 0 rel-error 0.02))
  ( 29 real scalar "r_dist_all_ss" (zero-point 0 rel-error 0.02))
  ( 30 real scalar "r_dist_aa" (zero-point 0 rel-error 0.02))
  ( 31 real scalar "r_dist_lines" (zero-point 0 rel-error 0.02))
 )
)
```

AutoClass Command File

```
(load "load-ac.lisp")
(in-package 'ac)
(autoclass-search :data-file "data/weekend/hh" :output-files-default
    "hh" :try-fn 'CONVERGE-SEARCH-1 :user-report 'printclsf)
```

```
CLASS  1
8    1.00
13   1.00
16   0.44
17   0.99
20   1.00
21   1.00
23   1.00
24   1.00
34   1.00
35   1.00
36   1.00
37   0.99
80   0.99
81   1.00
82   1.00
89   1.00
91   1.00
93   1.00
95   1.00
105  1.00
106  1.00
109  1.00
112  0.99
119  1.00
122  0.99
125  1.00
128  1.00
129  1.00
135  1.00
136  1.00
139  1.00
142  1.00
143  1.00
144  1.00
145  1.00

CLASS  2
9    1.00
18   1.00
19   1.00
26   1.00
31   0.99
47   1.00
58   1.00
59   1.00
60   1.00
64   1.00
65   1.00
66   1.00
70   1.00
71   1.00
72   1.00
76   1.00
77   1.00
87   1.00
103  1.00
140  1.00

CLASS  3
14   1.00
15   1.00
16   0.56
27   1.00
28   1.00
29   1.00
33   1.00
41   1.00
45   1.00
46   0.99
53   1.00
54   1.00
79   1.00
84   1.00
86   1.00
97   1.00
110  1.00
114  0.99
123  1.00
138  1.00

CLASS  4
3    1.00
5    1.00
11   1.00
30   1.00
56   1.00
62   1.00
68   1.00
74   1.00
104  1.00
117  1.00
120  1.00
124  1.00

CLASS  5
10   0.99
12   1.00
39   1.00
50   1.00
51   1.00
52   1.00
94   1.00
107  1.00
118  1.00
121  1.00
141  1.00

CLASS  6
```

```
1    1.00
2    1.00
7    0.99
22   0.99
48   1.00
88   1.00
90   0.94
96   0.99
99   0.99
108  1.00
130  1.00

CLASS  7
6    1.00
42   1.00
43   1.00
92   1.00
111  1.00
127  1.00
131  1.00
133  1.00
137  1.00

CLASS  8
25   1.00
32   1.00
38   1.00
49   1.00
115  1.00
116  1.00
132  1.00

CLASS  9
57   1.00
63   1.00
69   1.00
75   1.00
126  1.00

CLASS 10
40   1.00
44   1.00
101  1.00
102  1.00
113  0.99

CLASS 11
55   1.00
61   1.00
67   1.00
73   1.00

CLASS 12
83   1.00
85   1.00
100  1.00
134  1.00

CLASS 13
4    1.00
78   0.99
98   1.00
```

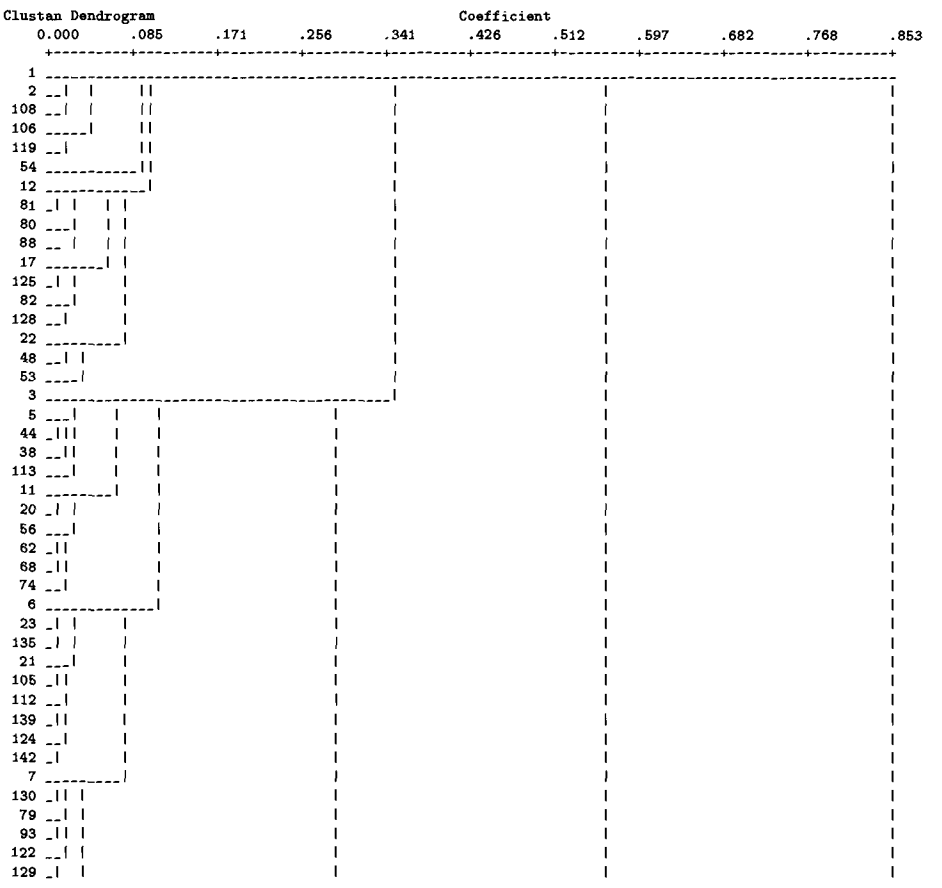
Example 2: AutoClass III Classification of 145 α-Helix Pairs.
This is a cross reference of all classes and helix pairs. This classification has an approximate marginal likelihood of -21514.78. All 145 α-helix pairs have been uniquely assigned to one of the 13 classes, except pair no. 16, which fits about equally into classes 1 and 3. The requirement for independence is not met by all attributes in PRL. However, partially redundant attributes were kept in the database to maximise information content.

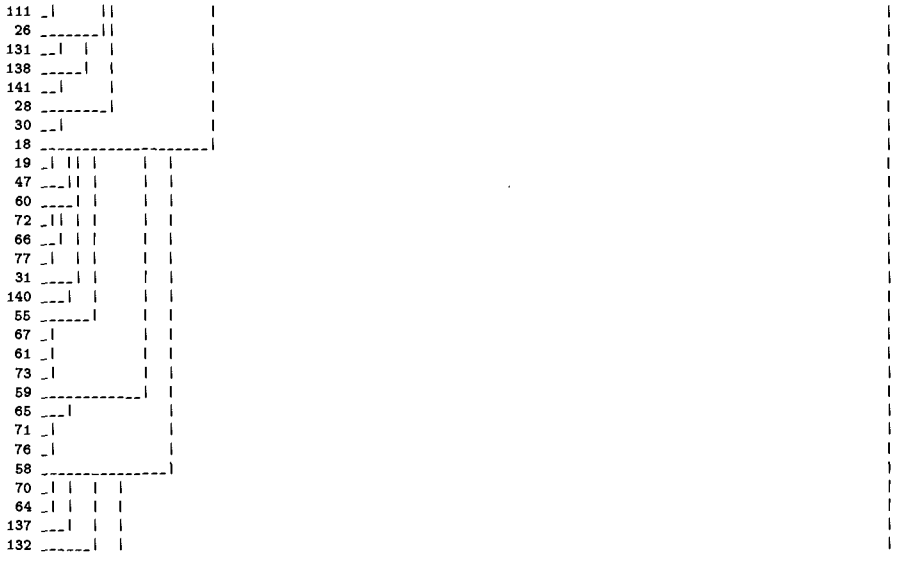
For comparison, a statistical clustering was performed. The following is a parameter control file for the program Clustan[20] to perform Ward's single linkage clustering method. Example 3 shows the classification produced by Clustan for the 145 α-helix pairs.

```
set mode batch
select graphics device, plotter
assign, file = hhtree, spec = 'HH_TREE'
assign, file = datahh, spec = 'DATA_HH'
comment assign, file = hhcls, spec = 'HH_CLS'
assign, file = hhwrit, spec = 'HH_CLASS'
read data, infile = datahh, variables continuous 1 - 32,
missingvalues = 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
                0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
                0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
                0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
title = "145 HH Pairs, new data set", format = (32F8.4)
read labels variables
a1_lengta1_accesa1_bendia1_gravia1_polapa1_polana1_twista1_hydroa1_saltp
a1_saltna2_lengta2_accesa2_bendia2_gravia2_polapa2_polana2_twista2_hydro
a2_saltpa2_saltnr_angle r_orientr_h_bondr_h_bondr_s_bondr_turn_cr_dist_c
r_dist_nr_dis_smr_dis_alr_dis_aar_dis_ln
mask variables 7, 17, 22
standardise all
calculate factor scores, factors = 10, save cmatrix, save eigenvectors
```

20 D. Wishart, *Hierarchical cluster analysis with messy data*, in **Classification as a Tool of Research**, (Eds. W. Gaul, M. Schader), Elsevier Science Publishers, pp. 453-460, 1986. Address: Clustan, 16 Kingsburgh Road, Murrayfield, Edingurgh EH12 6DZ, Scotland, Tel. + 44 – 31 – 337 1448.

```
calculate similarity, matrix, klinkages = 5
cluster, print icicle min 5 max 15 inc 1, print centres min 5 max 15,
         print members min 5 max 15, save tree, outfile hhtree, clusters
max 15 min 5
comment  save results, outfile = hhcls,
plink, infile = hhtree
relocate, start code = 1, coefficient = 24, fusion, min = 5, relocations,
maxit = 10

comment plink, infile = $dendro
relocate, coefficient = 24, start random size = 15
comment plink, infile = $dendro
density, k = 3
hierarchy, method average, clusters min = 5, max = 15
scatter, xaxis factor 1, yaxis factor 2, splot axes, cplot axes codes
         circles
classification code 5
tree, infile = hhtree
write classifications, codes = 1 - 10, outfile = hhwrit
print results
stop
```




126
8
52
99
136
4
78
104
49
116
115
98
120
10
35
36
37
16
89
90
14
15
41
34
96
27
29
110
123
32
46
114
45
91
97
84
86
103
87
33
118
107
121
50
51
143
9
111
26
131
138
141
28
30
18
19
47
60
72
66
77
31
140
55
67
61
73
59
65
71
76
58
70
64
137
132

Example 3: Statistical Cluster Analysis of 145 α-Helix Pairs according to Ward's Method. A uniform Euclidean metric was used to compare the 32-dimensional vectors of 145 α-helix pairs. The similarity coefficient provides a quantitative measure of similarity. Missing values were encoded by a unique value that did not occur elsewhere in the database. The requirement for independence is not met by all attributes in PRL. However, partially redundant attributes were kept in the database to maximise information content. The similarity coefficient s_{ik} between the examples i and k over all attributes j is defined as follows: $s_{ik} = \frac{\sum_j w_{ikj} s_{ikj}}{\sum_j w_{ikj}}$, $s_{ikj} = \frac{(x_{ij}-x_{kj})^2}{R_j^2}$ with x_{ik} as the value of attribute k of example j and w_{ikj} the comparison weight for attribute j in examples i and k. w_{ikj} is 1 if both examples have valid, observed values for attribute j and 0 if there is at least one value missing or otherwise invalid. s_{ikj} is the similarity coefficient for examples i and k concerning attribute j. R_j is the range of values for attribute j.

Combining the three clusterings produces a consensus classification (Table 3). The consensus clustering has four main clusters of super-secondary structure: class A with 31 pairs, class B with 25 pairs, class C with 20 pairs, and class D with 15 pairs. All other α-helix pairs were considered in a "dustbin" class. The four classes of α-helix pairs have distinct attribute properties and one can form a simple decision tree to distinguish classes B, C, and D from class A and the dustbin class (Figure 8).

In **Class A**, both α-helices are longer and have larger polar moments than the average of all helices in the 145 α-helix pairs. Helix 1 has a large hydrophobic moment and both α-helices are rarely connected through H-bonds. The centres distance relation has a lower value than average.

In **Class B**, all the examples are turn-connected: the examples consist of three pieces of secondary structure: an α-helix, a turn (DSSP type T or S), and a second α-helix. This is of interest because even though only *two* elements were explicit, a super-secondary structure with *three* elements was found by clustering. It can be concluded that significant chemo-physical, topological and geometric properties exist in PRL that allow identification of a complex super-secondary structure (the

Class	Main Classes	Subclasses
A	9, 14, 18, 19, 26, 28, 30, 31, 47, 55, 58, 59, 60, 61, 64, 65, 66, 67, 70, 71, 72, 73, 76, 77, 87 103, 110, 111, 131, 138, 140	((55,67), (61,73)), ((60, 72), (66, 77)), (58, 64, 70), (28, 30), (59, (65, 71, 76)), (18, 19)
B	13, 24, 25, 39, 40, 42, 43, 57, 63, 69, 75, 83, 85, 92, 94, 95, 100, 101, 102, 109, 127, 133, 134, 144, 145	((83, 85), 92, (101, 102)), (13, 24, 25, 39, (42, 43), (57, 63, 69, 75), 94, 95, 100, 109, 127, 133, 144, 145)
C	1, 2, 12, 17, 22, 48, 52, 53, 54, 80, 81, 82, 88, 106, 108, 118, 119, 121, 125, 128	(80, 88)
D	11, 20, 21, 51, 56, 62, 68, 74, 105, 113, 117, 120, 139, 142, 143	(11, 21, 105), (20, (56, 62, 68, 74), 142, 143) (113, 117, 120, 139)

Table 3: Consensus Classification of Cobweb/3, AutoClass III and Clustan. Pair numbers and class identifier are the same as in Table 2. Brackets indicate subclusters.

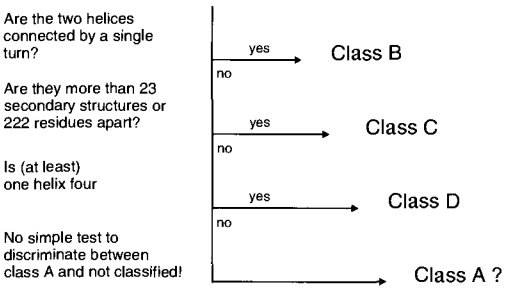


Figure 8: Classification Summary for α-Helix Pairs. The diagram shows the simplest constraints to discriminate between classes B, C and D. Class A cannot be characterised by only one attribute.

α-helix–turn–α-helix motif) based on only a partial description of its components. It is also interesting that class B was found to be distinct from pairs where there is no turn separating the helices but only coiled structure. Because the α-helices are turn-connected in class B, the following attributes are small:

- distance in secondary structures of the same type,
- distance in secondary structures of all types, and
- distance in residues.

Both helices have their gravitational centres close to the overall gravitational centre of the protein which means that they are near the centre of the protein, i.e. in the inner core. Helix 2 has fewer than average negative salt bridges. There are 2 distinct sub-classes in class B discovered by the clustering algorithms which when exam-

ined using a molecular modelling package have a quite different structural appearance. In one sub-class a "V"-shape is formed by the two helices of a pair (Examples 83, 85, 92, 101, 102 in Figure 10) and in the other the helices fold back on themselves ("U"-shape in Figure 11). The "V"-shaped cluster is mainly responsible for the large distance between the centres of the secondary structures (all distances are greater than 18 Å) and for the high solvent accessibility values (helix 1 average 13.6%, helix 2 average 12.42%).

In **Class C**, both helices of a pair are located extremely far from each other along the polypeptide chain. The relation attributes: distance in secondary structures of the same type, distance in secondary structures of all types and distance in residues are much larger than average. All distances in secondary structures of the same type are greater than 23. All helices are at least 222 residues apart in sequence. No other examples are so far separated. This class can truly be said to be predominately defined by *long range interactions*. The detection of such super-secondary structures could not be achieved by other methods that concentrate on secondary structures near in sequence. Both helices of class C pairs have low solvent accessibilities and large distances from the centre of gravity of the protein. This may result from the occurrence of class C pairs in larger than average proteins. Both helices have low values for negative polarity moment. Helix 1 also has a low hydrophobic moment. Both helices are commonly connected by H-bonds. The latter result may be biased as all the globin examples are H-bond connected.

In **Class D** all helices are very short. In all the examples, either helix 1 or helix 2 has a length of four residues (by definition a length of four amino acid residues is the minimum for a helix in PRL). Both helices are located near in sequence. All examples (except pair no. 20) are pairs of helices directly in sequence with no helices separating them (distance in secondary structures of the same type is one). Example 20 has only one helix separating the helices of the pair (distance in secondary structures of the same type is two). Both helices are distant from the gravitational centre of the protein and have a low positive polar moment. Helix 1 also has a low negative polar moment and low hydrophobic moment. In the terminology of Murzin and Finkelstein[21]: classes A, C, and D form longitudinal links; in class B, the sub-group (83, 85, 92, 101, 102) forms end-to-end links and no longitudinal links, while the other sub-group forms both.

The distribution of the examples in the classes also varied by domain type. The examples in classes A and B were found to occur almost exclusively in proteins of alpha domain type (class A mostly in globins), whereas the examples in class C were found to occur almost exclusively in proteins of α/β domain type. This is interesting because no information about domain type was explicit in the clustering data (domain information is at a higher level than used in the clustering).

Another way of examining the biological value of the clusters is to find out how similar the conformations of pairs are within one class and how they differ between distinct classes. This can be determined by calculating *r.m.s. fits* between different structures. The r.m.s. (root mean square deviation) measures the average distance between any two corresponding atoms in two different conformations P and Q after they have been aligned optimally and superimposed:

21 A. G. Murzin, A. V. Finkelstein, **Journal of Molecular Biology**, vol 204, pp. 749-769, 1988.

Class	A	B	C	D	No class	All
Alignments	465	300	190	105	1431	10440
r.m.s.-Deviation(Å)	4.68	3.61	3.55	3.00	4.43	4.14
Standard Deviation	2.20	1.70	1.31	1.17	1.65	1.73

Table 4: R.m.s. Deviations within Classes A, B, C, D and non-classified Pairs. Classes B, C and D show significantly lower r.m.s. deviations than non-classified α-helix pairs or the average over all pairs. Although tertiary structure and conformation *per se* were not explicitly included in the input data for clustering, the classes B, C and D exhibit strong conformational similarities. This result was expected for a biologically meaningful classification of super-secondary structures. Class A is populated mainly by very large helices which accounts for the larger than average structural difference. The non-classified pairs are likely to contain more dissimilar conformations and thus give a larger r.m.s. difference.

$$r.m.s._{PG} = \sqrt{\frac{\sum_{i=1}^{N}((x_P - x_Q)^2 + (y_p - y_Q)^2 + (z_P - z_Q)^2)}{N}}\,[\text{Å}].$$

Here, x, y and z are the three spatial coordinates of atom i in the structures P and Q. R.m.s. deviation for proteins is usually given in units of 10^{-10} m or Ångstrom (1Å = 10^{-10} m).

The average r.m.s. values for helix pairs within the four classes A, B, C, D and the dustbin class were calculated and compared with the total set of 10440 r.m.s. distances for every possible superposition among the 145 helix pairs (Table 4). If matching helices of two pairs are of different length the smaller helix is slid residue by residue against the larger one and all combinations are calculated. The smallest r.m.s. difference is then chosen to represent this comparison. All values are in the range of 3 to 5 Å which, when compared to values from studies of closely homologous proteins, indicates a relative large geometrical difference in the conformations. All 145 helix pairs have an average length of about 24 amino acid residues (residues of both helices of a pair added). Comparing this length and r.m.s. difference with random values[22] as taken from R. Unger *et al* who state a r.m.s. difference of 7.26 Å for random fragments 24 amino acids in length, shows that the whole collection of α-helix pairs and the classifications are far from being random.

To be able to give a more accurate estimation of how significant the differences in r.m.s. values of the various classes are compared with the average r.m.s. value over all 10440 possible combinations of α-helix pairs, Monte-Carlo statistics[23] were carried out. The test statistic is the mean r.m.s. of each class of α-helix pairs which is compared to the overall mean r.m.s. of all possible α-helix pair combinations. The results of these calculations are summarised in Table 5. The deviation of the r.m.s. distances of each class from the overall mean are highly significant and reject the

22 R. Unger, D. Harel, S. Wherland, J. L. Sussman, *A 3D Building Blocks Approach to Analyzing and Predicting Structure of Proteins*, **Proteins**, vol 5, pp. 355-373, 1989.

23 E. W. Noreen, **Computer Intensive Methods for Testing Hypotheses**, John Wiley, 1987.

Class	A	B	C	D	no class
Population Size	465	300	190	105	1431
Iteration	9999	9999	9999	9999	9999
Test	$A>m$	$B<m$	$C<m$	$D<m$	$X>m$
Error	0.01	0.01	0.01	0.01	0.01
Significance	1.0000	1.0000	1.0000	1.0000	1.0000

Table 5: Monte Carlo Statistics on Mean R.m.s. Distances.
Draw as many as "iteration" times randomly the number of "population size" pairs from all α-helix pairs and compare their mean r.m.s. to the overall mean r.m.s with the test $X>m$ (or $X<m$).

null hypothesis at a significance level of 0.01 that the clustered classes were taken randomly from the set of all possible combinations.

Thus, even though the clustering procedures were not directly guided by tertiary structure the resulting classifications show significant geometrical similarities. This confirms the biochemical relevance of the classifications produced. The approach of searching for similarities in a multi-dimensional space with more than the mere spatial coordinates can also help avoid the problem which Unger *et al* encountered when they found that intuitively similar building blocks have a higher r.m.s. difference than apparently dissimilar structures. The IPSA methodology utilises more types of information than only spatial conformation and still arrives at geometrical similar structures. However, the presence of rather dissimilar structures in class A must be taken into account.

4.2.1.4. Helix/β-Strand Pairs

279 pairs of one helix and one β-strand were extracted from the PRL database which were either connected by hydrogen or disulphide bridges or the backbones of which are at any one point 10 Å or less apart (Table 6). These pairs were clustered with the programs Cobweb/3, AutoClass III and Clustan and the results combined to produce a consensus classification with twelve main clusters of super-secondary structure (the number of examples per class is given in parentheses): A (23), B (15), C (18), D (13), E (8), F (8), G (8), H (8), I (5), K (5), L (5) and M (6).

Class A is the only class with disulphide bridges between both secondary structures. Every pair in this class has one disulphide bridge. The hydrophobic moment of the helix is about 40% less and the bending of the β-strand significantly smaller than for the average of all helix/β-strand pairs. There are no H-bonds between the helix and the β-strand. **Class B** has helices of the length four residues and β-strands of the length two. Because of the short length, bending and twisting of the β-strand, salt bridges and bending of the helix are zero. There are on the average two other secondary structures or seven residues between the β-strand and helix. There are no H-bonds between both. The helices in **Class C** have strong polar and hydrophobic moments. In 94% of all pairs in this class there is exactly one turn between the secondary structures. Helix and β-strand follow each other closely on the primary structure. The helix has twice as many salt bridges and the β-strand a

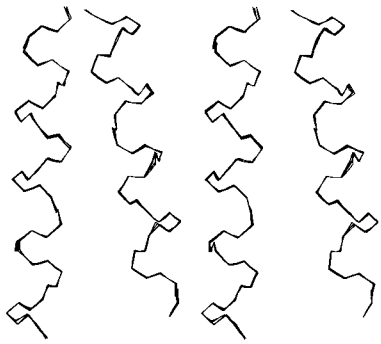


Figure 9

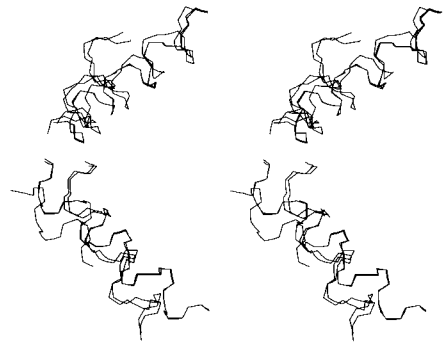


Figure 10

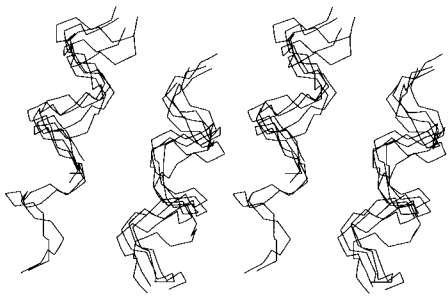


Figure 11

Figure 12

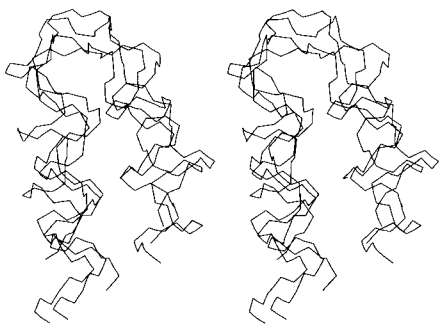


Figure 13

Figure 14

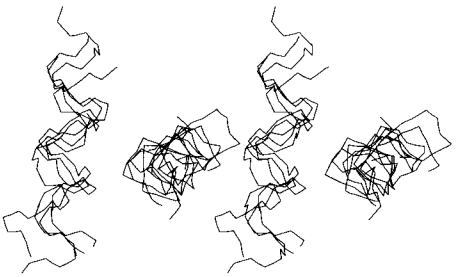


Figure 15

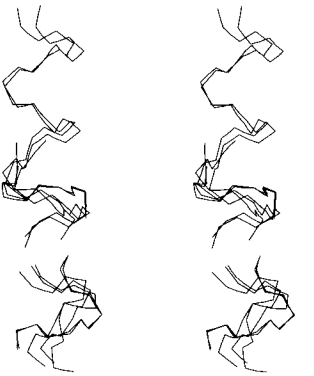


Figure 16

Figures 9 – 16.: Skeletal Drawing of α-Helix Pairs.
Figure 9: α-Helix pair class A3: four homologous α-helix pairs 59, 65 (2HHB), 70, 76 (4HHB), mean pairwise r.m.s. 0.11 Å, stereoprojection.
Figure 10: α-Helix pair class B1 ("V"-shape): pairs 83 (1LH3), 85 (2LH4), 92 (2LZM), 101, 102 (1MLT), mean pairwise r.m.s. 2.21 Å, stereoprojection.
Figure 11: α-Helix pair class B2 ("U"- shape): pairs 13 (1CPV), 24 (2CTS), 25 (1CRN), 39 (2CDV), 42 (351C), mean pairwise r.m.s. 2.31 Å, stereoprojection.
Figure 12: α-Helix pair class B1 with turns ("V"-form): pairs 83 (1LH3), 85 (2LH4), 92 (2LZM), 101, 102 (1MLT), mean pairwise r.m.s. 2.73 Å, stereoprojection.
Figure 13: α-Helix pair class B2 with turns ("U"-form): pairs 13 (1CPV), 24 (2CTS), 25 (1CRN), 39 (2CDV), mean pairwise r.m.s. 2.31 Å, stereoprojection.
Figure 14: α-Helix pair class C: pairs 1, 2 (5ATC), 52, 53 (1GOX), 119 (1PRC), mean pairwise r.m.s. 2.19 Å.
Figure 15: α-Helix pair class C: pairs 1, 2 (5ATC), 52, 53 (1GOX), 119 (1PRC), mean pairwise r.m.s. 2.19 Å, stereoprojection.
Figure 16: α-Helix pair class D: pairs 11, (2AAT), 56 (2HHB), 68 (4HHB), 113 (3PGM), 120 (1PRC), mean pairwise r.m.s. 1.05 Å, stereoprojection.

bending of four degree less than the average of all helix/β-strand pairs. **Class D** has a strong hydrophobic moment of the β-strand and on average the highest energy in H-bonds between helix and β-strand. Helix and β-strand follow each other closely on the primary structure. Bending of the β-strand is 52% larger than the average of all helix/β-strand pairs. **Class E** has strongly bent helices. Each helix/β-strand pair is directly connected by one turn only. The average angle formed by the axes of the two secondary structures is 50° and thus considerably smaller than the average of 88° in all helix/β-strand pairs. The short β-strands have little twist. This class resembles the αtβ-motif, a super-secondary structure made of three secondary structures. It is interesting that information about only two secondary structures was enough for the clustering algorithms to discover a super-secondary structure

composed of *exactly three* elements. **Class F** has large β-strands and helices that are located far from the centre of gravity of large proteins. The β-strands are not exposed to the surface as can be seen from their small relative solvent accessibility. β-Strand and helix are widely separated in primary structure: on the average there are 88 residues or 13 other secondary structures between them in sequence. Thus, this class is predominantly formed by long range interactions. The helix has strong polar and hydrophobic moments and three salt bridges on average. There are no H-bonds between the secondary structures. **Class G**, too, consists of larger than average secondary structures that are far away from each other in sequence. On average, 22 other secondary structures or 130 amino acid residues lie between them. In contrast to class F, bending and twisting of β-strands are larger than average. The helix has a small distance to the centre of the protein, low relative solvent accessibility and a strong polar moment. **Class H** has the β-strand immediately following the helix in sequence, on the average only two residues away. The distance between the centres of the secondary structures is larger than average and they are usually connected by two H-bonds. The helix has no salt bridges and a strong positive polar moment. The β-strand is twice as strongly twisted than the average of all β-strands. **Class I** has only very short helices (four residues long, the minimal size in IPSA) and short β-strands (mean 3.6 residues). **Class K** has long helices with a large positive polar moment. The angle between the axes of helix and β-strand is 154° on average which means that they are oriented almost antiparallel. Both secondary structures are separated by 11 other secondary structures or 54 residues on average. Therefore class K is another example of the effect of *long range interactions*. **Class L** has long helices which lie on the surface of small proteins, as can be seen from their high relative solvent accessibility and small distance to the centre of the protein. They show large polar moments and no salt bridges. The secondary structures enclose an angle of 160° on average, which means they are oriented antiparallel. **Class M** has helices of average length which have large relative solvent accessibility and are far from the centre of the protein. This indicates that the helix of a pair is located towards the surface. The β-strands in this class are on average the largest ones in the whole database set and with a strong twist but average bending. Both secondary structures are connected by one H-bond on average.

The unconditional (Figure 17) and conditional (Figure 18) properties of all classes of helix/β-strand pairs are summarised in two decision trees. To asses the most likely class for a new helix/β-strand pair:

1. Use the unconditional decision tree and check the properties of a new pair for the attribute values given there in any order. If no class membership can be assigned turn to step 2.
2. Use the conditional decision tree. Follow the conditions from top to the bottom of the tree and compare them to the values of the new helix/β-strand pair. If none of the conditions can be met the new pair is likely not to belong to any of the 12 helix/β-strand classes.

Figures 19 – 32 show stereoprojections of the 12 helix/β-strand classes. Although the classifications were not explicitly guided by three-dimensional information about helix and β-strand the backbone conformations within each class are strikingly similar. Helix/β-strand classes C and E are super-secondary structures of three

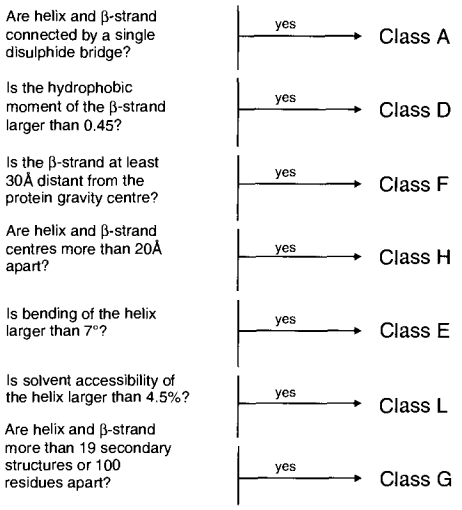


Figure 17: Unconditional Decision Tree for Classification of Helix/β-Strand Pairs. Each condition specifies one class independently. Class G can also be recognised by a β-strand twisted by more than 1.5 rad.

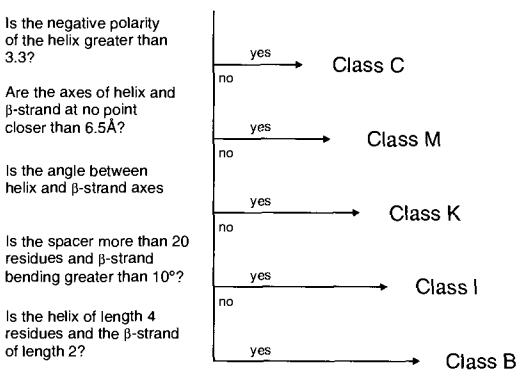


Figure 18: Conditional Decision Tree for Classification of Helix/β-Strand Pairs. The decision tree must be read top-down. Each condition in sequence qualifies for a particular class.

secondary structures because the helix and β-strand of a pair is always immediately connected by a turn which is why for classes C and E backbone drawings with and without the intermitting turns are shown. The numbering of the examples refers to the identification code ("No.") in Table 6. Additionally, the PDB code of the protein source of each example is listed.

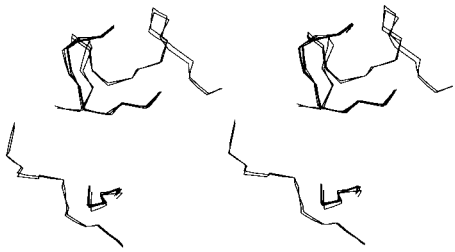


Figure 19

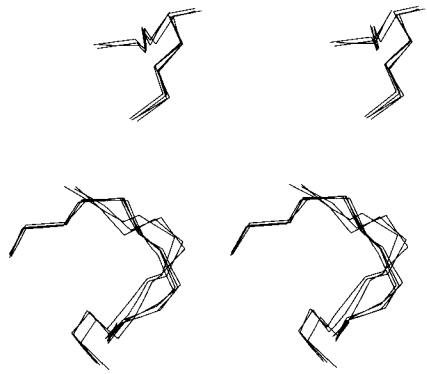


Figure 20

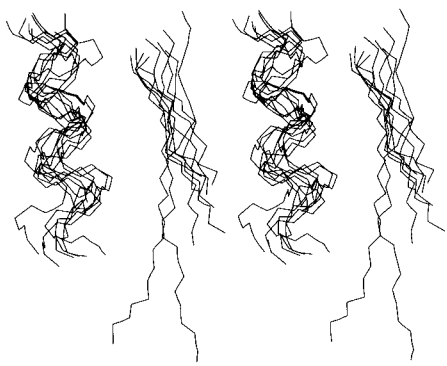


Figure 21

Figure 22

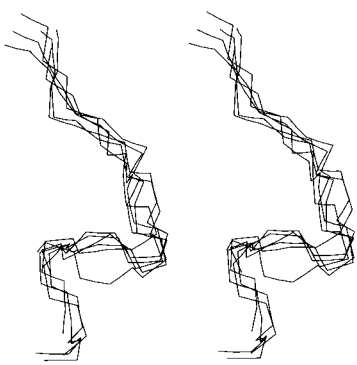


Figure 23

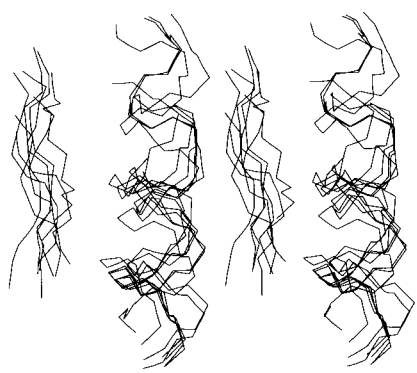


Figure 24

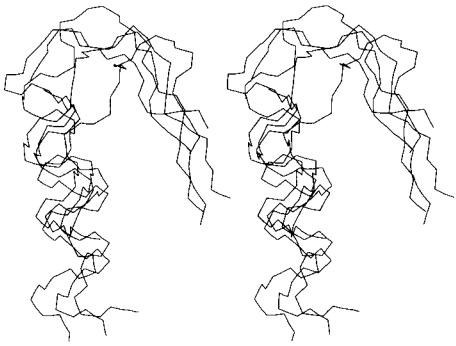


Figure 25

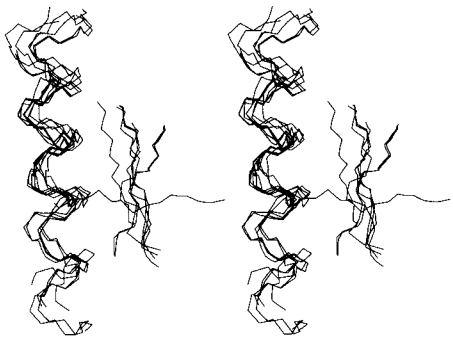


Figure 26

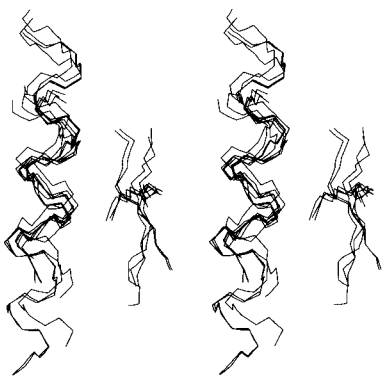


Figure 27

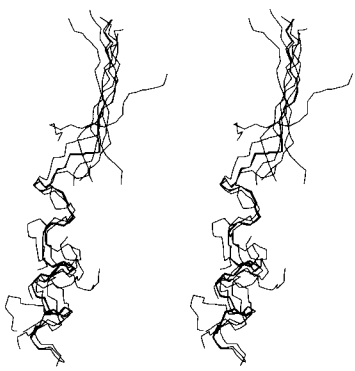


Figure 28

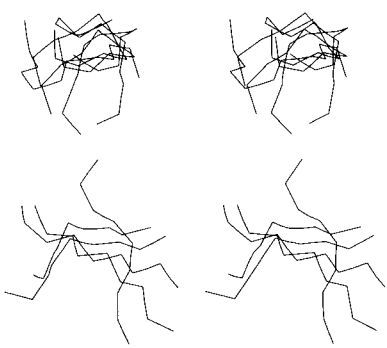


Figure 29

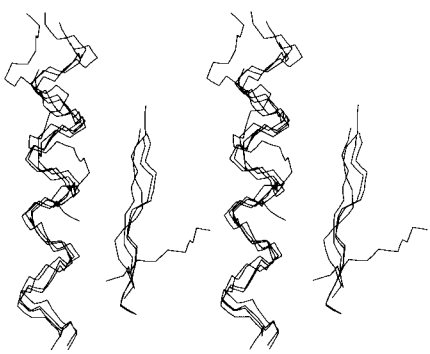


Figure 30

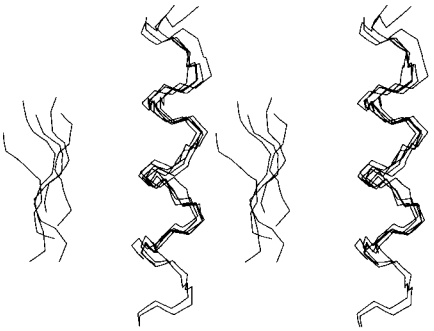


Figure 31

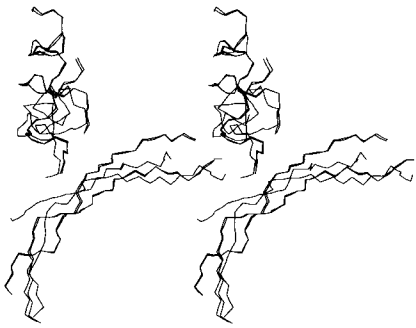


Figure 32

Figures 19 – 32: Skeletal Drawings of Helix/β-Strand Classes.

Figure 19: Helix/β-strand pair class A: pairs 18, 20, 22, 24 (3WGA), 69, 72 (5CHA), mean pairwise r.m.s. 1.88 Å, stereoprojection.

Figure 20: Helix/β-strand pair class B: pairs 19, 21, 23, 25 (3WGA), 271, 274, 277 (4SBV), mean pairwise r.m.s. 1.24 Å, stereoprojection.

Figure 21: Helix/β-strand pair class C: pairs 99, 100 (4FXN), 108 (3GRS), 111, 117 (1GPD), 124 (1GOX), 146 (2LBP), 178 (2PFK), 243 (1TIM), mean pairwise r.m.s. 2.30 Å, stereoprojection.

Figure 22: Helix/β-strand pair class C with turns: pairs 99, 100 (4FXN), 108 (3GRS), 124 (1GOX), 146 (2LBP), 243 (1TIM), mean pairwise r.m.s. 4.01 Å, stereoprojection.

Figure 23: Helix/β-strand pair class D: pairs 8 (5ATC), 53, 54 (1CPV), 84 (2CPP), 140 (6LDH), 157 (4MDH), mean pairwise r.m.s. 1.30 Å, stereoprojection.

Figure 24: Helix/β-strand pair class E (complete): pairs 1 (5ATC), 37 (8ADH), 136 (6LDH), 142 (2LBP), 147 (2LIV), 234 (1TEC), 242 (1TIM), 251 (1WSY), mean pairwise r.m.s. 1.63 Å, stereoprojection.

Figure 25: Helix/β-strand pair class E with turns: pairs 1 (5ATC), 37 (8ADH), 234 (1TEC), 242 (1TIM), 251 (1WSY), mean pairwise r.m.s. 3.26 Å, stereoprojection.

Figure 26: Helix/β-strand pair class F (complete): pairs 114, 120 (1GPD), 145 (2LBP), 150 (2LIV), 179, 181 (2PFK), 190 (1PRC), 257 (1WSY), mean pairwise r.m.s. 2.37 Å, stereoprojection.

Figure 27: Helix/β-strand pair class G (complete): pairs 12 (2ACT), 49 (2AAT), 164, 166 (9PAP), 208, 211 (2PRK), 228 (1SBT), 233 (1TEC), mean pairwise r.m.s. 2.96 Å, stereoprojection.

Figure 28: Helix/β-strand pair class H (complete): pairs 39 (2ALP), 46 (6API), 71, 74 (5CHA), 192, 193, 194 (2PAB), 204 (3SGB), mean pairwise r.m.s. 3.86 Å, stereoprojection.

Figure 29: Helix/β-strand pair class I (complete): pairs 10 (4APE), 67 (1CMS), 173 (1PSG), 232 (1SBT), 268 (2STV), mean pairwise r.m.s. 3.06 Å, stereoprojection.

Figure 30: Helix/β-strand pair class K (complete): pairs 17 (3ADK), 135 (6LDH), 209, 210 (2PRK), 236 (1TEC), mean pairwise r.m.s. 2.51 Å, stereoprojection.

Figure 31: Helix/β-strand pair class L (complete): pairs 14, 15 (3ADK), 58 (5CPA), 131 (1PHH), 137 (6LDH), mean pairwise r.m.s. 1.27 Å, stereoprojection.

Figure 32: Helix/β-strand pair class M (complete): pairs 70, 73 (5CHA), 167 (9PAP), 168 (1PSG), 197 (2SGA), 203 (3SGB), mean pairwise r.m.s. 4.23 Å, stereoprojection.

No.	PDB ID	Chain	StartH	EndH	StartS	EndS	Class	No.	PDB ID	Chain	StartH	EndH	StartS	EndS	Class
1	5ATC	a	111	118	124	126	E	71	5CHA	a	231	244	225	230	H
2	5ATC	a	135	149	124	126		72	5CHA	b	165	172	180	184	A
3	5ATC	a	215	218	156	160		73	5CHA	b	231	244	206	215	M
4	5ATC	a	215	218	183	187		74	5CHA	b	231	244	225	230	H
5	5ATC	a	215	218	224	227		75	2CTS	a	71	77	63	65	
6	5ATC	a	285	290	262	264	D	76	1CRN	a	23	30	33	34	
7	5ATC	b	147	152	134	137	D	77	2B5C	a	9	12	16	17	
8	5ATC	b	147	152	144	146	D	78	2B5C	a	9	12	20	25	
9	4APE	a	225	232	287	289		79	2B5C	a	9	12	52	53	
10	4APE	a	304	307	300	302	I	80	2B5C	a	44	48	16	17	
11	2ACT	a	121	130	5	6		81	2B5C	a	55	60	20	25	
12	2ACT	a	25	42	213	216	G	82	2CDV	a	30	33	36	37	B
13	2ACT	a	70	80	112	115		83	2CPP	a	38	42	391	392	
14	3ADK	a	21	31	35	38	L	84	2CPP	a	150	153	147	149	D
15	3ADK	a	21	31	90	93	L	85	2CPP	a	150	153	403	405	
16	3ADK	a	39	49	35	38	D	86	2CPP	a	253	266	398	399	
17	3ADK	a	39	49	90	93	K	87	8DFR	a	118	126	3	10	
18	3WGA	a	28	31	17	18	A	88	8DFR	a	54	59	47	53	
19	3WGA	a	28	31	24	25	B	89	8DFR	a	54	59	70	75	
20	3WGA	a	71	74	60	61	A	90	8DFR	a	103	106	109	114	
21	3WGA	a	71	74	67	68	B	91	3DFR	a	43	48	38	42	
22	3WGA	a	114	117	103	104	A	92	3DFR	a	43	48	58	62	
23	3WGA	a	114	117	110	111	B	93	3DFR	a	79	88	58	62	
24	3WGA	a	157	160	146	147	A	94	3EST	a	165	168	180	183	A
25	3WGA	a	157	160	153	154	B	95	1ETU	a	183	198	169	172	
26	3WGA	b	28	31	17	18	A	96	4FD1	a	66	75	2	4	
27	3WGA	b	28	31	24	25	B	97	4FD1	a	45	48	54	56	
28	3WGA	b	71	74	60	61	A	98	3FXC	a	26	32	3	8	
29	3WGA	b	71	74	67	68	B	99	4FXN	a	11	25	31	34	C
30	3WGA	b	114	117	103	104	A	100	4FXN	a	94	105	81	88	C
31	3WGA	b	114	117	110	111	B	101	3GAP	a	99	105	38	41	
32	3WGA	b	157	160	146	147	A	102	3GAP	a	99	105	97	98	
33	3WGA	b	157	160	153	154	B	103	3GAP	b	99	108	92	98	
34	8ADH	a	324	336	157	159		104	3GAP	b	180	192	196	198	
35	8ADH	a	229	235	239	241		105	1GP1	a	48	62	67	73	
36	8ADH	a	272	280	264	267		106	1GP1	a	120	128	162	165	
37	8ADH	a	272	280	288	291	E	107	1GP1	b	48	62	67	73	
38	2ALP	a	231	238	209	216		108	3GRS	a	30	41	46	50	C
39	2ALP	a	231	238	226	230	H	109	3GRS	a	197	208	212	216	C
40	2ALP	a	231	238	241	242		110	3GRS	a	440	452	377	381	
41	2TAA	a	50	55	11	14		111	1GPD	g	37	45	57	60	C
42	2TAA	a	236	239	249	250	B	112	1GPD	g	37	45	63	66	
43	6API	a	54	65	111	121		113	1GPD	g	102	111	91	94	C
44	6API	a	54	65	182	191		114	1GPD	g	313	332	115	119	F
45	6API	a	54	65	381	388		115	1GPD	g	252	264	271	274	C
46	6API	a	300	307	291	298	H	116	1GPD	r	37	45	46	47	
47	1ABP	a	210	217	200	202		117	1GPD	r	37	45	57	60	C
48	2AAT	a	84	95	107	112		118	1GPD	r	37	45	63	66	
49	2AAT	a	116	127	268	273	G	119	1GPD	r	102	112	91	94	C
50	2AAT	a	351	354	360	362		120	1GPD	r	313	333	115	119	F
51	2AZA	a	56	66	49	52		121	1GPD	r	149	165	271	272	
52	3BCL	a	183	189	192	194		122	1GPD	r	252	264	271	272	C
53	1CPV	a	60	64	57	59	D	123	1GPD	r	313	333	304	311	
54	1CPV	a	99	107	96	98	D	124	1GOX	a	235	243	248	251	C
55	3CLN	a	65	77	63	64		125	1GOX	a	291	300	304	307	
56	3CLN	a	138	145	136	137		126	1PHH	a	12	24	28	31	
57	5CPA	a	254	260	189	197		127	1PHH	a	53	57	47	49	
58	5CPA	a	216	231	201	204	L	128	1PHH	a	63	68	70	72	
59	5CPA	a	243	246	201	204		129	1PHH	a	63	68	97	99	
60	5CPA	a	216	231	239	242		130	1PHH	a	237	244	218	225	
61	5CPA	a	243	246	239	242		131	1PHH	a	299	317	281	283	L
62	8CAT	a	285	290	228	237		132	2FB4	l	124	128	125	129	
63	1CMS	a	226	236	204	206		133	2PKA	b	165	171	180	184	A
64	1CMS	a	226	236	222	225		134	2PKA	y	166	169	180	184	A
65	1CMS	a	226	236	287	290		135	6LDH	a	55	67	77	80	K
66	1CMS	a	272	275	276	279		136	6LDH	a	140	151	132	135	E
67	1CMS	a	272	275	282	285	I	137	6LDH	a	140	151	158	160	L
68	1CMS	a	300	303	306	311		138	6LDH	a	164	178	185	186	
69	5CHA	a	165	172	180	184	A	139	6LDH	a	211	214	205	206	B
70	5CHA	a	231	244	206	215	M	140	6LDH	a	211	214	209	210	D

(continues on the next page)

No.	PDB ID	Chain	StartH	EndH	StartS	EndS	Class
141	2LBP	a	16	35	43	50	
142	2LBP	a	79	91	95	98	E
143	2LBP	a	299	308	114	116	
144	2LBP	a	149	163	141	145	C
145	2LBP	a	149	163	196	200	F
146	2LBP	a	203	215	221	224	C
147	2LIV	a	79	91	95	98	E
148	2LIV	a	297	306	114	116	
149	2LIV	a	149	163	141	145	C
150	2LIV	a	149	163	196	200	F
151	2LIV	a	276	291	309	311	
152	1LZ1	a	90	100	59	60	
153	1CTF	a	100	113	116	119	
154	4MDH	a	132	141	151	153	
155	4MDH	a	156	170	181	183	
156	4MDH	a	208	212	198	200	D
157	4MDH	a	208	212	205	207	D
158	4MDH	b	132	141	151	153	
159	4MDH	b	156	170	181	183	
160	4MDH	b	208	212	198	201	D
161	1SN3	a	23	30	38	42	
162	1SN3	a	23	30	45	49	A
163	9PAP	a	118	127	5	6	
164	9PAP	a	25	42	207	210	G
165	9PAP	a	68	77	109	112	
166	9PAP	a	68	77	207	210	G
167	9PAP	a	118	127	159	167	M
168	1PSG	a	136	142	94	104	M
169	1PSG	a	225	235	203	205	
170	1PSG	a	225	235	221	224	
171	1PSG	a	225	235	287	290	
172	1PSG	a	272	275	276	278	
173	1PSG	a	272	275	283	285	I
174	1PSG	a	304	307	310	315	
175	2PFK	a	16	29	3	9	
176	2PFK	a	40	46	3	9	
177	2PFK	a	40	46	33	37	
178	2PFK	a	103	114	119	123	C
179	2PFK	a	227	238	163	168	F
180	2PFK	a	175	184	188	190	
181	2PFK	a	227	238	242	246	F
182	2PFK	a	258	276	282	287	
183	2PFK	a	296	299	290	295	D
184	3PGK	a	77	87	91	92	
185	3PGK	a	315	325	282	285	
186	1BP2	a	2	12	75	78	A
187	1BP2	a	2	12	81	84	
188	1BP2	a	90	107	81	84	
189	1PRC	h	12	30	10	11	
190	1PRC	h	232	248	193	197	F
191	1PCY	a	52	55	40	41	B
192	2PAB	a	75	82	67	73	H
193	2PAB	a	75	82	88	97	H
194	2PAB	b	75	82	67	73	H
195	2SGA	a	56	59	40	43	A
196	2SGA	a	56	59	103	108	
197	2SGA	a	231	237	208	219	M
198	2SGA	a	231	237	223	230	
199	2SGA	a	231	237	239	241	
200	3SGB	e	56	59	41	43	A
201	3SGB	e	56	59	84	93	
202	3SGB	e	56	59	103	108	
203	3SGB	e	231	237	208	219	M
204	3SGB	e	231	237	223	230	H
205	3SGB	e	231	237	240	241	
206	3SGB	i	34	43	16	17	A
207	2PRK	a	8	13	23	24	
208	2PRK	a	69	78	208	212	G
209	2PRK	a	223	239	127	131	K
210	2PRK	a	223	239	154	158	K
211	2PRK	a	223	239	178	183	G
212	2PRK	a	248	254	178	183	A
213	3RP2	a	231	242	206	215	
214	3RP2	a	231	242	226	230	
215	3RP2	b	231	242	206	215	
216	3RP2	b	231	242	226	230	
217	1RHD	a	108	117	94	97	
218	1RHD	a	108	117	123	126	
219	1RHD	a	129	136	123	126	C
220	7RSA	a	4	12	43	47	
221	7RSA	a	4	12	116	123	
222	7RSA	a	25	32	79	86	A
223	1RNS	s	4	12	43	47	
224	1RNS	s	4	12	117	123	
225	1RNS	a	25	32	79	86	A
226	2RNT	a	13	29	76	81	
227	2SNS	a	99	105	97	98	
228	1SBT	a	64	73	205	209	G
229	1SBT	a	104	116	101	102	
230	1SBT	a	220	237	121	124	
231	1SBT	a	220	237	148	152	
232	1SBT	a	260	263	198	201	I
233	1TEC	e	71	80	209	213	G
234	1TEC	e	112	124	129	132	E
235	1TEC	e	224	239	129	132	
236	1TEC	e	224	239	156	160	K
237	3TLN	a	137	151	113	115	
238	3TLN	a	301	312	253	255	
239	1TIM	a	47	54	59	64	
240	1TIM	a	178	196	205	208	
241	1TIM	a	216	221	228	231	
242	1TIM	b	45	54	60	63	E
243	1TIM	b	106	118	122	129	C
244	1TIM	b	178	195	205	208	
245	5TNC	a	75	105	72	74	
246	5TNC	a	115	123	113	114	
247	1TPP	a	165	171	180	183	A
248	5PTI	a	48	55	18	24	
249	5PTI	a	48	55	29	35	A
250	1WSY	a	103	107	275	279	
251	1WSY	a	137	145	149	151	E
252	1WSY	a	217	226	230	233	
253	1WSY	a	236	243	230	233	C
254	1WSY	b	167	178	105	109	
255	1WSY	b	146	151	155	158	
256	1WSY	b	235	241	226	230	
257	1WSY	b	349	364	226	230	F
258	1WSY	b	329	343	251	259	
259	1WSY	b	349	364	251	259	
260	1WSY	b	270	273	282	286	
261	1WSY	b	270	273	322	328	
262	1WSY	b	311	318	282	286	
263	1WSY	b	311	318	322	328	
264	1WSY	b	329	343	322	328	
265	1WSY	b	349	364	370	376	
266	2TS1	a	124	127	119	122	D
267	2TS1	a	164	184	186	192	
268	2STV	a	102	105	43	46	I
269	2STV	a	117	121	125	135	
270	4SBV	a	147	151	130	136	
271	4SBV	a	216	219	197	198	B
272	4SBV	b	147	150	130	136	
273	4SBV	b	168	172	237	250	
274	4SBV	b	216	219	197	198	B
275	4SBV	c	147	150	130	136	
276	4SBV	c	168	172	237	250	
277	4SBV	c	216	219	197	198	B
278	2TBV	a	192	195	240	244	
279	3XIA	a	302	305	12	13	

Table 6: 279 Pairs of Helix/β-Strand Super-Secondary Structures.
This table lists the protein ID from PDB, the chain identifier, the location of start and end of the helix (StartH, EndH) and of the β-strand (StartS, EndS) on the primary sequence. The numbers of helix/strand pairs in Figures 19 to 32 correspond to the column labelled "No.".

The work on PRL is similar to work of A. Sali and T. L. Blundell[24] on general topological equivalencies in protein structures. However, in their work different types of information are added numerically to give just one number between zero and one to express the degree of homology between two structures. In IPSA the different types of information are carried along *separately* through the whole analysis process and hence a detailed description of the classification generated can be automatically produced. Like the data of A. Sali and T. L. Blundell the PRL database can be used to find relationships between homologous proteins. In addition, however, the PRL database can also be applied to find relationships between *non-homologous* proteins at the super-secondary and domain level and to produce biochemically meaningful descriptions of any relationships between them.

An attempt to detect patterns in primary structure for the members of super-secondary structure classes led to the following results. For single helices significant patterns were found. This was expected since it is well known that certain amino acid residues occur more frequently in helices than others. Similarly, weak patterns for β-strands were detected. However, in none of the cases were the patterns specific enough to discriminate between members of different super-secondary classes. The interpretation of this finding is that 1) there may have been too few examples (in many cases only about ten) to detect class specific patterns and 2) one type of super-secondary structure may usually be encoded by several different sequences.

The basic methodology exemplified in IPSA: describe objects of interest; compute similarities; derive and analyse the resulting classification; and finally predict folding classes has been adopted in many research projects. Depending on the focus of interest concepts on different levels of detail can be used to characterise proteins. It is most important to choose a representation that is powerful enough to capture all essential features in the application. If the representation formalism cannot cope with certain concepts, e.g. structured attributes or approximate values, and it becomes obvious that those features are needed in the application one has go back and to start from the beginning. Hence, the representation issue should be given particular attention.

4.2.2. Symbolic Induction on Protein and DNA Sequences

Sequence analysis of DNA and protein sequences has become an increasingly important issue in molecular biology over the past few years mainly because of the following two reasons:

1. The entire genome of an organism contains the complete information about its structure and function *in toto* and of each of its molecular components. This fact should not be confused with the statement that the purpose of each DNA fragment or protein is entirely encoded in its own sequence. In many cases, the ultimate function or final structure of one molecular object is determined by the interaction with other objects (e.g. chaperones for protein folding; distantly located DNA sequences for transcription control or cis/trans regulation of expression).

24 A. Sali, T. L. Blundell, *Definition of General Topological Equivalencies in Protein Structure*, **Journal of Molecular Biology**, vol 213, pp. 327-336, 1990.

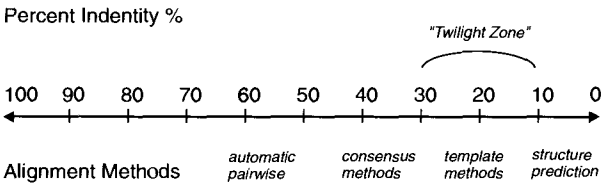


Figure 34: Protein Sequence Comparison.
On a scale of 0 to 100% residue identity four different regions of homology can be discerned that must be treated separately for sequence analysis.

2. DNA sequencing is nowadays routinely done on a large scale. In 1995, modern laboratories can measure 40 000 base pairs double-stranded DNA (40 kb) in two to four weeks. Another four to eight weeks are required to assemble, edit and proof-read the sequences. That means in about two to three months 40kb can be deciphered in one laboratory with one set of gel reading robots. The good news is that more sequences are becoming available ever faster. At the same time, however, this situation creates the demand for more intelligent tools for automatic interpretation of those large quantities of raw sequence data.

The current situation can be summarised as follows. Nearly every research project in molecular biology benefits from sequencing of genes (or proteins) and involves sequence analysis and comparison. At the same time, the sequencing community is approaching a state where entire genomes of selected organisms and whole families of proteins are available. This implies that more and more molecular biological issues can be treated and resolved on the level of sequences. As a consequence, increasing effort is put into automatic analysis and computer based interpretation of sequences. Figure 34 summarises several application areas of different methods for protein sequence comparison.

Sequence identity is measured as the percentage of residues that are identical after alignment. Two random sequences of length L, each composed of an alphabet of M letters can achieve about $1/M$ sequence identity:

$$\text{sequence identity} = \frac{1}{L} \cdot \sum_{i=1}^{L} \frac{1}{M} = \frac{1}{M}$$

If, in addition, gaps and insertions are allowed, the score for the alignment of two random protein sequences can reach about 20%. From 100% sequence identity down to about 50% *automatic pairwise alignment* is reasonable and safe. If the similarity coefficient drops below 50% more elaborate methods for sequence alignment are required because the best alignment is no longer obvious. If more than one sequence (e.g. members of a protein super family) is available the alignment can be improved by looking for identical stretches (*consensus sequences*) in several of the sequences. At around 30% to 10% sequence identity an alignment becomes very difficult. Recognisable conserved regions are no longer apparent. The approach for sequence comparison becomes *pattern based* where one tries to optimise the number and length of local patterns to give a reasonable global alignment. Below 10%

sequence identity no meaningful alignments can be achieved so far. For new sequences with identities less than 10% to any known protein sequence only *ab initio* methods for structure prediction are available.

In the following sections not the general problem of sequence alignment but one special application will be discussed: the analysis and prediction of *signal sequences in membrane proteins*. Nascent proteins contain signal tags that determine their ultimate destination inside or outside a cell. Eucaryotic cells can send proteins to internal sites such as lyzosomes, mitochondria, chloroplasts or into the nucleus. Protein synthesising ribosomes can move freely in cytosolic space unless they are directed to the so-called endoplasmatic reticulum (ER) by a signal sequence in the protein being synthesised. The ER is a different compartment that can connect to other membranes in the cell. Nascent polypeptide chains from membrane-bound ribosomes are translocated across the ER-membrane from where the newly synthesised proteins are sorted for their different destinations and different modes of transferral. Signal sequences are ancient. For example, bacteria and eucaryotic cells use rather similar sequences to send their proteins to the plasma membrane for secretion (see below).

Understanding protein targeting and signal sequences is important for a number of objectives.

1. Some diseases are a result of interrupted protein transport. In these cases, it would be helpful to understand the molecular cause for malfunction and from there to repair or complement the faulty mechanism.
2. For biotechnologically designed and produced proteins, signal sequences can be used to direct the protein outside the producing cell thereby facilitating recovery and isolation.
3. Signal sequences can be used to design proteins that are transported to a particular compartment in the body where they can then carry out a therapeutic task.

Currently, there are more than a hundred secretory proteins with known signal sequences. Although there is no well-defined consensus pattern for signal sequences biochemists have learned a number of common features about them:

1. The length of a signal peptide is between 13 and 36 amino acid residues.
2. The amino-terminal part of the signal contains at least one positively charged residue.
3. The centre of the signal is made of a stretch of about 10 – 15 hydrophobic residues.
4. The cleavage site is from its carboxy-terminal end preceded by a few residues that are more polar than the hydrophobic stretch.
5. Residues 1 and 3 from the cleavage site towards the N-terminus are usually small and neutral.

Two different approaches of machine learning will be shown to address the same problem: the recognition of sequence patterns and significant features to characterise signal sequences. As signal sequences are known to be reasonably constrained one expects a successful algorithm to produce results at least compatible to the

rules of human experts and, hopefully, in addition some rules that increase predictive accuracy and specificity or that describe previously unknown features.

Although both applications deal with peptide signal sequences the methods presented could be (and in part have already been) applied to the search for other motifs in DNA and protein sequences as e.g. secondary structure, promoters or splice sites.

4.2.2.1. Decision Trees over Regular Patterns

The authors S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara and S. Arikawa have developed a method for indexing decisions trees over regular patterns and applied this technique to the analysis and characterisation of transmembrane protein fragments and to the recognition of peptide signal sequences[25]. Their method is a derivation and extension of the classical ID3 approach introduced in chapter 2.1.3.

When looking at a large number of sequences that are known to belong to the same functional class or that exhibit similar biochemical properties it is interesting to ask if those sequences can be reduced to a common pattern. One way to proceed is to invent new classes of amino acid residues (or nucleotides) and to express the sequences in terms of those classes instead of individual residues. Both approaches, the one in this section and that of the following section, are based on the generalisation of amino acids into chemically and biologically related classes. S. Shimozono *et al* call this *indexing* of amino acid residues. If the 20 amino acids can be meaningfully divided into fewer subclasses these subclasses may then be used to define patterns that discriminate, for example, signal sequences from non-signal sequences.

S. Shimozono *et al* define decision trees over regular patterns as follows. Patterns consist of a sequence of variables, constants and strings. Strings in this case are sequences of residues or classes of residues. Variables denote a continuous fragment of residues. No label may occur in one pattern both as a constant *and* as a variable. A pattern is called a *regular pattern* if each variable occurs only once. For example, XABYCBAZCA is a regular pattern with the variables X, Y and Z and the constant symbols taken from the finite alphabet $\Sigma = \{A, B, C\}$. The pattern XABCX, however, is not a regular pattern. (Why?)

A decision tree over regular patterns is a binary tree with each internal node labelled with a regular pattern and each terminal node labelled with either 0 or 1. Figure 35 shows an example of a decision tree over regular patterns. This tree defines the language $\{B^m, A^n, B^l \mid m, n, l \geq 1\}$ over the alphabet $\Sigma = \{A, B\}$. This language, for example, contains the words BBBBBAAABBB, BAAABB, and BBBBAB but not BAABABB.

The approach of S. Shimozono *et al* is now to find a better indexing for the elements of an alphabet (representing amino acid residues or nucleotides) that opti-

25 S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, S. Arikawa, *Finding Alphabet Indexing for Decision Trees over Regular Patterns: An Approach to Bioinformatical Knowledge Acquisition*, **Research Institute of Fundamental Information Science Report RIFIS-TR-CS-60**, Kyushu University, email sin@ces.kyutech.ac.jp, Fukuoka, Japan, 1992.

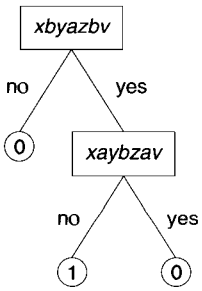


Figure 35: Decision Tree over Regular Patterns.
There are the variables x, y, and z and an alphabet with the constants a and b. Depending on whether a sequence can be matched by one of the regular patterns, "yes" and "no" determine the output of the decision tree. "1" means the input pattern belongs to the language defined by the decision tree, "0" means it doesn't.

mises the performance of a decision tree in classifying signal sequences from non-signal sequences. Their algorithm consists of the following two steps:

1. *Construction of an ID3 decision tree.* Shimozono *et al* extended the standard ID3 algorithm explained in chapter 2.1.3 to allow for a small number of inconsistent examples. The decision tree is expanded stepwise. The shortest regular pattern of the form XAY is generated that minimizes the entropy increase for splitting signal sequences from non-signal sequences. The entropy function $E(\pi, P, N)$ for a regular pattern π with positive (P) and negative (N) examples is:

$$\begin{aligned} E(\pi,P,N) &= \frac{p_1+n_1}{|P|+|N|}I(p_1+n_1)+\frac{p_0+n_0}{|P|+|N|}I(p_0+n_0) \quad \text{with} \\ I(x,y) &= \begin{cases} 0 & \text{if } x=0 \text{ or } y=0 \\ -\frac{x}{x+y}\log\frac{x}{x+y}-\frac{y}{x+y}\log\frac{y}{x+y} & \text{otherwise.} \end{cases} \end{aligned}$$

   p_1 (n_1) is the number of positive (negative) examples that match pattern π, p_0 (n_0) is the number of positive (negative) examples that do not match pattern π.
2. *Finding a better indexing.* First, a randomly selected indexing scheme is used to build a decision tree for a small subset of all positive and negative examples (here 10 of each). Next, similar indexing schemes are examined until no further improvement can be achieved. Then a new subset of examples is selected and the whole procedure repeated until the locally optimal indexing schemes converge. Similar indexing schemes are defined to be those that have exactly one symbol indexed differently. The authors also suggest that other optimization methods could be used to generate alternative indexing schemes (e.g. a genetic algorithm).

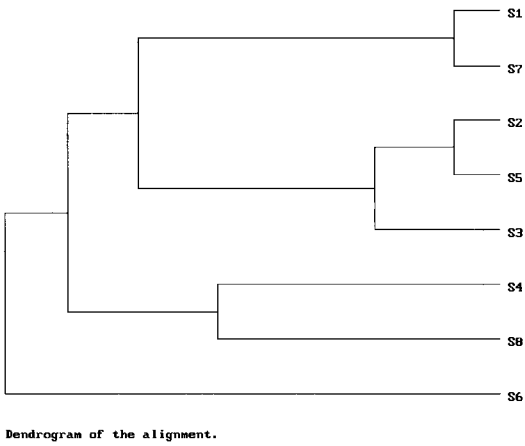
```
S1: Cytochrome c2 precursor of Rhodopseudomonas viridis
 M R K L V F G L F V L A A S V A P A A A

S2: Hepatocyte growth factor receptor precursor of human
 M K A P A V L A P G I L V L L F T L V Q R S N G

S3: Epidermal growth factor receptor precursor of human
 M R P S G T A G A A L L A L L A A L C P A S R A

S4: Protein-tyrosine kinase (EC 2.7.1.112) of rat
 M I I M E L A A W C R W G F L L A L L

S5: Nerve growth factor receptor precursor, high affinity of human
 M L R G G R R G Q L G W H S W A A G P G S L L A W L I L A S A G

S6: Triacylglycerol lipase (EC 3.1.1.3) precursor, pancreatic of dog
 M V S I W T I A L F L L G A A K A

S7: Lipoprotein lipase (EC 3.1.1.34) precursor of human
 M E S K A L L V L T L A V W L Q S L T A S R G G V A A

S8: Phospholipase A2 (EC 3.1.1.4) precursor of dog
 M K F L V L A A L L T V A A A
```

Example 1: Eight Signal Sequences from Different Organisms.

```
S1    MRK---LV-FGLFV--------LAASVAPAAA
S2    MKAPA--------VLAPGILVLLFTLVQRSNG
S3    MRPSGTAG--------AALLALLAALCPASRA
S4    MI---IMELAAWCRWG----------FLLALL
S5    MLRGGRRGQLGWHSWAAGPGSLLAWLILASAG
S6    MVS----------IWTIALFLL-----GAAKA
S7    MESKALLV-LTLAVWLQS----LTASRGGVAA
S8    MK---FLVLAAL--------------LTVAAA
```

Example 2: Alignment of Eight Signal Sequences.
The N-terminal methionine was removed for the alignment and later added again. Otherwise, the common methionine would suggest a strongly conserved residue at position 1 and the algorithm would deceptively be guided to search for more similarities in the vicinity of the N-terminus.

Figures 36 and 37 show the result of experiments with the decision tree indexing algorithm on signal sequences. One-letter labels in the graphs denote amino acid residues; numbers represent regular patterns, e.g. "0201" is shorthand for the pattern "X0201Y" which means that the sequence in question must somewhere contain a subsequence with the amino acid indices 0-2-0-1 in that order. The final (optimal) index residue-mapping of each experiment appears above the graphs. For comparison, decision trees derived without indexing are presented. A leaf label of "1" means "*sequence is a signal peptide*" and "0" means "*sequence is not a signal peptide*". The numbers below each leaf indicate percent of positive (P) and negative (N) examples covered by that node, i.e. positive or negative predictions. The to-



Example 3: Similarity Dendrogram of Eight Signal Sequences.
The dendrogram is derived from the sequences in Example 1.

Amino Acid Residue	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
Index	2	2	1	1	2	2	2	2	0	2	2	0	2	0	2	2	2	2	2	0

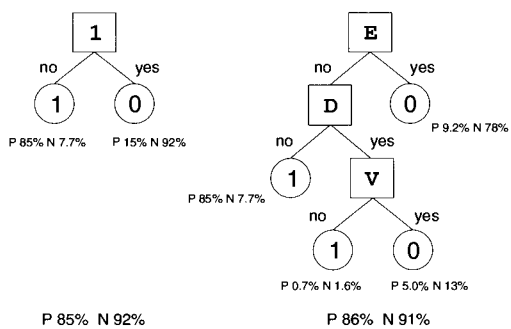


Figure 36: Signal Peptides of Bacterial Sequences.
There were 436 positive and 4635 negative examples of signal sequences. See main text for details.

tal coverage of a whole decision tree is given below the tree. As can be seen, the predictive value of decision trees derived with indexing is as good as or better than that of the trees without indexing. The indexed tree for bacterial signal sequences is also much simpler than the corresponding tree without indexing.

The only prokaryotic sequence in Example 1 (S1) would be correctly predicted to be a signal sequence by both trees in Figure 36. The non-indexed tree of Figure 36 has an additional test on valine and is therefore a subset of the indexed version.

Figure 37 shows two decision trees for primate signal sequences. Here, the algorithm used three indices as opposed to only two for procaryotes. The trees for eucaryotes are more refined than those for the bacterial signal peptides. The tree with indexing is more general than the tree without indexing. Use of indexing im-

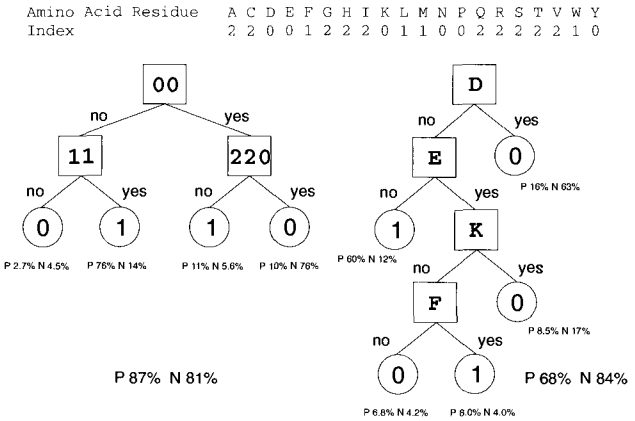


Figure 37: Signal Peptides of Primate Sequences.
There were 735 positive and 1345 negative examples of signal sequences. See main text for details.

Sequence	Kingdom	Prediction with Indexing Signal Peptide Probability	Prediction without Indexing Signal Peptide Probability
S1	procar.	92%	92%
S2	eucar.	84%	83%
S3	eucar.	84%	83%
S4	eucar.	84%	67%
S5	eucar.	84%	83%
S6	eucar.	84%	83%
S7	eucar.	84%	35%
S8	eucar.	84%	83%
Mean		85%	76%

Table 9: Summary of Prediction Performance.
The eight signal peptide sequences from Example 1 were analysed as to whether a decision tree would predict them to be signal or non-signal encoding. The indexed decision trees perform as well as or better than their non-indexed counter parts. Note that this selection of signal sequences is statistically not at all representative.

proves the overall prediction of signal sequences by 19% compared to the decision tree without indexing but at the same time decreases the prediction accuracy for non-signal peptide sequences by 3%.

Table 9 summarises the prediction probabilities of the four decision trees on the above (non-representative) selection of eight signal sequences in Example 1. The indexed versions are always superior to the non-indexed versions except for the bacterial sequence where both versions achieve identical results.

The pattern learning approach of S. Shimozono *et al* has the advantage of being fully automatic. One limitation, however, is that in general this method can only be

used to decide whether a particular sequence fragment is likely to contain a signal sequence or not but not to determine its location. The approach presented in the following section can provide more information in this respect.

4.2.2.2. Searching Signal Peptide Patterns in Predicate Logic Hypothesis Space

Promis[26] is a machine learning program that uses Prolog representations of amino acid sequences to detect sequence patterns. Originally it was applied to the prediction of three types of secondary structure. In Promis, primary structure is represented as a sequence of types (classes) of amino acids. The classification of amino acids is based on work of W. Taylor[27]. Part of the classification in Prolog syntax is listed below:

```
class (all, [a,e,q,d,n,l,g,k,s,v,r,t,p,i,m,f,y,c,w,h]).
class (hydrophobic, [h,w,y,f,m,l,i,v,c,a,g,t,k]).
class (small, [p,v,c,a,g,t,s,n,d]).
class (positive, [r,k,h]).
class (charged, [d,e,r,k,h]).
class (polar, [s,n,d,e,q,r,k,h,w,t]).
class (tiny, [a,g,s]).
class (aliphatic, [l,i,v]).
class (aromatic, [h,w,y,f]).
class (negative, [d,e]).
class (charged_and_not_h, [d,e,r,k]).
class (hydrophilic_and_not_positive, [s,n,d,e,q]).
class (small_p_or_polar_and_not_aromatic, [p,v,c,a,g,t,s,n,d,e,q,r,k]).
```

Altogether, 71 classes are specified. The class identifiers (ALL, HYDROPHOBIC, SMALL, POSITIVE, etc.) can be appended in a list to form a generalised sequence pattern. A rule in Promis connects a pattern and a class. In the original application Promis learned to associate patterns composed of those 71 amino acid residue classes with the type of secondary structure of the pattern. Here, there are only two classes to learn and predict: signal peptides and non-signal peptides. For example, the decision tree with indexing for bacterial signal sequences of S. Shimozono *et al* in the last section could be written as a rule with a generalised sequence pattern as follows: "[NEGATIVE, NEGATIVE] ⇒ BACTERIAL SIGNAL SEQUENCE".

Promis implements a "*generate and test hill climbing beam search*" algorithm which cycles through the following steps.

1. Initialise the current set of rules with the most general rule. For the signal peptide application this rule would be: "Any residue determines the presence of a targeting signal", written as: "[ALL] ⇒ SIGNAL". Certainly, this rules covers all positive examples but also predicts any non-signal sequence to be a signal sequence. The goal is now to arrive at one or more rule patterns that best discriminate signal sequences from non-signal sequences.

26 R. D. King, *A Machine Learning Approach to the Problem of Predicting a Protein's Secondary Structure from its Primary Structure (PROMIS)*, **Ph.D. Thesis**, University of Strathclyde, Department of Computer Science, Strathclyde, UK, 1988.

27 W. R. Taylor, *The Classification of Amino Acid Conservation*, **Journal of Theoretical Biology**, 119, pp. 205-221, 1986.

2. Generalisation and specialisation operators are applied to generate new rules. Specialisation operators are:
   - *Expansion of a pattern.* One more residue is added on one side of the pattern, e.g. the pattern [POLAR, SMALL] becomes [POLAR, SMALL, ALL].
   - *Specialisation of a residue*, e.g. the pattern [POLAR, ALL] becomes [POLAR, SMALL].

   Generalisation operators are:
   - *Shortening of a pattern* by removing one residue at the end.
   - *Generalising a pattern* by replacing a more specific residue class by a more general one.
3. After a number of new rules have been generated their performance is measured. The quality evaluation function for rules is the difference of correctly and incorrectly predicted residues divided by the sum of all residues in the learning set. This measure combines the desire for high coverage and high accuracy.
4. The n best rules are selected for the next round of modification. n is also called the beam size.
5. Continue with step 2 until no further improvement in prediction of the current set of rules is seen.
6. The final rules must be tested on a representative set of examples that were not included in the training set.

Test and training data were taken from the SigPep Database[28]. Three experiments shall be presented here[29]. Promis was adapted to discriminate between signal and non-signal sequences. This requires a slight re-interpretation of the original conventions used in Promis: the original class assignments were "A" for α-helices, "B" for β-strands and "T" for coils. For signal peptide recognition class "A" was interpreted as signal encoding and classes "B" and "T" were both non-signal encoding. Table 10 summarises the number of sequences in the training and test sets of membrane proteins.

Training and test protein sequences are stored in three types of Prolog clauses. The NAME clause specifies the class of the sequence fragment. ALPHA stands here for signal encoding and BETA and TURN for non-signal encoding (see above).

```
name(mz1,1,23,alpha,protein).
name(mz2,1,22,alpha,protein).
name(mz3,1,21,alpha,protein).
name(mz4,1,24,alpha,protein).
name(mz5,1,25,alpha,protein).
name(mz6,1,20,alpha,protein).
...
name(ceb11,1,30,beta,protein).
name(ceb12,1,30,beta,protein).
```

28 G. v Heijne, **Protein Data Sequence Analysis**, vol 1, pp. 41 – 42, 1987.

29 G. Schneider, diploma thesis, Department of Physics, Biophysics Group, Free University Berlin, Germany, 1991. The author of this book acted as instructor for those parts of the diploma thesis presented here.

Type of Sequence	Number of Sequences in Training Set	Number of Sequences in Test Set	Sum
Eucaryotes	74 (63 pos., 11 neg.)	38 (32 pos., 6 neg.)	112
Eubacteria	57 (46 pos., 11 neg.)	25 (19 pos., 6 neg.)	82
Mitochondria	31 (20 pos., 11 neg.)	17 (11 pos., 6 neg.)	48

Table 10: Number of Sequences for Signal Sequence Recognition. Sequences marked "pos." refer to true signal sequences; those marked "neg." are 30 residue N-terminal fragments from proteins without a signal sequence.

```
name(cmz1,1,30,turn,protein).
name(cmz4,1,30,turn,protein).
name(cmz5,1,30,turn,protein).
```

The PROTEIN clause stores the actual sequence of a fragment.

```
protein(mz1,23,[d,y,y,r,k,y,a,a,v,i,l,a,i,l,s,l,f,l,q,i,l,h,s]).
protein(mz2,22,[d,l,t,s,p,l,c,f,s,i,l,l,v,l,c,i,f,i,q,s,s,a]).
protein(mz3,21,[k,p,i,q,k,l,l,a,g,l,i,l,l,t,s,c,v,e,g,c,s]).
protein(mz4,24,[i,p,a,k,d,m,a,k,v,m,i,v,m,l,a,i,c,f,l,t,k,s,d,g]).
protein(mz5,25,[a,t,g,s,r,t,s,l,l,l,a,f,g,l,l,c,l,p,w,l,q,e,g,s,a]).
protein(mz6,20,[a,r,s,s,l,f,t,f,l,c,l,a,v,f,i,n,g,c,l,s]).
...
protein(ceb11,30,[a,d,k,e,l,k,f,l,v,v,d,d,f,s,t,m,r,r,i,v,r,n,l,l,k,
                  e,l,g,f,n]).
protein(ceb12,30,[m,q,p,s,i,k,p,a,d,e,h,s,a,g,d,i,i,a,r,i,g,s,l,t,r,
                  m,l,r,d,s]).
protein(cmz1,30,[a,a,s,i,f,a,a,v,p,r,a,p,p,v,a,v,f,k,l,t,a,d,f,r,e,d,
                 g,d,s,r]).
protein(cmz4,30,[a,p,a,e,i,l,n,g,k,e,i,s,a,q,i,r,a,r,l,k,n,q,v,t,q,l,
                 k,e,q,v]).
protein(cmz5,30,[s,t,t,g,q,i,i,r,c,k,a,a,v,a,w,e,a,g,k,p,l,v,i,e,e,v,
                 e,v,a,p]).
```

The assignment of which region on the sequence actually belongs to the signal recognition fragment is determined in ACTUAL SECONDARY STRUCTURE clauses. Again, this application does not work on secondary structure but simply reinterprets the original class assignments in terms of signal encoding ("A") and non-signal encoding ("B" and "T").

```
actual_secondary_structure(mz1,23,[a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,
                                   a,a,a,a,a,a]).
actual_secondary_structure(mz2,22,[a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,
                                   a,a,a,a,a]).
actual_secondary_structure(mz3,21,[a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,
                                   a,a,a,a]).
actual_secondary_structure(mz4,24,[a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,
                                   a,a,a,a,a,a,a,a]).
actual_secondary_structure(mz5,25,[a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,
                                   a,a,a,a,a,a,a,a]).
actual_secondary_structure(mz6,20,[a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,
                                   a,a,a]).
```

```
...
actual_secondary_structure(ceb11,30,[b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,
                                     b,b,b,b,b,b,b,b,b,b,b,b,b,b]).
actual_secondary_structure(ceb12,30,[b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,
                                     b,b,b,b,b,b,b,b,b,b,b,b,b,b]).
actual_secondary_structure(cmz1,30,[t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,
                                    t,t,t,t,t,t,t,t,t,t,t,t,t,t]).
actual_secondary_structure(cmz4,30,[t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,
                                    t,t,t,t,t,t,t,t,t,t,t,t,t,t]).
actual_secondary_structure(cmz5,30,[t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,
                                    t,t,t,t,t,t,t,t,t,t,t,t,t,t]).
```

Now, Promis can be set to work starting with the most general rule for signal sequences:

```
rule([[[-10000,0,0,0,0,0,0,0,0],[all]],[i,0,a,positive]]).
```

This rule has an initial performance value of -10000. The performance value is set arbitrarily very low so that any other reasonable rule can surpass it. The following eight zeros indicate that no statistics have been calculated yet for this rule. The pattern of the rule is "[ALL]" which is the most general pattern in Promis. In the third sublist the letter "i" indicates that specialisation of the rule pattern is allowed. The letter "A" refers to the class of sequences for which patterns should be learned (signal sequences in this case).

A run of Promis for eubacterial sequences produced the following rule:

```
rule (
[all,
 hydrophobic_or_small,
 all,
 all_minus_k_p,
 hydrophobic_or_small_and_not_p,
 hydrophobic,
 hydrophobic_or_small,
 neutral,
 neutral,
 hydrophobic_or_small,
 very_hydrophobic_or_small_and_not_p_or_k,
 all_minus_p,
 hydrophobic_or_small_and_not_p]).
```

This rule should be read as follows. The coverage of this rule is 99% which means that in the learning phase 99% of the signal residues were predicted to occur in signal sequences. The remaining 1% were erroneously predicted not to be part of signal sequences (false negatives). The rule pattern itself says that wherever a sequence occurs that fits the description starting with the amino acid classes ALL, HYDROPHOBIC_OR_SMALL, ALL, ALL_MINUS_K_P, etc. then this fragment is predicted to be part of a signal sequence.

The corresponding rule for eucaryotes with a coverage of 97% is the following:

```
rule (
[hydrophobic_or_small_and_not_p,
 all,
 aromatic_or_very_hydrophobic,
 all_minus_k_p,
```

Type of Sequence	Coverage	Prediction Training Set	Prediction Test Set
Eucaryotes	97%	89%	78%
Eubacteria	99%	92%	83%
Mitochondria	87%	81%	68%

Table 11: Coverage and Prediction Accuracy of Signal Peptide Rules.
"Coverage" is the percentage of actual signal sequence residues predicted by that rule on the training set. "Prediction" accuracy is defined as the ratio of correct positive predictions to all (including false positive) predictions of that rule, expressed in percent.

```
 hydrophobic_or_small,
 hydrophobic_or_small,
 neutral,
 all,
 neutral,
 all_minus_k_p,
 hydrophobic_or_p]).
```

Mitochondria prepeptide sequences produced to the following rule with coverage 87%:

```
rule (
[all,
 all,
 all,
 aliphatic_or_small_and_not_hydropilic,
 all,
 hydrogen_bond_doners,
 hydrophobic_or_small,
 all_minus_p,
 all,
 small_or_polar,
 all_minus_p,
 all]).
```

Table 11 summarises the coverage and prediction performance of the three rules. All three rules show a high coverage, i.e. most of the residues of the known signal sequences are covered by these rules. Prediction accuracy is in general higher for the training set than for the test set. This indicates an overfitting of the rules for the training examples. Eubacterial sequences are simpler and more accurately predicted than eucaryotic or mitochondrial signal sequences.

In all experiments, the N-terminal methionine was removed for the same reason as mentioned in the previous section. As a result, the location where a pattern matches a signal sequence varies from rule to rule. Patterns can occur close to the N-terminal residue or further downstream. They help to characterise the actual signal rather than just predicting whether a particular sequence fragment is likely to contain a signal sequence or not. The occurrence of signal patterns shows that the relative distribution of chemo-physical properties *along the primary structure* is relevant for signal particle recognition. This finding is consistent with the results of

S. Shimozono's *et al* approach. The strong tendency to exclude hydrophilic residues from signal patterns (Figures 36, 37) can be found in all three signal rule patterns which are dominated by hydrophobic residues. These hydrophobic stretches are in perfect agreement with classical knowledge about signal sequences stated at the beginning of this section. However, the one positively charged residue which is known to occur in signal sequences was not discovered by Promis in these runs.

5. Evolutionary Computation

Evolutionary Computation is, like neural networks, an example *par excellence* of an information processing paradigm that was originally developed and exhibited by nature and later discovered by man who subsequently transformed the general principle into computational algorithms to be put to work in computers. Nature makes use of the principle of genetic heritage and evolution in an impressive way. Application of the simple concept of performance based reproduction of individuals ("survival of the fittest") led to the rise of well adapted organisms that can endure in a potentially adverse environment. Mutually beneficial interdependencies, co-operation and even apparently altruistic behaviour can emerge solely by evolution. The investigation of those phenomena is part of research in artificial life but cannot be dealt with in this book.

Evolutionary computation comprises the four main areas of *genetic algorithms*[1], *evolution strategies*[2], *genetic programming*[3] and *simulated annealing*[4]. Genetic algorithms and evolution strategies emerged at about the same time in the United States of America and Germany. Both techniques model the natural evolution process in order to optimise either a fitness function (evolution strategies) or the effort of generating subsequent, well-adapted individuals in successive generations (genetic algorithms). Evolution strategies in their original form were basically stochastic hill-climbing algorithms and used for optimisation of complex, multi-parameter objective functions that in practise cannot be treated analytically. Genetic algorithms in their original form were not primarily designed for function optimisation but rather to demonstrate the efficiency of genetic crossover in assembling successful candidates over complicated search spaces. Genetic programming takes the idea of solving an optimisation problem by evolution of potential candidates one step further in that not only the parameters of a problem but also the algorithm for problem solving is subject to evolutionary change. Simulated Annealing is mathematically similar to evolution strategies. It was originally derived from a physical model of crystallisation. Only two individuals compete for the highest rank according to a fitness function and the decision about accepting suboptimal candidates is controlled stochastically.

All methods presented in this chapter are heuristic, i.e. they contain a random component. As a consequence (and in contrast to deterministic methods) it can

1 J. H. Holland, *Genetic algorithms and the optimal allocations of trials*, **SIAM Journal of Computing**, vol 2, no 2, pp. 88 – 105, 1973.
2 I. Rechenberg, *Bionik, Evolution und Optimierung*, **Naturwissenschaftliche Rundschau**, vol 26, pp. 465 – 472, 1973.
3 J. Koza, **Genetic Programming**, MIT Press, 1993.
4 S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, *Optimization by Simulated Annealing*, **Science**, vol 220, no 4598, pp. 671-680, 1983.

never be guaranteed that the algorithm will find an optimal solution or even any solution at all. Evolutionary algorithms are therefore used preferably for applications where deterministic or analytic methods fail, for example because the underlying mathematical model is not well defined or the search space is too large for systematic, complete search (n.-p. completeness). Another application area for evolutionary algorithms that is rapidly growing is the simulation of living systems starting with single cells and proceeding to organisms, societies or even whole economic systems[5,6]. The goal of artificial life is not primarily to model biological life as accurately as possible but to investigate how our life or other, presumably different forms of life could have emerged from non-living components.

Work with evolutionary algorithms bears the potential for a philosophically and epistemologically interesting recursion. At the beginning, evolution emerged spontaneously in nature. Next, man discovers the principle of evolution and acquires knowledge of its mathematical properties. He ("re-") defines genetic algorithms for computers. To complete the recursive cycle, computational genetic algorithms can be applied to the very objects (DNA, proteins) of which they had been derived in the beginning. A practical example of such a meta-recursive application will be given below in the sections on protein folding. Figure 1 illustrates this interplay of natural and simulated evolution.

5.1. Methodology

5.1.1. Genetic Algorithms

The so-called genetic algorithm[7] is a heuristic method that operates on pieces of information like nature does on genes in the course of evolution. Individuals are represented by a linear string of letters of an alphabet (in nature nucleotides, in genetic algorithms bits, characters, strings, numbers or other data structures) and they are allowed to *mutate*, *crossover* and *reproduce*. All individuals of one generation are evaluated by a *fitness function*. Depending on the generation replacement mode a subset of parents and offspring enters the next reproduction cycle. After a number of iterations the population consists of individuals that are well adapted in terms of the fitness function. Although this setting is reminiscent of a classical function optimisation problem genetic algorithms were originally designed to demonstrate the benefit of genetic crossover in an evolutionary scenario, not for function optimisation. It cannot be proven that the individuals of a final generation contain an optimal solution for the objective encoded in the fitness function but it can be

5 T. Jones, S. Forrest, **An Introduction to SFI Echo**, November 1993, Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe NM 87501, email terry@santafe.edu, forrest@cs.unm.edu. World Wide Web Server at ftp://alife.santafe.edu/pub/SOFTWARE.

6 J. H. Holland, *Echoing Emergence: Objectives, Rough Definitions and Speculations for Echo-class Models*, in **Integrative Themes**, (G. Cowan, D. Pines, D. Melzner, Eds.), Santa Fe Institute Studies in the Science of Complexity, Proc. Vol XIX, Reading, MA, Addison-Wesley, 1993.

7 J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd Ed., **MIT Press**, 1992.

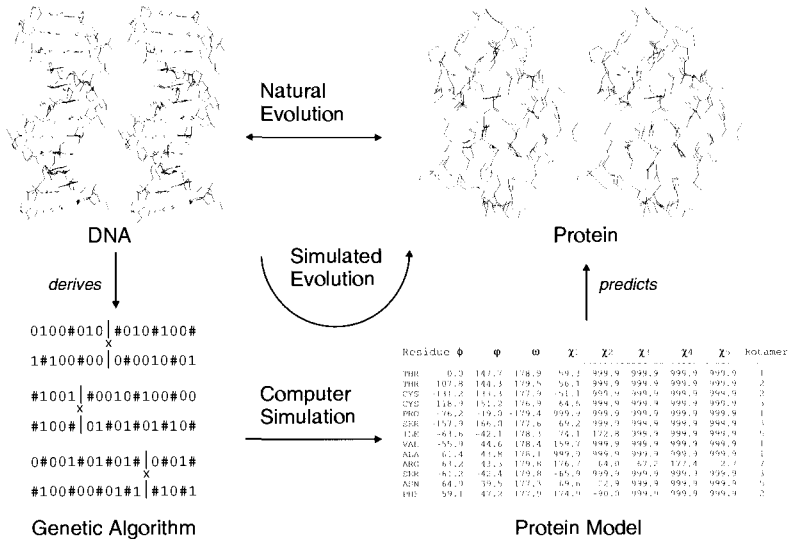


Figure 1: Interplay of Natural and Simulated Evolution.
Respond to nature's challenges by her own means. The principle of evolution can be used to model structures and to simulate biomolecular reactions.

shown mathematically that the genetic algorithm optimises the effort of testing and producing new individuals if their representation permits development of *building blocks* (also called *schemata*). In that case, the genetic algorithm is driven by an implicit parallelism and generates significantly more successful progeny than random search. In a number of applications where the search space was too large for other heuristic methods or too complex for analytic treatment genetic algorithms produced favourable results[8].

The basic outline of a genetic algorithm is as follows:

1. *Initialise a population of individuals.* This can be done either randomly or with domain specific background knowledge to start the search with promising seed individuals. Where available the latter is always recommended.

   - *Individuals* are represented as a string of bits. This is not a restriction for the type of problem because other data types (numbers, strings, structures) can also be encoded as bit strings.
   - A *fitness function* must be defined that takes as input an individual and returns a number (or a vector) that can be used as a measure of the quality (fitness) of that individual.
   - The application should be formulated such that the desired solution to the problem coincides with the most successful individual according to the fitness function.

8 L. Davis, (ed.) **Handbook of Genetic Algorithms**, Van Nostrand Reinhold, New York, 1991.

2. *Evaluate* all individuals of the initial population.
3. *Generate* new individuals. The reproduction probability for an individual is proportional to its relative fitness within the current generation. Reproduction involves domain specific genetic operators (Figure 2). Operations to produce new individuals are:
   - *Mutation.* Substitute one or more bits of an individual randomly by a new value (0 or 1).
   - *Variation.* Change the bits in a way that the number encoded by them is slightly incremented or decremented.
   - *Crossover*. Exchange parts (single bits or bit strings) of one individual with the corresponding parts of another individual. Originally, only one-point crossover was performed but theoretically one can process up to $L - 1$ different crossover sites (with L as the length of the individual). For one-point crossover, two individuals are aligned and one location on their strings is randomly chosen as the crossover site. Now the parts from the beginning of the individuals to the crossover site are exchanged between the two. The resulting hybrid individuals are taken as the new offspring individuals.
4. *Select* individuals for the new parent generation.
   - In the original genetic algorithm all offspring were selected while all parents were discarded. This is motivated by the biological model and is called *total generation replacement.*
   - More recent variations of generation replacement compare the original parent individuals and the offspring which are then ranked by their fitness values. Only the n best individuals (n is the population size, i.e. the number of individuals in one generation) are taken into the next generation. This method is called *elitist generation replacement.* It guarantees that good individuals are not lost during a run. With total generation replacement good individuals may "die out" because they produce only offspring inferior in terms of the fitness function. Another variant is *steady state replacement.* There, two individuals are randomly selected from the current population. The genetic operators are applied and the offspring replace the parents in the population. Steady state replacement often converges sooner because on average it requires fewer fitness evaluations than elitist or total generation replacement.
5. Go back to step 2 until either a desired fitness value is reached or until a predefined number of iterations is performed.

The mathematical foundation of genetic algorithms is the *schemata theorem* of J. H. Holland. It makes a statement about the propagation of *schemata* (or *building blocks*) within all individuals of one generation. A schema is implicitly contained in an individual. Like individuals, schemata consist of bit strings (1, 0) and can be as long as the individual itself. In addition, schemata may contain "don't care" positions where it is not specified whether the bit is 1 or 0, i.e. schemata H_i are made from the alphabet $\{1, 0, \#\}$. In other words, a schema is a generalisation of (parts of) an individual. For example, the individuals:

`1001001001000001111101000101` and

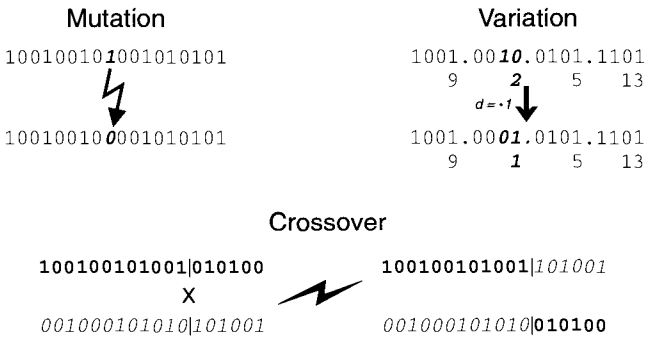


Figure 2: Genetic Operators for the Genetic Algorithm.
Mutation exchanges one single bit. Variation modifies the encoded value by a small increment (or decrement). Crossover (single-point) exchanges a contiguous fragment of an individual. Analogously, more than one crossover point can be selected and only the fragments between those positions exchanged (two-point crossover for two crossover points; uniform crossover for as many crossover sites as positions in the individual).

```
0100100101000001111111010101
```

can be summarised by the schema:

```
##0##0##01000001111#10#0101
```

where all identical positions are retained and differing positions marked with a "#" which stands for "don't care". The *length* $\delta(H)$ of the above schema is 25 which is the distance from the first to the last fixed symbol (1 or 0 but not #). The total number of different schemata of length l over an alphabet of cardinality k is $(k + 1)^l$. Because each string of length l contains 2^l schemata, with n individuals in one generation there are between 2^l and $n2^l$ schemata in the population (depending on the similarity of the n individuals). The *order* of a schema $o(H)$ is the number of fixed positions (1 or 0 but not #).

Let[9] $s(H, t)$ be the number of occurrences of a particular schema H in a population of n individuals at time t. The bit string A_i of individual i then gets selected for reproduction with probability p_i:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j},$$

where $f_{i,j}$ is the fitness value of the i, j-th individual. The expected number of occurrences of schema H at time t+1 is:

$$s(H,t+1) = s(H,t) \cdot n \cdot \frac{\bar{f}(H)}{\sum_{i=1}^{n} f_i}$$

9 D. E. Goldberg, **Genetic Algorithms in Search, Optimization & Machine Learning,** Addison-Wesley, 1989.

with $\bar{f}(H)$ as the average fitness of all individuals (strings A_i) that contain H. Crossover and mutation operators can destroy schemata during reproduction. The longer a single individual, the smaller the probability that a schema H will be involved in a crossover event. The longer a schema, i.e. the larger $\delta(H)$, the more likely is its destruction through recombination with another individual. Hence, for *crossover* the lower bound for the survival probability of a schema H is:

$$p_s \geq 1 - \frac{\delta(H)}{L-1}$$

with L as the length of one whole individual. If we perform crossover stochastically at a frequency p_c the survival probability p_s becomes:

$$p_s \geq 1 - p_c \frac{\delta(H)}{L-1}$$

Summarising the effects of independent crossover and reproduction we arrive at the following equation for the expected occurrence of a schema H at time t+1:

$$s(H,t+1) = s(H,t) \cdot n \cdot \frac{\bar{f}(H)}{\sum_{i=1}^{n} f_i} \cdot \left(1 - p_c \frac{\delta(H)}{L-1}\right).$$

This equation tells us that schemata increase over time proportional to their relative fitness and inversely proportional to their length. – *Mutation* can effect a schema H at each of its $o(H)$ fixed positions with mutation probability p_m. Survival of a single constant position in a schema is then $p_s = 1 - p_m$ and survival of the entire schema:

$$p_s = (1 - p_m)^{o(H)}$$

which for small p_m can be approximated by $p_s \approx 1 - o(H) \cdot p_m$. Summarising the effects of independent mutation, crossover and variation we get the following formula for the expected count of a schema H:

$$s(H,t+1) = s(H,t) \cdot n \cdot \frac{\bar{f}(H)}{\sum_{i=1}^{n} f_i} \cdot \left(1 - p_c \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right).$$

Assuming a schema H could always outperform other schemata by a fraction b of the total mean fitness then this equation can be rewritten as:

$$\begin{aligned} s(H,t+1) &= s(H,t) \cdot \frac{\frac{1}{n}\sum_{i=1}^{n} f_i + b\frac{1}{n}\sum_{i=1}^{n} f_i}{\frac{1}{n}\sum_{i=1}^{n} f_i} \cdot \left(1 - p_c \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right) \\ &= s(H,t) \cdot (1+b) \cdot \left(1 - p_c \frac{\delta(H)}{L-1} - o(H) \cdot p_m\right). \end{aligned}$$

This equation is of the form $f_j = f_0 \cdot (1+b)^j \cdot g(p_c, p_m, L, \delta(H))$ which says that the number of schemata better than average will *exponentially increase over time*. Effectively, many different schemata are sampled implicitly in parallel and good schemata will persist and grow. This is the basic rationale behind the genetic algorithm. It

Individual	**Bit String**	**Integer Value**	**Fitness** $f(i) = i^2$	**Reproduction Probability** $\frac{f(i)}{\sum f}$	**Expected Count** $\frac{f_i}{\bar{f}}$	**Actual Count (Roulette wheel)**
1	01010	10	100	8.2 %	0.33	1
2	10101	21	441	36.1 %	1.45	1
3	00010	2	4	0.3 %	0.01	0
4	11010	26	676	55.4 %	2.22	2
Sum			1221	100.0 %	4.01	4
Average			305.25			
Max			676			
Schemata	**Pattern**	**in Individual**	**Average Schema Fitness**			
H_1	00###	3	4			
H_2	1####	2,4	558.5			
H_3	#1#1#	1	100			

Table 1: Genetic Algorithm at Work (Part I).
Continued in Table 2. See main text for explanation.

is suggested that if the (linear) representation of a problem allows the formation of schemata then the genetic algorithm can efficiently produce individuals that continuously improve in terms of the fitness function.

Let us examine the performance of the genetic algorithm on a simple application which is the search for the largest square product of a 5-bit integer. Table 1 shows four initial individuals that were randomly generated. The bit strings of the individuals are decoded to unsigned integer values. The fitness function f $(i) = i^2$ is used to assign a fitness value to each individual. Depending on their relative fitness values the reproduction probability (between 0% and 100%) for each individual is calculated and converted into the number of expected successors. Then the so-called *roulette wheel* algorithm is used to perform a stochastic selection based on the reproduction probability. Three particular schemata and their occurence and distribution over the four individuals is monitored.

Table 2 shows the situation after reproduction. The individuals selected for reproduction have been replicated according to their relative fitness. Crossover sites and mating partners have been assigned randomly. To keep this example simple mutation is not used here. After performing crossover the new fitness values of the individuals in the new population are calculated. The performance of the three schemata H_1, H_2, and H_3 is also shown. Schema H_1 is of low fitness because it implies that the decoded integer is smaller than 8. Therefore, this schema gets only a small chance for reproduction. Actually, H_1 dies out as its only parent (the original individual #3) does not get selected for reproduction. Schemata H_2 and H_3 both have a reasonable chance for reproduction and are subsequently found in the new

Mating Pool after Reproduction with Crossover Site	**Mating Partner**	**Crossover Site**	**New Population**	**Integer Value**	**Fitness Value** $f(i) = i^2$
010|10	3	3	01010	10	100
10|101	4	2	10010	18	324
110|10	1	3	11010	26	676
11|010	2	2	11101	29	841
Sum					1941
Average					485.25
Max					841
Schemata	**After Reproduction**		**After Crossover**		
Expected Count $\frac{\Sigma f(H)}{f}$	**Actual Count**	**in Individual**	**New Expected Count**	**Actual Count**	**in Individual**
0.01	0	-	0.00	0	-
1.83	3	2, 3, 4	2.54	3	2, 3, 4
0.33	3	1, 3, 4	1.60	2	1, 3

Table 2: Genetic Algorithm at Work (Part II).
Continued from Table 1. See main text for explanation.

generation. Both average and best fitness values have significantly improved in the new generation.

In this example, we monitored only three schemata. There are, however, between $2^5 = 32$ and $4 \cdot 2^5 = 128$ schemata in this small population that were all *implicitly* evaluated in the same manner *in parallel* only at the small computational cost of copying and exchanging a few bit strings. The implicit arithmetics of finding and promoting the best schemata do not actually have to be carried out by the computer. They are, so to speak, side effects of the genetic paradigm. This *implicit parallelism* is the basic reason for the efficiency of genetic algorithms.

As an addendum, the stochastic universal sampling algorithm for minimisation of fitness values by J. E. Baker is implemented below in the C programming language. This implementation is especially elegant because the source code is quite short and the generation of *only one* random number between 0 and 1 is needed to perform a random selection among all individuals in one generation according to their individual fitness values.

```
k = 0;        /* k is an integer index of next individual to be selected */
ptr = Rand();                  /* spin the roulette wheel; 0 < ptr < 1 */
Scaling_Factor = 1.0 / (Prev_Worst_Fitness - Average_Current_Fitness);
for (Sum = i = 0; i < Popsize; i++)   /* Popsize is size of population */
{                     /* Fitness[i] is the fitness value of individual i */
  if (Fitness[i] < Prev_Worst_Fitness)
```

```
   Expected_Count = (Prev_Worst_Fitness - Fitness[i]) * Scaling_Factor;
  else Expected_Count = 0.0;
  for (Sum += Expected_Count; Sum > ptr; ptr++)
       sample[k++] = i; /* sample is an array of ''Popsize'' integers. */
}            /* The value of each array element defines an individual. */
```

These instructions fill the array SAMPLE with integer values in a way that each individual with a fitness better than the average fitness of the current generation *and* better than the worst fitness in the last generation gets a chance of replication proportional to its fitness. Note the concerted increments of SUM and PTR in the inner FOR-loop.

5.1.2. Evolution Strategy

In 1973 I. Rechenberg developed the so-called evolution strategy[10]. The central idea of a computational evolution strategy is to create from one (or, in general, μ) "parent(s)" one (or λ) "offspring" individuals that have been modified by a mutation operator. μ and λ are usually small integers. As with the genetic algorithm, parents and offspring represent potential solutions to an optimisation problem encoded as a set of numerical parameters. Either only the λ offspring or both μ parents plus λ offspring individuals compete for survival into the next generation.

The mutation rate can be adjusted between successive generations to optimise the search progress. The so-called "1/5 rule" predicts an optimal run time performance when about 20% of all mutations within one population produce "viable" offspring. If more mutations produce successful offspring, the mutation rate is probably too small and more mutations would be useful to cover more of the search space. If much less than 20% of all mutations lead to viable successors then it may be that the mutation rate is too large. A large mutation rate can have the effect that the individuals are scattered widely in search space and no convergence is found. In order to keep the mutation rate in an optimal region it was suggested that the current mutation rate be multiplied (divided) by 1.25 between successive generations if the count of successful mutations falls below (exceeds) 20%. Mutation is a domain specific operator and can be implemented, for example, as a Gaussian distributed increment of a numerical parameter.

There are five subtle differences between a genetic algorithm and the evolution strategy.

1. The evolution strategy was *designed as a function optimiser* while genetic algorithms were originally developed to demonstrate the benefits of crossover in simulated evolution.
2. Reproduction in genetic algorithms is *proportional to fitness* but not in evolution strategy.
3. The genetic algorithm makes a distinction between *genotype* (the bit strings which are manipulated by genetic operators) and *phenotype* (the decoded val-

10 I. Rechenberg, **Evolutionsstrategie**, Frommann-Holzboog, Stuttgart, 1973, 2nd Edition 1994.

ue, e.g. an integer value interpreted as a parameter of the objective function i.e. the fitness) of an individual whereas in the original evolution strategy both coincide.

4. In evolution strategy, *both parents and offspring* may compete to survive into the next generation but not so in the original genetic algorithm. The extension to an early model of evolution strategy to exclude parents was suggested by H.-P. Schwefel[11].
5. *Mutation* is the main force to drive an evolution strategy whereas it is *crossover* for the genetic algorithm. Meanwhile, both approaches have converged and hybrid systems use both mutation and crossover.

In the recent past, both genetic algorithms and evolution strategy have gained wide public attention and many applications try to combine the best of both worlds[12].

5.1.3. Genetic Programming

Genetic Programming, pioneered by J. Koza[13], carries the application of genetic algorithms one step further. Not only are the parameters of a formula subject to mutation and crossover but the goal is to automatically let a completely new algorithm and program evolve to model and satisfy a set of constraints. The preferred vehicle for this type of problem solving is, again, the traditional artificial intelligence programming language Lisp. In a genetic programming application the programmer defines the basic components that he or she thinks might be helpful to describe relations among a given set of input / output pairs of training data. A genetic algorithm then rearranges the building blocks until no improvement in the performance of the newly created programs can be seen. The result is a new computer program that symbolically solves the desired task. Unfortunately, these programs are often unreadable, redundant and require streamlining by a human programmer before they become understandable. To some extent this task can be done automatically based on a set of logical transformation rules for the equivalency of statements and expressions in program source code.

Although all this might sound quite futuristic there are already a number of applications where the genetic programming paradigm produced meaningful results[14]. The basic algorithm for genetic programming is shown in Figure 3.

In this section, the potential and advantage of genetic programming shall be illustrated by the comparison of numerical and symbolic function approximation. Assume we have a set of 100 data points, each with an x and y component and the question is "Which mathematical function $y = f(x)$ best approximates those 100

11 H.-P. Schwefel, **Numerical Optimization Of Computer Models**, Chichester, John Wiley, 1981, (originally published in 1977).

12 World Wide Web Servers with more information and software at http://lautaro.fb10.tu-berlin.de, http://www.cis.ohio-state.edu/hypertext/faq/usenet/ai-faq/genetic/top.html and http://www-mitpress.mit.edu/jrnls-catalog/evolution.html.

13 J. Koza, **Genetic Programming (I, II)**, MIT Press, 1993, 1994.

14 World Wide Web Server with technical reports from J. Koza at Stanford University, http://elib.stanford.edu.

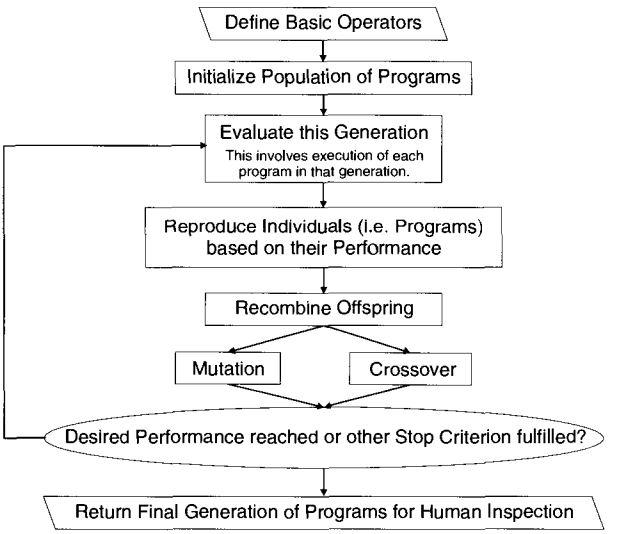


Figure 3: Genetic Programming Paradigm.

data points?". Traditionally, one would plot the data on an x / y chart, make an initial guess about the distribution and from there postulate a certain type of function, e.g. polynomial, trigonometric or exponential. Then, by least squares approximation the free parameters of that function can be either analytically or empirically adjusted (depending on the complexity of the function).

Not so with genetic programming. Here, the user defines a set of basic operators (in this case elementary mathematical functions, e.g. PLUS, MINUS, DIVIDE, MULTIPLY, SIN, LOG, LN, COS, TAN, SQUARE, EXP, ...) and a genetic algorithm is used to assemble those mathematical operators into a more complex function. The individuals for the genetic algorithm to work on are symbolic representations of the objective function. For the function aproximation application the performance of each individual is given by the sum of root mean square deviation of the function values of the newly assembled function $y' = f_i(x)$ and the y values that represent the desired output in the training stage. Individuals can *mutate* (one basic building block replaces another) or perform *crossover* (part of one individual is exchanged with part of another individual). Thus, new, complex functions emerge. In contrast to numerical approximation the restriction to one predefined type of function which has initially to be guessed from a preliminary x / y plot and which is invariable during the run is abandoned.

How to make Lisp-programs evolve? Lisp is intriguingly simple[15] in terms of data typing (see section 2.1.1). One can get around with only two types of data: atoms and lists. Moreover, there is no formal distinction between the type of data and the

15 *Another Glitch in the Call* (sung to the tune of a recent Pink Floyd song)
"We don't need no indirection,
We don't need no flow control,
No data typing or declarations,
Did you leave the lists alone?

type of a program. Both are identical. This makes it especially easy to design and implement a program that operates on other programs. The EVAL function in Lisp can be used to evaluate any such newly assembled program during run time. Thus, the basic edit-compile-run cycle for program development can be automated.

As an example, we shall look at a small program that implements the idea of genetic programming for the above mentioned problem of symbolic approximation of a mathematical function. The task is to find a symbolic representation of a function $y = f(x)$ that satisfies the following constraints:

x	$y = f(x)$
1	3
2	5
3	7

Here, one version with source code to solve this problem by genetic programming is presented. This program was written and tested with the TI Scheme interpreter PC Scheme v3.03. Scheme[16,17] is one major dialect of Lisp. In the source code below all upper case words are keywords of the Scheme programming language. Words that start with an upper case letter and continue lower case are user defined variables or functions.

First, let's examine the basic principle behind genetic programming: manipulating programs like data[18]. The following three function definitions are given:

```
(DEFINE My-Function
     '(LAMBDA (X) (+ X (* A B))))
(DEFINE Environment
     (LET ((A 2 ) (B 3)) (THE-ENVIRONMENT)))
(DEFINE Function-Evaluation
     (LAMBDA (FUNC ENV ARG) ((EVAL FUNC ENV) ARG)))
```

The first line defines the function $f(x) = x + A \cdot B$, where A and B are constants. The lambda-expression is quoted because this is a function definition, not the application (call) of a function to a particular argument. The second DEFINE expression establishes a local ENVIRONMENT where the variables A and B are assigned the values 2 and 3. There can be different values assigned for A and B on the global level without a conflict, e.g. (DEFINE A 12345). The third definition establishes the function FUNCTION-EVALUATION. This function takes three arguments: 1) another function, 2) one argument to that function and 3) a local environment with variable bindings. Now, typing

```
⇒ (Function-Evaluation My-Function Environment 4)
```

Hey! Hacker! Leave those lists alone!
All in all it's just a pure-LISP function call,
All in all it's just a pure-LISP function call ..."

16 J. Rees, W. Clinger (eds.), *Revised Report on the Algorithmic Language Scheme*, AI Memo 848a, Massachusetts Institute of Technology, September 1986.

17 H. Abelson, G. J. Sussman, J. Sussman, **Structure and Interpretation of Computer Programs**, MIT Press, Cambridge, Massachusetts, 1985.

18 See also section 2.1.1.

to the Scheme interpreter produces the output:

```
⇐ 10
```

The result, 10, is the sum of the product of A = 2 and B = 3 plus the function argument X = 4. FUNCTION-EVALUATION is passed the function MY-FUNCTION which is then evaluated in the context of ENVIRONMENT and applies MY-FUNCTION to the argument "4". Two things are important about this example:

1. The definition of MY-FUNCTION, that is (LAMBDA (X) (+ X (* A B))), is a mere list that can be manipulated like any other list of data items (atoms, sublists). For example, one could change the sublist (+ X (* A B)) into (+ X A B)) and thus have a new function f $(x) = x$ + A + B defined. At the same time, the Lisp function EVAL can interpret the list (LAMBDA (X) (+ X (* A B))) as a function object and apply it to a particular argument.
2. In Scheme, one can pass local variable bindings as an encapsulated environment (also called "closure").

These two features facilitate the implementation of a program that employs a genetic algorithm to discover novel mathematical functions. The first item ensures the availability of data objects that can be modified and interpreted as mathematical functions. The second item provides a practical means of implementing constants and local bindings. To keep the following example simple the second feature will not be used here. Instead, all properties of the functions to be optimised by the genetic algorithm are assumed to be contained in their LAMBDA-definitions.

We start with the definition of the top level function which will be given to the Scheme interpreter. The call of RUN initiates a genetic programming application that optimises a symbolic function representation to fit a set of [x, f (x)] pairs. The three arguments INITIAL-FUNCTION, POP-SIZE and MAX-ITER specify in turn the mathematical function that is used to initialise the first generation; the population size of each generation; and the maximum number of generations to evaluate in that run. If an exact solution to the optimisation problem has been found before MAX-ITER generations are reached the program stops and returns the solution. Alternatively, one could also directly call GENETIC-PROGRAMMING at the top level.

```
(DEFINE Run (LAMBDA (Initial-Function Pop-Size Max-Iter)
  (Genetic-Programming Initial-Function Pop-Size Max-Iter)))
```

The following definitions establish the objective of this application. DATA is the list of a set of [x, f (x)] pairs that determine the input / output behaviour of the unknown mathematical function. X-DATA and Y-DATA are derived lists with only the x and y values. The function FUNCTION will be used later to initialise the genetic algorithm.

```
(DEFINE Data '((1 3) (2 5) (3 7)))
(DEFINE X-Data (MAPCAR CAR Data))
(DEFINE Y-Data (MAPCAR CADR Data))
(DEFINE Function '(LAMBDA (X) (+ X 1)))
```

The following function INITIALISE-POPULATION takes a list FDEF and a number N and returns a list with N sublists FDEF. Here, FDEF will be the definition of a LAMBDA-function in list form and the result of INITIALISE-POPULATION is a complete population of N (yet identical) mathematical functions.

```
(DEFINE Initialise-Population
  (LAMBDA (Fdef N)
    (COND ((EQ? N 0) NIL)
          (T (CONS Fdef (Initialise-Population Fdef (- N 1)))))))
```

CALC-DATA applies the definition of a function FUNCTION to the x values of this application. The variable X-DATA is defined globally (see above). This function does not perform error checking (e.g. for division by zero).

```
(DEFINE Calc-Data
  (LAMBDA (Function)
    (MAPCAR (EVAL Function) X-Data)))
```

CALC-FITNESS, as follows, computes the fitness of a mathematical function by comparing its y values to the desired y values in Y-DATA. CONCAT recursively constructs a list of Lisp dotted pairs (<actual y value> . <desired y value>), e.g. (47 . 56). There must be a blank before and after the dot. Dotted pairs are necessary because in contrast to other Lisp dialects in Scheme the MAPCAR function can only call a function with one argument whereas the fitness calculation requires the comparison of both y values (actual and desired) which therefore must be present simultaneously. Alternatively, one could define MAP2CAR.

```
(DEFINE Calc-Fitness
  (LAMBDA (List-Of-Ys)
    (MAPCAR (LAMBDA (Pair)
                (ABS (- (CAR Pair) (CDR Pair))))
            (Concat List-Of-Ys Y-Data))))

(DEFINE Concat
  (LAMBDA (L1 L2)
    (COND (L1 (CONS (CONS (CAR L1) (CAR L2))
                    (Concat (CDR L1) (CDR L2))))
          (T NIL))))
```

GENETIC-PROGRAMMING is the main function to control the genetic algorithm. After initialisation of POPULATION with the initial function INIT-FUNCTION all individuals are mutated with probability 100%. This is done to start the search with a diverse population that covers different regions in search space. Finally, the initial population is printed using PRINT-POP.

```
(DEFINE Genetic-Programming
  (LAMBDA (Init-Function Pop-Size Max-Iter)
    (LET ((New-Population (Mutate (Initialise-Population
                                   Init-Function Pop-Size) 100))
          (Results NIL)
          (Fitness NIL)
          (Population NIL))
      (NEWLINE) (PRINC "Initial Population")
      (Print-Pop New-Population)
```

After this, the main loop begins. For maximally MAX-ITER iterations the members of one generation are evaluated. First, the function values $y = f(x)$ of all individuals in one generation are collected with CALC-DATA and subsequently converted into fitness values via CALC-FITNESS.

```
(DO ((Iter 1 (1+ Iter)))
    ((OR (> Iter Max-Iter)
         (AND Fitness (MEMBER 0 Fitness)))
     (NEWLINE)
     (COND ((> Iter Max-Iter)
            (PRINC "No Solutions Found")
            (NEWLINE)
            NIL)
           ((MEMBER 0 Fitness)
            (PRINC "Solution is")
            (MAP (LAMBDA (X) (AND (EQ? (CAR X) 0)
                                  (PRINT (CDR X))))
                 (Concat Fitness Population))
            (NEWLINE)
            T)))
    (SET! Population New-Population)
    (SET! Results (MAPCAR Calc-Data Population))
    (NEWLINE) (PRINC "Generation ") (PRINC Iter)
    (NEWLINE) (PRINC "Results ") (PRINC Results)
    (SET! Fitness (MAPCAR Sum
                          (MAPCAR Calc-Fitness Results)))
    (NEWLINE) (PRINC "Fitness ") (PRINC Fitness)
```

REPRODUCE takes as input fitness values and selects individuals proportional to their fitness for reproduction. As this implementation may not always return a population of the exact size POP-SIZE this has to be checked and adjusted. After that, the new population is printed.

```
    (SET! New-Population (Reproduce Population Fitness))
    (COND ((< (LENGTH New-Population) (LENGTH Population))
           (SET! New-Population
                 (CONS (CAR New-Population) New-Population)))
          ((> (LENGTH New-Population) (LENGTH Population))
           (SET! New-Population (CDR New-Population))))
    (NEWLINE) (PRINC "New Population After Reproduce")
    (Print-Pop New-Population)
```

Next in the body of GENETIC-PROGRAMMING each individual of the new generation is mutated with a 30% probability. Again, the results are printed to the screen.

```
    (SET! New-Population (Mutate New-Population 30))
    (NEWLINE) (PRINC "New Population After Mutate")
    (Print-Pop New-Population)
```

Finally, one crossover between two individuals in the current population is performed and the results printed. In another application one might prefer to have more than only one crossover per generation.

```
    (SET! New-Population (Crossover New-Population))
    (NEWLINE) (PRINC "New Population Crossover")
    (Print-Pop New-Population)))))
```

This concludes the instructions in the DO-loop. Next the termination condition is checked (see above) and depending on the result either the cycle for the next generation is entered or the program is halted.

MUTATE does a simple mutation on a mathematical function. With a certain PROBABILITY each individual of POPULATION is changed in the following way. The first operator in the function definition of FUNCTION is replaced by a "+", "–", "*" or a "/", each with an equal probability of 25%. For example, the function definition (LAMBDA (X) (+ X (+ X 1))) could be changed to (LAMBDA (X) (* X (+ X 1))) by mutation.

```
(DEFINE Mutate
  (LAMBDA (Population Probability)
    (MAPCAR (LAMBDA (Lmbd)
              (LET ((Function (COPY (CADDR Lmbd)))
                    (Dice (RANDOM 100))
                    (Chance (RANDOM 100)))
                (AND (< Chance Probability)
                     (COND ((< Dice 25) (SET-CAR! Function '+))
                           ((< Dice 50) (SET-CAR! Function '-))
                           ((< Dice 75) (SET-CAR! Function '*))
                           (T (SET-CAR! Function '/))))
                (APPEND '(LAMBDA (X)) (COPY (LIST Function)))))
            Population)))
```

CROSSOVER takes as input an entire population and returns the same population but with one individual changed. The change involves grafting either a part or the whole of one individual (the corresponding definition of the mathematical function is kept in the local variable L1) into another (after node L2; before node F2). CROSSOVER performs several list manipulations. I1 is the index of the source code donating individual within the current population. I2 is the source code receiving individual that will be modified. More complicated versions of CROSSOVER could also transfer code the other way from L2 to L1 and / or process more than one pair of individuals.

```
(DEFINE Crossover
 (LAMBDA (Population)
   (LET ((L (LENGTH Population))
         (I1 NIL) (I2 NIL) (I NIL)
         (F1 NIL) (F2 NIL) (L1 NIL) (L2 NIL))
     (SET! I1 (RANDOM L))  (SET! I2 (RANDOM L))
     (SET! I I2)
     (SET! F1 (CADDR (LIST-REF Population I1)))
     (SET! F2 (CADDR (LIST-REF Population I2)))
     (SET! L1 (LENGTH F1)) (SET! L2 (LENGTH F2))
     (SET! I1 (RANDOM L1)) (SET! I2 (RANDOM L2))
     (SET! L1 (COND ((EQ? I1 0) F1)
                    (T (LIST-REF F1 I1))))
     (SET! L2 (COND ((EQ? I2 0) (APPEND F2 (LIST L1)))
                    (T (MAPCAR (LAMBDA (X)
                                 (COND ((EQUAL? X (LIST-REF F2 I2))
                                        (COPY L1))
                                       (T X)))
```

```
                                F2))))
    (MAPCAR (LAMBDA (X)
              (COND ((AND I (EQUAL? X (LIST-REF Population I)))
                     (SET! I NIL)
                     (COPY (APPEND '(LAMBDA (X)) (LIST L2))))
                    (T X)))
            Population))))
```

REPRODUCE takes a list of fitness values and a list of individuals as input. In this application one individual is better than another if it has a lower fitness. Therefore, we must derive the reproduction frequency of each individual from its fitness. This is done by complementing each fitness value with the worst (highest) fitness and normalising those values to add up to 1. These numbers are then multiplied and rounded to give the count of the desired number of offspring.

$$\begin{aligned}
\text{reverse fitness}_i &= \text{worst fitness} - \text{fitness}_i \\
\text{sum reverse fitness} &= \sum_{i=1}^{n} \text{reverse fitness}_i \\
\text{number of offspring}_i &= \mathit{Round}\left(\frac{N \cdot \text{reverse fitness}_i}{\text{sum reverse fitness}}\right)
\end{aligned}$$

Because of precision errors in some cases the new population size may deviate by 1 from the original population size. This is checked and corrected in the function GENETIC-PROGRAMMING above. The function SUM (definition see below) adds numbers in a list. REPRODUCE, as follows, prints the following information to the screen: reverse fitness, i.e. the complement of each original fitness value with respect to the worst (highest) fitness in the current generation and the reproduction count for each individual. The return value is the new population based on the reproduction count of each individual proportional to its fitness.

```
(DEFINE Reproduce
  (LAMBDA (Population Fitness)
    (LET ((Pop-Size (LENGTH Fitness))
          (Max-Fit (APPLY MAX Fitness))
          (Sum-Fit NIL)
          (New-Fit NIL))
      (AND (EQ? (APPLY MIN Fitness) (APPLY MAX Fitness))
           (NOT (EQ? 0 (CAR Fitness)))
           (SET! Fitness (CONS (+ (CAR Fitness) 1) (CDR Fitness))))
      (SET! New-Fit (MAPCAR (LAMBDA (X)
                              (- Max-Fit X)) Fitness))
      (SET! Sum-Fit (Sum New-Fit))
      (NEWLINE) (PRINC "Rev Fit ") (PRINC New-Fit)
      (SET! New-Fit (MAPCAR (LAMBDA (X)
                              (ROUND (* Pop-Size
                                        (/ X Sum-Fit)))) New-Fit))
      (NEWLINE) (PRINC "Reprodc ") (PRINC New-Fit)
      (APPLY APPEND (MAPCAR (LAMBDA (X)
                              (Initialise-Population (CAR X) (CDR X)))
                            (Concat Population New-Fit))))))
```

Finally, there are two auxiliary function, PRINT-POP and SUM. PRINT-POP prints the source code of all individuals of one population (i.e. the mathematical functions in the current generation) to the screen in one line. SUM returns the sum of numbers in a list without a "+" sign. This function differs from the predefined function "+" in that it takes one argument which must be a list of numbers. In contrast, the predefined Lisp function "+" takes several separate arguments that must be numbers.

In the following, the output of the program after issuing the command

```
⇒ (Run Function 5 4)
```

is shown. First, we see the initial population of functions. Only the body of the functions, not the heads nor variable lists are printed (because those do not change and to make the output short and clear). Originally, all functions were set to FUNCTION which increments its only variable by one:

```
(DEFINE Function '(LAMBDA (X) (+ X 1)))
```

Because the mutation rate is 100% for the first generation the mutation operator modifies all individuals. Only the second individual was conservatively mutated (the plus sign replaced by a plus sign). Next follows the fitness calculation for the first generation. Then the individuals are replicated according to their fitness. After reproduction another cycle of mutation and crossover is performed. This completes the population of the next generation. The new individuals will subsequently have to be evaluated by the fitness function.

```
(- X 1) (+ X 1) (* X 1) (- X 1) (- X 1)
Generation 1
Results ((0 1 2) (2 3 4) (1 2 3) (0 1 2) (0 1 2))
Fitness (12 6 9 12 12)
Rev Fit (0 6 3 0 0)
Reprodc (0 3 2 0 0)
New Population after Reproduce
(+ X 1) (+ X 1) (+ X 1) (* X 1) (* X 1)
New Population after Mutate
(+ X 1) (* X 1) (+ X 1) (* X 1) (* X 1)
New Population Crossover
(+ X 1) (* X (+ X 1)) (+ X 1) (* X 1) (* X 1)
```

In the second generation a new individual (* X (+ X 1)) which corresponds to the function $f(x) = x(x + 1) = x^2 + x$ was created by crossover. This illustrates the ability of crossover to construct more complex, nested functions. This version of the crossover operator only combines or replaces substructures in equations but does not eliminate parts of an equation. The consequence is that a function may become quite large and complicated after a few crossover events but cannot be simplified again. The interested reader may contemplate an extension to CROSSOVER that simplifies the source code of the donor individual appropriately.

```
Generation 2
Results ((2 3 4) (2 6 12) (2 3 4) (1 2 3) (1 2 3))
Fitness (6 7 6 9 9)
Rev Fit (3 2 3 0 0)
Reprodc (2 1 2 0 0)
New Population after Reproduce
```

```
(+ X 1) (+ X 1) (* X (+ X 1)) (+ X 1) (+ X 1)
New Population after Mutate
(+ X 1) (+ X 1) (* X (+ X 1)) (+ X 1) (* X 1)
New Population Crossover
(+ X 1 (* X 1)) (+ X 1) (* X (+ X 1)) (+ X 1) (* X 1)
```

Through another crossover event the second generation produced the individual (+ X 1 (* X 1)) which corresponds to the function f $(x) = x + 1 + (x * 1) = 2x + 1$. This individual entirely satisfies the constraints on x and y values in the problem definition (x values [1 2 3], y values [3 5 7]). This individual is identified as a perfect solution in generation 3 on position 1. However, it contains an obsolete and superfluous multiplication of x by 1. This illustrates that genetic programming may well produce correct but unnecessary complicated answers that require post-processing.

```
Generation 3
Results ((3 5 7) (2 3 4) (2 6 12) (2 3 4) (1 2 3))
Fitness (0 6 7 6 9)
Rev Fit (9 3 2 3 0)
Reprodc (3 1 1 1 0)
New Population after Reproduce
(+ X 1 (* X 1)) (+ X 1 (* X 1)) (+ X 1) (* X (+ X 1)) (+ X 1)
New Population after Mutate
(/ X 1 (* X 1)) (+ X 1 (* X 1)) (+ X 1) (* X (+ X 1)) (+ X 1)
New Population Crossover
(/ X 1 (* X 1)) (+ X 1 (* X 1)) (+ X (+ X 1)) (* X (+ X 1)) (+ X 1)
Solution is
(LAMBDA (X) (+ X 1 (* X 1)))
T
```

Before the program stops it performs one more cycle of reproduction, mutation and crossover. This is just to once more illustrate the mechanism of reproduction, mutation and crossover. Finally, the successful individual (of the previous generation) is returned.

As can be seen, there is a fascinating ability of a computer program to optimise other programs by evolution and concerted trial and error. We have seen before that the genetic algorithm optimises the effort of evaluating successive generations. Genetic programming not only optimises a problem by numerical approximation but also delivers a symbolic representation that can lead to new insights about underlying correlations in the data analysed. Note, however, that a solution is not guaranteed. Also, if genetic programming returns a solution, one cannot be certain that it is the best or simplest one. If repeated runs produce the same solution this may be an indication of its validity but in general the output is heavily dependent on the representation formalism and the choice and implementation of mutation and crossover operators. One has to carefully think about inherent limitations. For example, in the above example there is a tendency to construct more and more complicated functions without the ability to simplify them again. It is therefore often necessary to transform and optimise the programs produced by the genetic programming system in order to understand them.

For the genetic programming paradigm to produce a proper solution to a problem the user must:

- *define* the problem in terms of correlated input / output pairs for training,
- *know* about the basic functional elements of which a solution can be composed, and
- have those basic elements *implemented* where they are not predefined in Lisp.

The true merit of genetic programming lies in the ability to *symbolically* learn from empirical data and construct new concepts. This is fundamentally different from the numerical optimisation of predefined models or equations. In genetic programming the basic elements for the functions to be assembled can be abstract operators that could simulate anything from stock market behaviour to metabolism of an eucaryotic cell. This concludes the introduction into genetic programming.

5.1.4. Simulated Annealing

Simulated annealing is an algorithm originally devised to model the behaviour of a collection of atoms or molecules in a temperature gradient[19]. This model was later abstracted to simulate the settling of any system of which the objective is to adopt a state of minimal energy. Simulated annealing is similar to Monte-Carlo search but with one distinction: Monte-Carlo randomly generates new candidates for the solution to the optimisation problem. A candidate is retained only if it has a higher fitness (lower energy) that its predecessor or discarded otherwise. Simulated annealing, however, occasionally accepts a new candidate even if it performs worse than its predecessor. The basic simulated annealing algorithm is as follows.

1. Define the problem at hand as a function minimisation problem $y = \mathrm{f}(x, z, v, \ldots)$ and initialise the variables temperature $T = T_0$ and *cycle* = 0.
2. Initialise the algorithm with a random instantiation of the parameters of the objective function and calculate its fitness values, e.g. $y = \mathrm{f}(7, 2, 5, \ldots)$. The set of parameter values (7, 2, 5,...) is called the start individual.
3. Increment *cycle* by 1. Derive another individual from the parent of step 2 (or step 4, respectively) and calculate its fitness, e.g. $y' = \mathrm{f}(8, 1, 4, \ldots)$. The new individual can be randomly derived, for example, by Gaussian distributed increments on the parameters. In general, the computation of a successor individual is application dependent.
4. If the fitness of the new individual is better than that of its parent ($y' < y$) then replace the parent by its offspring. Otherwise replace the parent by its offspring only with the probability $p(\Delta E) = e^{-\frac{\Delta E}{kT}}$, $with\ \Delta E = y' - y$. . Originally, k was the Boltzmann constant ($k = 1.3805 \cdot 10^{-23}$ J/K) but its value may be adjusted to fit the particular objective function of an application. T is the current temperature. If the parent is not replaced it serves again as a predecessor for the derivation of another offspring individual.
5. If a maximum number of cycles is reached (*cycle* > *max-cycle*), the fitness value is good enough, or if ΔE remains 0 over a long time then stop and return the final individual as the solution to the minimisation problem. Otherwise, update

19 N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, *Equation of state calculations fast computing machines*, **J. Chem. Phys.**, vol 21, p. 1087, 1953.

the temperature $T_{i+1} = T_i \cdot q$ by a multiplicative constant q (e.g. $q = 0.9$) or by an additive constant δ (e.g. $\delta = 0.001$ in $T_{i+1} = T_i - \delta$) and go back to step 3.

Comparing simulated annealing with the genetic algorithm or evolution strategy we find that we have here a population size of only one and also only one offspring individual. Generation replacement is partly controlled stochastically.

As an example, the C source code of a simple but complete and operational application of simulated annealing to the Travelling Salesman problem (TSP) is presented. The problem is: given N cities, their coordinates and a distance function, what is the shortest round-trip during which each city is visited exactly once and which leads back to the origin of departure?

There are three standard libraries required:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

Some constants are defined at compile time. CITY is a structure that holds the two coordinates (X, Y) of a city.

```
#define RAND_MAX        32768.0 /* Random generator max return value */
#define N_CITIES             20 /* Number of cities */
#define MAX_ITER           1000 /* Max cycles after which to stop */
#define T0                   50 /* Start temperator T0 */
#define T1T0               0.90 /* Temperatur factor T1 / T0 */
#define KB                 1000 /* Boltzmann constant modified */
#define MAX_X               540 /* Boundary for city x-coordinate */
#define MAX_Y               380 /* Boundary for city y-coordinate */

typedef struct {
    double x;
    double y;
} city;
```

INIT_CITIES_RANDOM initialises a number of N_CITIES cities with random X / Y coordinates in the intervals [0, MAX_X] and [0, MAX_Y]. The standard random number generator RAND in C is not very good. On some systems, RANDOM is available which may be used to replace RAND.

```
void init_cities_random (city *cities, int n_cities, int *map1,
                         int *map2)
{
    int i;
      for (i=0; i<n_cities; i++)
    {
        cities[i].x = rand () / (double) RAND_MAX * MAX_X;
        cities[i].y = rand () / (double) RAND_MAX * MAX_Y;
        map1[i] = i;
        map2[i] = i;
    }
}
```

WRITE_SOLUTION prints the result to the file RESULT.

```
void write_solution (city *cities, int n_cities, int *map, double cost,
                     double temp, long iter, double kb)
```

```
{
    FILE *res;
    int i;
      res = fopen ("result", "a");
    fprintf (res, "Cities %d   Iterations %ld   Cost %.3lf
                   Temperature %.3lf   kb %.1l\n", n_cities, iter-1,
                   cost, temp, kb);
    fprintf (res, "Round-trip ");
    for (i = 0; i < n_cities; i++)
        fprintf (res, "%d ", map[i] + 1);
    fprintf (res, "\n");
    for (i = 0; i < n_cities; i++)
        fprintf (res, "%2d %8.3lf %8.3l\n", i+1, cities[i].x,
                                                 cities[i].y);
    fprintf (res, "\n");
}
```

DRAW_CITIES should draw the cities as points on a computer screen and connect those points based on the information in MAP. As this portion of the program is highly platform dependent the reader should consult the manual of his / her C library for available graphics commands and implement this function from there. The following version was written for the Borland C compiler.

```
void draw_cities (city *cities, int n_cities, int *map, double cost,
                  long iter, double temp, long uphill)
{
    int gmode, errorcode;
    int x, y, x1, y1, i;
    char *string, s[50];
    string = (char *) malloc (50 + 5 * n_cities);
    cleardevice();
    outtextxy (130, 0, "Travelling Sales Man * Simulated Annealing");
    for (i = 0; i < n_cities; i++)
    {
        x = (int) floor (cities[i].x);
        y = (int) floor (cities[i].y);
        sprintf (string, "City %d", i+1);
        outtextxy (x + OFFSET, y + OFFSET, string);
    }
    for (i = 0; i < n_cities - 1; i++)
    {
        x  = (int) floor (cities[map[i]].x);
        y  = (int) floor (cities[map[i]].y);
        x1 = (int) floor (cities[map[i+1]].x);
        y1 = (int) floor (cities[map[i+1]].y);
        line (x + OFFSET, y + OFFSET, x1 + OFFSET, y1 + OFFSET);

    }
    x  = (int) floor (cities[map[n_cities-1]].x);
    y  = (int) floor (cities[map[n_cities-1]].y);
    x1 = (int) floor (cities[map[0]].x);
    y1 = (int) floor (cities[map[0]].y);
    line (x + OFFSET, y + OFFSET, x1 + OFFSET, y1 + OFFSET);
    sprintf (string, "Last Improvement in Cycle %5ld Cost %8.2lf
            Temperature %8.3lf Uphill %5ld", iter, cost, temp, uphill);
    outtextxy (0, getmaxy() - 30, string);
```

```
    strcpy (string, "");
    strcpy (s, "");
    for (i=0; i<n_cities; i++)
    {
        sprintf (s, " %d ", map[i]+1);
        strcat (string, s);
    }
    outtextxy (0, getmaxy() - 10, string);
    free (string);
}
```

COPY_CITIES copies the route in array MAP2 to array MAP1.

```
void copy_cities (int n_cities, int *map1, int *map2)
{
    int i;
    for (i = 0; i < n_cities; i++)
      map1[i] = map2[i];
}
```

SHUFFLE_CITIES takes a continuous portion of the route in array MAP1, inverses this portion and inserts it at the proper position in MAP2. In this application, this is the only operator to generate a new individual.

```
void shuffle_cities (int n_cities, int *map1, int *map2)
{
    int i, p1, p2;
    p1 = floor (rand() / (double) RAND_MAX * n_cities);
    p2 = floor (rand() / (double) RAND_MAX * n_cities);
    if (p1 > p2)
    {
        i = p1;
        p1 = p2;
        p2 = i;
    }
    for (i = p1; i <= p2; i++)
        map2[i] = map1[p2-(i-p1)];
}
```

EVALUATE_ROUTE measures the Euclidean distance of a round-trip through all N cities, visiting each city exactly once and returning to the start point.

```
double evaluate_route (int *map, city *cities, int n_cities)
{
    int i;
    double sum = 0.0;
    for (i = 0; i < n_cities - 1; i++)
        sum += sqrt ((cities[map[i]].x - cities[map[i+1]].x)*

                     (cities[map[i]].x - cities[map[i+1]].x) +
                     (cities[map[i]].y - cities[map[i+1]].y)*
                     (cities[map[i]].y - cities[map[i+1]].y));
    sum += sqrt ((cities[map[n_cities - 1]].x - cities[map[0]].x)*
                 (cities[map[n_cities - 1]].x - cities[map[0]].x) +
                 (cities[map[n_cities - 1]].y - cities[map[0]].y)*
                 (cities[map[n_cities - 1]].y - cities[map[0]].y));
    return sum;
}
```

This following is the MAIN function. The first part handles the initialisation of various variables and performs memory allocation for N cities and two maps.

```
int main (void)
{
    city *cities;
    int *map1, *map2, n_cities;
    long iter, max_iter, uphill;
    double t, t0, t1t0, cost1, cost2, probability, kb;

    max_iter = MAX_ITER;
    n_cities = N_CITIES;
    t1t0     = T1T0;
    t0       = T0;
    kb       = KB;

    cities = (city *) calloc (n_cities, sizeof (city));
    map1   = (int *)  calloc (n_cities, sizeof (int));
    map2   = (int *)  calloc (n_cities, sizeof (int));
    t      = t0;       /* starting temperature */
    iter   =  0;       /* cycle counter */
    uphill =  0;       /* number of cycles were bad offspring wins */
```

Next, the cities are initialised, the cost (route length) of the first round-trip is calculated, and the first map is drawn on the screen.

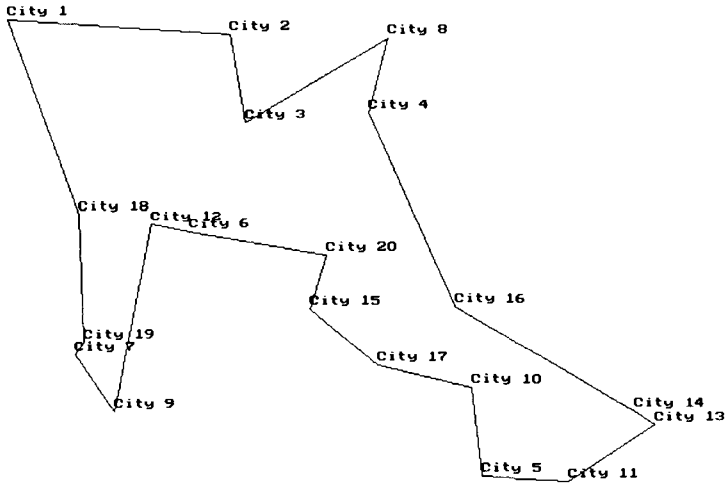
```
    init_cities_random (cities, n_cities, map1, map2);
    cost1 = evaluate_route (map1, cities, n_cities);
    draw_cities (cities, n_cities, map1, cost1, iter, t, uphill);
```

Now, the actual annealing process is carried out. A new candidate is produced, evaluated and compared to its predecessor. If the new individual outperforms its predecessor it replaces the predecessor for the next cycle.

```
    while (++iter <= max_iter)
    {
        shuffle_cities (n_cities, map1, map2);
        cost2 = evaluate_route (map2, cities, n_cities);
        if (cost2 < cost1)
        {
            cost1 = cost2;
            copy_cities (n_cities, map1, map2);
            draw_cities (cities, n_cities, map1, cost1, iter,
                         t, uphill);
        }
        else
```

Otherwise, a probability value based on the comparison of old and new fitness values is used to determine whether or not to replace the parent.

```
        {
            probability = exp (- (cost2 - cost1) / (kb * t));
            if (probability > (rand () / (double) RAND_MAX))
            {
                uphill++;
                cost1 = cost2;
                copy_cities (n_cities, map1, map2);
                draw_cities (cities, n_cities, map1, cost1, iter,
```



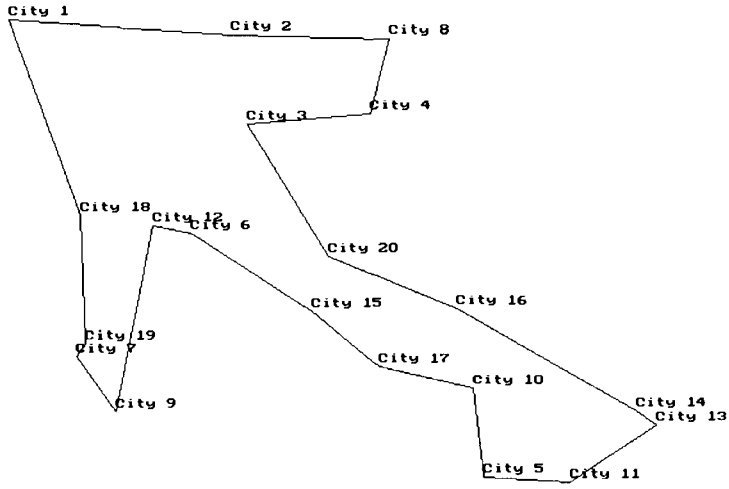

```
1000 iterations finished.
Last Improvement in Cycle   979 Cost  1818.13 Temperature    0.001 Uphill    86
 1  18  19  7  9  12  6  20  15  17  10  5  11  13  14  16  4  8  3  2
```

Figure 4: Travelling Salesman Solution by Simulated Annealing.
This round-trip was found by the simulated annealing example program in the text.

```
                                t, uphill);
            }
            else
                copy_cities (n_cities, map2, map1);
        }
        if ((t * t1t0) > 0.001) /* lower temperature boundary */
            t *= t1t0;
    }
    write_solution (cities, n_cities, map1, cost1, t, iter, kb);
    free (cities);
    free (map1);
    free (map2);
    return 0;
}
```

If the parent was not replaced the original map must be restored. The temperature is updated after each cycle. After the predefined number of cycles the solution is printed to a file and dynamically allocated memory blocks are released.

Figures 4 and 5 show two screen shots of the travelling salesman application. Figure 4 shows the result of the simulated annealing procedure after 1,000 cycles. The shortest path found has the length 1818.13 units. In 86 cycles a successor worse than its parent was selected for the next cycle. Figure 5 has the result of a Monte-Carlo random search on the same problem but run for 10,000 cycles. Although ten times as many trials were performed as with the simulated annealing run, the best




```
10000 iterations finished.
Last Improvement in Cycle   633 Cost  1860.59 Temperature    0.001 Uphill     0
 11  13  14  16  20  3  4  8  2  1  18  19  7  9  12  6  15  17  10  5
```

Figure 5: Travelling Salesman Solution by Monte-Carlo.
This round-trip was found by a Monte-Carlo algorithm. Although ten times as many trials were performed than with the simulating annealing run of Figure 4 the final route was 42.46 units longer. For a general statement about the relative performance of both methods more trials would have to be performed and statistically analysed.

solution found is 1860.59 units long and thus worse than the result of simulated annealing. For a general statement on the relative performance of simulated annealing and Monte-Carlo search more runs have to be performed and statistically analysed.

5.2. Applications

5.2.1. 2-D Protein Model for Conformation Search

R. Unger and J. Moult have used a 2-dimensional, simplified protein model to demonstrate the usefulness of a genetic algorithm in the search for minimal energy conformations[20]. Their protein model has only two kinds of amino acid residues: hydrophobic (black circles) and hydrophilic ones (white circles). The "protein" is a chain of these two types of residues on a 2-dimensional, orthogonal grid. Bond angles are restricted to the values 90°, 180° and 270°. The "force field" to determine the inner energy of a fold is defined to be the sum of all hydrophobic interactions.

20 R. Unger, J. Moult, *Genetic Algorithms for Protein Folding Simulations*, **Journal of Molecular Biology**, vol 231, pp. 75, 81, 1993.

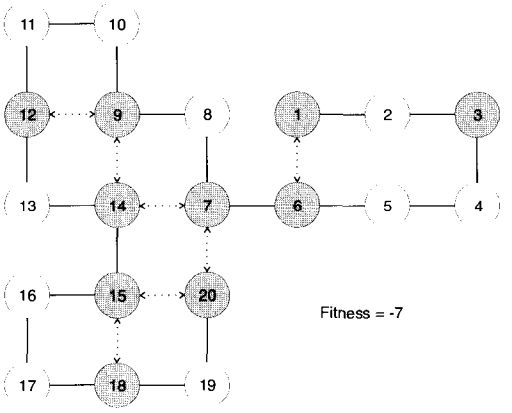


Figure 6: Simplified 2-Dimensional Protein Model.
Each hydrophobic interaction between two adjacent hydrophobic residues (black circles) contributes -1 energy units. There are seven such interactions (indicated by dotted arrows) in this molecule.

A hydrophobic interaction contributes -1 energy units when two not directly connected hydrophobic residues come to lie orthogonally adjacent to each other. For example, the "molecule" in Figure 6 has seven hydrophobic interactions and hence an "energy" of -7 units.

This section presents the implementation of a simplified 2-D protein model with a hydrophobic energy function for use with the public domain genetic algorithm software Genesis[21]. Genesis version 5.0 was written to promote the study of genetic algorithms for function minimisation. Since genetic algorithms are task independent optimisers the user must provide a domain specific crossover operator and an evaluation function which returns a fitness value for each individual. The Genesis system was written by J. J. Grefenstette in the programming language C.

To represent the conformation of a 2-D protein the following definitions are used. The *primary structure* is a list of bits 0 and 1, where 0 stands for a hydrophilic residue and 1 for a hydrophobic one (see the constant SEQUENCE in the example code below). The *conformation* is described by a bit string where every two bits in sequence define the bond angle between the current residue and the next one in sequence. Figure 7 explains the meaning of the four directional codes 00, 01, 10, and 11 for a bond angle.

Crossover is the prominent operator in genetic algorithms. For 2-D proteins it can be implemented as follows (Figure 8). One bond (except the first in a molecule) is randomly selected. The fragments to the left and right of that bond (crossover site) are exchanged between two individuals. The new value for the bond at the crossover site is selected such that the overlap of the two fragments being joined

21 By J. J. Grefenstette. Connect to leland.stanford.edu, log in as *anonymous* using your email address as password and change to the directory /pub/cs426/GA/GENESIS/genesis. Then use *ftp* to download the software. Alternatively, use *archie genesis* to find alternate locations of the file.

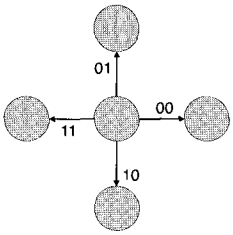


Figure 7: Encoding Conformation and Direction.
A positional code determines the location of the next residue, whether it is up (01), down (10), to the left (11) or right (00). Note: in analogy to Gray codes in this representation a change of one bit in the positional code implies an increment of 90° (or -90°) on the bond angle.

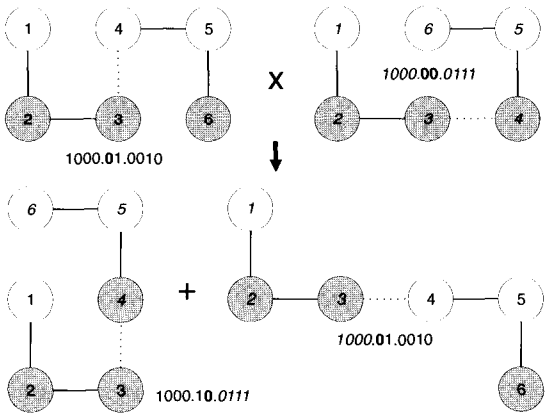


Figure 8: Crossover for 2-D Proteins.
This one-point crossover exchanges the bonds to the left and right of the crossover site. The bond at the crossover site is selected to minimise the overlaps between residues of the newly joined fragments.

becomes minimal. Depending on the structure of the fragments some overlap in the newly created individuals cannot always be avoided. However, individuals with overlaps die out quickly because they get high (i.e. bad) fitness values (see function COUNT_OVERLAPS below). The crossover probability is set to 60% for every pair of individuals.

The following C-source code is the complete implementation for crossover of 2-D proteins. The file with this code must be called CROSS.C and compiled and linked with the remaining Genesis files.

```
/* FILE CROSS.C */

#define SEQ_LENGTH  20              /* 2-D protein with 20 residues */
#define STR_LENGTH  ((SEQ_LENGTH - 1) * 2) /* 38 bits for 19 bonds */
#define X           ((SEQ_LENGTH + 1) * 2) /* max X grid extension */
#define Y           ((SEQ_LENGTH + 1) * 2) /* max Y grid extension */
```

```
#include "extern.h"

typedef struct choice {
    char piece[STR_LENGTH+1];
    double overlaps;
} choice;
```

The function SELECT_MIN_OVERLAP determines the one individual with the smallest number of overlapping residues out of n choices (n = UPPER - LOWER; UPPER and LOWER are indices of individuals in the current population). If two or more conformations have the same (smallest) number of overlaps one of them is chosen at random.

```
int select_min_overlap (choice *choices, int lower, int upper)
{
    int i, j, best[4];
    double tmp = 10e10;

    for (i = lower; i < upper; i++)
        if (choices[i].overlaps < tmp)
            tmp = choices[i].overlaps;
     j = 0;
    for (i = lower; i < upper; i++)
        if (choices[i].overlaps == tmp)
            best[j++] = i;
    return (best[Randint (0, j -1)]);
}
```

COUNT_OVERLAPS calculates the number of overlapping residues in one individual.

```
double count_overlaps (char *str)
{
    int i, j;
    int x = X / 2;
    int y = Y / 2;
    double sum = 0.0;
    static int matrix[X][Y];

    for (i = 0; i < X; i++)            /* initializing arrays */
        for (j = 0; j < Y; j++)
            matrix[i][j] = 0;
    for (i = 0; i < SEQ_LENGTH; i++)  /* filling arrays */
    {
        matrix[x][y]++;
        if (str[2*i] == '0')
            if (str[2*i+1] == '0')
                x += 1;
            else
                y += 1;
            else
                if (str[2*i+1] == '0')
                    y -= 1;
                else
                    x -= 1;
    }
```

```
    for (i = 0; i < X; i++)            /* summation */
        for (j = 0; j < Y; j++)
            if (matrix[i][j] > 1)
                sum += matrix[i][j] - 1;
      return sum;
}
```

Next, the source code of the actual CROSSOVER function is presented. CROSSOVER selects a pair of individuals, finds a crossover site and exchanges the fragments to the left and right between the two individuals. The length of individuals in this application is set to 40 although there are only 19 bonds and thus 38 bits required. The extra bond allows CROSSOVER to modify the last real bond in the molecule. Several test runs demonstrated that this feature improves the performance of the genetic algorithm.

```
Crossover()
{
    register int mom, dad;        /* participants in the crossover */
    register int xpoint; /* first crossover point w.r.t. structure */
    register int i, j, k;                  /* loop control variable */
    static int last;          /* last element to undergo Crossover */
     int boy, gal;         /* set if parents differ from offspring */
    static int firstflag = 1;
    static char i1[STR_LENGTH+1], i2[STR_LENGTH+1];
    static char piece1a[STR_LENGTH+1],piece1b[STR_LENGTH+1];
    static char piece2a[STR_LENGTH+1],piece2b[STR_LENGTH+1];
    static choice choices[8];

    Trace("Crossover entered");
    Dtrace("crossover");

    if (firstflag)
    {
        last = (C_rate*Popsize*Gapsize) - 0.5 ;
        firstflag = 0;
    }

    for (mom = 0; mom < last ; mom += 2)
    {
        dad = mom + 1; /* kids start as identical copies of parents */
        Unpack (New[mom].Gene, i1, Bytes);
        Unpack (New[dad].Gene, i2, Bytes);

        xpoint = 2 * Randint (1, SEQ_LENGTH - 1);/* crossover point */
        strncpy (piece1a, i1, xpoint);
        piece1a[xpoint] = '\0';
        strcpy (piece1b, i1 + xpoint);
        strncpy (piece2a, i2, xpoint);
        piece2a[xpoint] = '\0';
        strcpy (piece2b, i2 + xpoint);

        strcpy (choices[0].piece, piece1a);
        strcat (choices[0].piece, piece2b);
        for (i = 1; i < 4; i++)
            strcpy (choices[i].piece, choices[0].piece);
        strcpy (choices[4].piece, piece2a);
```

```
        strcat (choices[4].piece, piece1b);
        for (i = 5; i < 8; i++)
            strcpy (choices[i].piece, choices[4].piece);
```

The above source code performs a crossover at the site XPOINT between the two individuals I1 and I2 and makes four copies of each of the two resulting hybrid individuals. In the following FOR-loop the four choices (00, 01, 10, 11) for the new crossover site are set up. Each bit pair corresponds to one of the four bond angles (0°, 90°, 270°, 180°) between two fragments. Now the overlap of the hybrid fragments joined by each of the four bond angles can be calculated.

```
        for (k = 0; k < 2; k++)
            for (i = 0; i < 2; i++)
                for (j = 0; j < 2; j++)
                {
                    choices[k*4+i*2+j].piece[xpoint]   = i + '0';
                    choices[k*4+i*2+j].piece[xpoint+1] = j + '0';
                }
        for (i = 0; i < 8; i++)
```

Finally, the conformations with the smallest overlap are selected to replace their parents.

```
        gal = select_min_overlap (choices, 0, 4);
        boy = select_min_overlap (choices, 4, 8);
        Pack (choices[gal].piece, New[mom].Gene, STR_LENGTH);
        Pack (choices[boy].piece, New[dad].Gene, STR_LENGTH);

        if ((strcmp (choices[gal].piece, i1) != 0) ||
            (strcmp (choices[boy].piece, i2) != 0))
        {
            New[mom].Needs_evaluation = 1;
            New[dad].Needs_evaluation = 1;
        }
    }
    Trace("Crossover completed");
}
```

Mutation is applied with a probability of 0.1%. The remaining parameters for this application in Genesis are listed below:

```
       Experiments = 1
      Total Trials = 10000
   Population Size = 50
  Structure Length = 40
    Crossover Rate = 0.6
     Mutation Rate = 0.001
    Generation Gap = 1.0
    Scaling Window = 5
   Report Interval = 10
  Structures Saved = 10
 Max Gens w/o Eval = 4
     Dump Interval = 10
       Dumps Saved = 0
           Options = celD
       Random Seed = 123456789
          Rank Min = 0.75
```

The *fitness function* is quite elementary. Only orthogonally neighbouring hydrophobic residues that are not connected by a covalent bond contribute to the "energy" of a conformation. Each such interaction decrements[22] the fitness value by 1 energy unit. If one residue is placed on a grid point that was already occupied by another residue each such overlap increments the fitness value by a penalty sum of +100 energy units. As the fitness function is to be minimised there is a strong tendency to loose individuals with overlaps. The following source code must be saved in the file EVAL.C and compiled and linked with the remaining files of Genesis.

```
/* FILE EVAL.C */

#include <stdio.h>
#include <stdlib.h>

#define SEQ_LENGTH    20
#define X             ((SEQ_LENGTH + 1) * 2)
#define Y             ((SEQ_LENGTH + 1) * 2)
#define SEQUENCE      {1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
                       0, 1, 0, 1}
```

The binary sequence in the constant SEQUENCE corresponds to the primary structure B-W-B-W-W-B-B-W-B-W-W-B-W-B-B-W-W-B-W-B.

```
double eval (str, length, vect, genes)
char str[];              /* string representation         */
int length;              /* length of bit string          */
double vect[];           /* floating point representation */
int genes;               /* number of elements in vect    */
{
    int i, j;
    int x = X / 2;
    int y = Y / 2;
    double sum = 0.0;
    static int sequence[SEQ_LENGTH] = SEQUENCE;
    static int matrix[X][Y][3];

    for (i = 0; i < X; i++)
        for (j = 0; j < Y; j++)
            matrix[i][j][0] = matrix[i][j][1] = matrix[i][j][2] = 0;
    for (i = 0; i < SEQ_LENGTH; i++)
    {
        matrix[x][y][1] = i + 1;    /* holds the residue numbers */
        matrix[x][y][2]++;          /* counts overlaps */
        if (sequence[i])
            matrix[x][y][0] = 1;    /* determines hydrophobic or ... */
        else
            matrix[x][y][0] = -1;   /* ... hydrophilic */

        if (str[2*i] == '0')
            if (str[2*i+1] == '0')
                x += 1;
            else
                y += 1;
```

22 Remember: the lower the fitness value (energy) the better (more stable) a protein conformation.

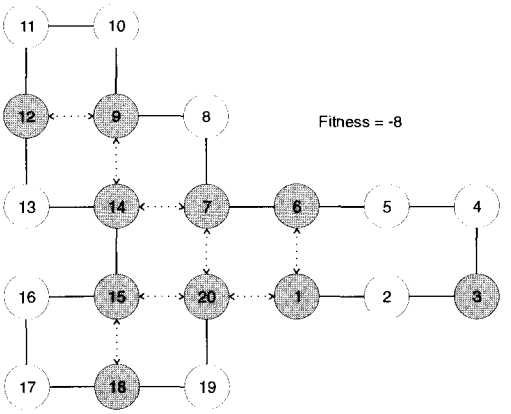


Figure 9: Near-Optimal Solution with Fitness -8 Energy Units.

```
            else
                if (str[2*i+1] == '0')
                    y -= 1;
                else
                    x -= 1;
    }
    for (i = 1; i < X - 1; i++)
        for (j = 1; j < Y - 1; j++)
        {
            if (matrix[i][j][2] > 1)
                sum += (matrix[i][j][2] - 1) * 200;
            if (matrix[i][j][0] == 1) /* look for neighboring */
            {                         /* hydrophobic residue  */
                if ((matrix[i-1][j][0] == 1) &&
                    (abs (matrix[i][j][1] - matrix[i-1][j][1]) > 2))
                    sum--;
                if ((matrix[i][j-1][0] == 1) &&
                    (abs (matrix[i][j][1] - matrix[i][j-1][1]) > 2))
                    sum--;
                if ((matrix[i+1][j][0] == 1) &&
                    (abs (matrix[i][j][1] - matrix[i+1][j][1]) > 2))
                     sum--;
                if ((matrix[i][j+1][0] == 1) &&
                    (abs (matrix[i][j][1] - matrix[i][j+1][1]) > 2))
                    sum--;
            }
        }
    sum /= 2.0;   /* because each hydroph. interaction is counted 2x */
    if (sum == 0.0)
        sum = 1;  /* Genesis cannot cope with fitness zero */
    return sum;
}
```

The following conformations were found by the genetic algorithm (Figures 9 and 10). One of them has the optimal fitness value for this application with -9 energy units.

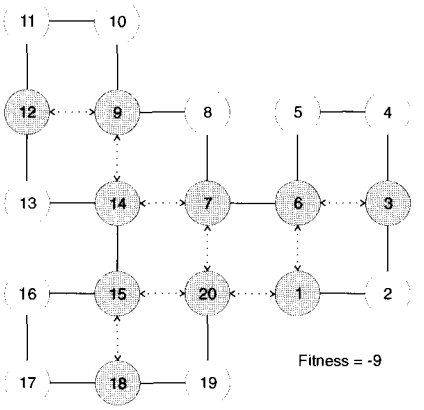


Figure 10: Optimal Solution with Fitness -9 Energy Units.

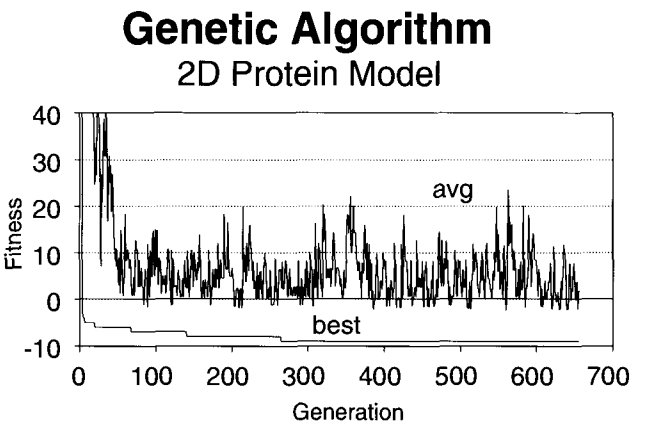


Figure 11: Performance of Genetic Algorithm on the 2-D Protein Model. The best individual was always passed on in this run, hence the monotonously decreasing fitness. The average fitness of all individuals also decreases but fluctuates considerably.

Figure 11 shows the performance of a typical run of the genetic algorithm on the 2D protein model. As the best individual is always propagated into the next generation (elitist option "e" in Genesis), the fitness value for the best individual of each generation decreases monotonously. The average fitness, however, fluctuates considerably because the genetic algorithm produces worse individuals all the time.

Table 3 shows some data on the performance of the GA. It can be seen that the number of trials becomes much smaller than the product of population size and number of generations. This means that some individuals are propagated without any changes or identical individuals are re-discovered which did not have to be evaluated again. 10,000 evaluations divided by 50 individuals in one generation gives 200 generations if every individual would have to be evaluated. In this run how-

Gens	Trials	Lost	Conv	Bias	Online	Offline	Best	Avg
0	50	0	0	0.55	730.24	138.00	98	730.24
1	80	0	0	0.56	683.31	123.00	98	602.04
2	111	0	0	0.57	628.90	115.89	97	513.98
3	141	0	1	0.59	585.48	111.87	97	416.00
4	171	1	1	0.60	547.30	106.92	-3	353.90
5	201	1	1	0.61	505.15	90.44	-4	298.10
6	232	1	1	0.61	474.90	77.82	-4	270.04
7	262	1	1	0.62	443.59	68.35	-5	212.02
8	292	1	1	0.62	414.65	60.81	-5	174.20
...								
17	564	0	1	0.67	263.06	29.07	-5	70.42
18	594	1	1	0.67	251.91	27.35	-5	46.66
19	624	1	2	0.69	241.33	25.80	-5	32.70
20	654	1	2	0.68	231.43	24.35	-6	24.68
21	684	1	2	0.68	223.12	23.02	-6	32.66
...								
65	2020	1	2	0.76	92.65	3.83	-6	6.70
66	2050	1	2	0.76	91.47	3.68	-6	8.70
67	2080	1	2	0.76	90.28	3.54	-6	6.66
68	2111	1	2	0.76	89.02	3.40	-7	2.74
69	2141	1	3	0.76	87.77	3.25	-7	0.84
70	2171	1	3	0.76	86.54	3.11	-7	0.94
71	2201	1	2	0.76	85.44	2.97	-7	4.82
...								
138	4243	1	6	0.77	48.17	-1.83	-7	8.34
139	4273	1	6	0.78	47.87	-1.86	-7	2.54
140	4303	1	3	0.77	47.55	-1.90	-7	4.42
141	4334	1	4	0.77	47.25	-1.94	-8	2.28
142	4365	1	4	0.77	47.04	-1.98	-8	10.44
143	4395	1	4	0.77	46.75	-2.03	-8	2.50
...								
262	8012	1	5	0.77	29.05	-4.72	-8	6.46
263	8043	2	6	0.77	28.98	-4.74	-8	6.38
264	8074	2	6	0.77	28.91	-4.75	-8	6.30
265	8105	2	6	0.77	28.81	-4.76	-9	2.34
266	8135	2	6	0.77	28.74	-4.78	-9	8.20
267	8165	2	7	0.76	28.63	-4.80	-9	2.26
268	8195	2	6	0.76	28.53	-4.81	-9	2.30
269	8225	2	6	0.76	28.46	-4.83	-9	4.28
270	8256	2	7	0.76	28.40	-4.84	-9	8.38

Table 3: Performance Criteria for the Genetic Algorithm.
"Gens" denotes the number of generations calculated. "Trials" is the number of invocations of the fitness function. "Lost" refers to the number of bit positions that are 100% identical over the whole population; "Conv" refers to those that are only in 95% of all individuals identical. "Bias" indicates the average convergence of all positions (theoretical minimum is 50%). "Online" is the mean of all fitness evaluations so far. "Offline" is the mean of the current best evaluations, i.e. those that are improvements over the average of the previous generation. "Best" is the best fitness detected so far and "Avg" is the average fitness of the current population. Some of data of this run were removed for brevity.

ever, 270 generations were performed. Less than 3 (or 8) bit positions remained 100% (or 95%) constant during the run. This defies any premature convergence and shows that the genetic algorithm is still improving on its individuals. A maximum average bias of 75% for any bit position at the end of the run substantiates this finding (i.e. on the average not more than 75% of all individuals in one generation have the same value at any bit position). As expected, online (calculated over all evaluations) and offline performance (calculated only for those better than the average fitness of the last generation) both decrease steadily along with the best fitness. The average fitness of the current population tends to fluctuate considerably because there are always a few individuals created with much worse fitness values.

R. Unger and J. Moult continue to show that the performance of the genetic algorithm is much more efficient than various Monte-Carlo strategies. The genetic algorithm arrives faster and with less computational effort at better fitness values than Monte-Carlo search. These results show that the genetic algorithm is certainly a useful search tool for the simplified protein folding model. The next step is now to extend the protein model to three dimensions and to make the fitness function more realistic. This approach will be discussed in the following sections.

5.2.2. Protein Folding Simulation by Force Field Optimisation

This section describes the application of a genetic algorithm to the problem of protein structure prediction[23,24,25] with a simple force field as the fitness function. It is a continuation of work presented earlier[26]. Similar research on genetic algorithms and protein folding was done independently by several groups world wide[27]. Genetic algorithms have been used to predict optimal sequences to fit structural constraints[28], to fold Crambin in the Amber force field[29] and Mellitin in an empirical, statistical potential[30], and to predict main chain folding patterns of small proteins based on secondary structure predictions[31].

23 G. E. Schulz, R. H. Schirmer, **Principles of Protein Structure,** Springer Verlag, 1979.
24 A. M. Lesk, **Protein Architecture – A Practical Approach,** IRL Press, 1991.
25 C. Branden, J. Tooze, **Introduction to Protein Structure,** Garland Publishing New York, 1991.
26 S. Schulze-Kremer, *Genetic Algorithms for Protein Tertiary Structure Prediction,* in **Parallel Problem Solving from Nature II,** (R. Männer, B. Manderick, eds.), North Holland, pp. 391-400, 1992.
27 For more information or to get in touch with researchers using genetic algorithms send an email to one of the following mailing lists: ga-molecule@interval.com, ga-list-request@aic.nrl.navy.mil or to Melanie Mitchell at mm@santafe.edu who keeps an extensive bibliography on applications of genetic algorithms in chemistry. Alternatively, try a search for "genetic algorithm" in gopher space, for example at gopher://veronica.sunet.se.
28 T. Dandekar, P. Argos, *Potential of genetic algorithms in protein folding and protein engineering simulations,* **Protein Engineering,** vol 5, no 7, pp. 637-645, 1992.
29 S. M. Le Grand, K. M. Merz, *The application of the genetic algorithm to the minimization of potential energy functions,* **The Journal of Global Optimization,** vol 3, pp.49-66, 1993.
30 S. Sun, *Reduced representation model of protein structure prediction: statistical potential and genetic algorithms,* **Protein Science,** vol 2, no 5, pp. 762-785, 1993.
31 T. Dandekar, P. Argos, *Folding the Main Chain of Small Proteins with the Genetic Algorithm,* **Journal of Molecular Biology,** vol 236, pp. 844-861, 1994.

In this section the individuals of the genetic algorithm are conformations of a protein and the fitness function is a simple force field. In the following, the representation formalism, the fitness function and the genetic operators are described. Then, the results of an *ab initio* prediction run and of an experiment for side chain placement for the protein Crambin will be discussed.

5.2.2.1. Representation Formalism

For every application of a genetic algorithm one has to decide on a representation formalism for the "genes". In this application, the so-called *hybrid approach* is taken[8]. This means that the genetic algorithm is configured to operate on numbers, not bit strings as in the original genetic algorithm. A hybrid representation is usually easier to implement and also facilitates the use of domain specific operators. However, three potential disadvantages are encountered:

1. Strictly speaking, the mathematical foundation of genetic algorithms holds only for binary representations, although some of the mathematical properties are also valid for a floating point representation.
2. Binary representations run faster in many applications.
3. An additional encoding/decoding process may be required to map numbers onto bit strings.

It is not the principal goal of this application to find the single optimal conformation of a protein based on a force field but to generate a *small set of native-like conformations*. For this task the genetic algorithm is an appropriate tool. For a hybrid representation of proteins one can use Cartesian coordinates, torsion angles, rotamers or a simplified model of residues.

For a representation in Cartesian coordinates the 3-dimensional coordinates of all atoms in a protein are recorded. This representation has the advantage of being easily converted to and from the 3-dimensional conformation of a protein. However, it has the disadvantage that a mutation operator would in most instances create invalid protein conformations where some atoms lie too far apart or collide. Therefore a filter is needed which eliminates invalid individuals. Because such a filter would consume a disproportionate large amount of CPU time a Cartesian coordinate representation considerably slows down the search process of a genetic algorithm.

Another representation model is by torsion angles. Here, a protein is described by a set of torsion angles under the assumption of constant standard binding geometries. Bond lengths and bond angles are taken to be constant and cannot be changed by the genetic algorithm. This assumption is certainly a simplification of the real situation where bond length and bond angle to some extent depend on the environment of an atom. However, torsion angles provide enough degrees of freedom to represent any native conformation with only small r.m.s.[32] deviations.

Special to the torsion angle representation is the fact that even small changes in the ϕ (phi) / ψ (psi) angles can induce large changes in the overall conformation.

32 r.m.s. = root mean square deviation; two conformations are superimposed and the square root is calculated from the sum of the squares of the distances between corresponding atoms.

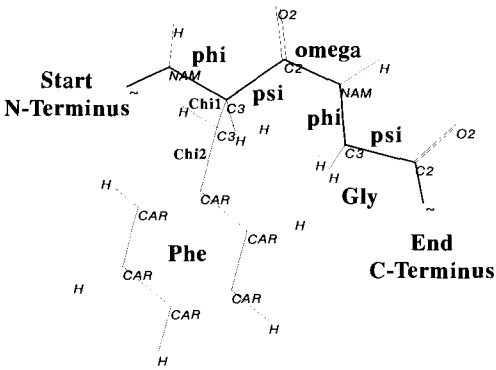


Figure 12: Torsion Angles $\varphi, \psi, \omega, \chi_1$ and χ_2 of the di-peptide phe-gly.

This is useful when creating variability within a population at the beginning of a run. Figure 12 explains the definition of the torsion angles ϕ, ψ, ω (omega), $\chi 1$ (chi1) and $\chi 2$ (chi2). A small fragment taken from a hypothetical protein is shown. Two basic building blocks, the amino acids phenylalanine (Phe) and glycine (Gly), are drawn as wire frame models. Atoms are labelled with their chemical symbols. Bonds in bold print indicate the backbone. The labels of torsion angles are placed next to their rotatable bonds.

In the present work the torsion angle representation is used. Torsion angles of 129 proteins from the Brookhaven database[33] (PDB) were statistically analysed for the definition of the MUTATE operator. The frequency of each torsion angle in intervals of 10° was determined and the ten most frequently occurring intervals are made available for substitution of individual torsion angles by the MUTATE operator. At the beginning of the run, individuals were initialised with either a completely extended conformation where all torsion angles are 180° or by a random selection from the ten most frequently occurring intervals of each torsion angle. For the ω torsion angle the constant value of 180° was used because of the rigidity of the peptide bond between the atoms C_i and N_{i+1}. A statistical analysis of ω angles shows that with the exception of proline average deviations from the mean of 180° occur rather frequently up to 5°, but only in rare cases up to 15°.

The genetic operators in this application operate on the torsion angle representation but the fitness function requires a protein conformation to be expressed in Cartesian coordinates. For the implementation of a conversion program bond angles were taken from the molecular modelling software Alchemy[34] and bond lengths from the program Charmm[35]. Either a complete form with explicit hy-

33 F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, E. F. Meyer Jr., M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, M. Tasumi, *The Protein Data Bank: A Computer-based Archival File for Macromolecular Structures*, **Journal of Molecular Biology**, 112, pp. 535-542, 1977.

34 J. G. Vinter, A. Davis, M. R. Saunders, *Strategic approaches to drug design. An integrated software framework for molecular modelling*, **Journal of Computer-Aided Molecular Design**, vol 1, pp. 31-51, 1987.

35 B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, M. Karplus, *Charmm: A program for Macromolecular Energy, Minimization and Dynamics Calculations*, **Journal of Computational Chemistry**, vol 4, no 2, pp. 187-217, 1983.

drogen atoms or the so-called extended atom representation with small groups of atoms represented as "super-atoms" can be calculated. One conformation of a protein is encoded as an array of structures of the C programming language. The number of structures equals the number of residues in the protein. Each structure includes a three letter identifier of the residue type and ten floating point numbers for the torsion angles ϕ, ψ, ω, χ_1, χ_2, χ_3, χ_4, χ_5, χ_6, and χ_7. For residues with less than seven side chain torsion angles the extra fields are filled with a default value. The main chain torsion angle ω was kept constant at 180°.

5.2.2.2. Fitness Function

In this application a simple steric potential energy function was chosen as the fitness function (i.e. the objective function to be minimised). It is very difficult to find the global optimum of any potential energy function because of the large number of degrees of freedom even for a protein of average size. In general, molecules with n atoms have $3n - 6$ degrees of freedom. For the case of a medium-sized protein of 100 residues this amounts to:

$$((100 \text{ residues} \cdot \text{approximately 20 atoms per residue}) \cdot 3) - 6 = 5994$$

degrees of freedom. Systems of equations with this number of free variables are analytically intractable today. Empirical efforts to heuristically find the optimum are almost as difficult[36]. If there are no constraints for the conformation of a protein and only its primary structure is given the number of conformations for a protein of medium size (100 residues) can be approximated to:

$$(5 \text{ torsion angles per residue} \cdot 5 \text{ likely values per torsion angle})^{100} = 25^{100}.$$

This mean that in the worst case 25^{100} conformations would have to be evaluated to find the global optimum. This is clearly beyond the capacity of today's and tomorrow's super computers. As can be seen from a number of previous applications genetic algorithms were able to find sub-optimal solutions to problems with an equally large search space[37,38,39]. Sub-optimal in this context means that it cannot be proven that the solutions generated by the genetic algorithm do in fact include an optimal solution but that some of the results generated by the genetic algorithm practically surpassed any previously known solution. This can be of much help in non-polynomial complete problems where no analytical solution of the problem is available.

36 J. T. Ngo, J. Marks, *Computational complexity of a problem in molecular-structure prediction*, **Protein Engineering**, vol 5, no 4, pp. 313-321, 1992.

37 L. Davis, (ed.) **Handbook of Genetic Algorithms**, van Nostrand Reinhold, New York, 1991.

38 C. B. Lucasius, G. Kateman, *Application of Genetic Algorithms to Chemometrics*, **Proceedings 3rd International Conference on Genetic Algorithms**, (J. D. Schaffer, ed.), Morgan Kaufmann Publishers, San Mateo, CA, pp. 170-176, 1989.

39 P. Tuffery, C. Etchebest, S. Hazout, R. Lavery, *A new approach to the rapid determination of protein side chain conformations*, **J. Biomol. Struct. Dyn.**, vol 8, no 6, pp. 1267-1289, 1991.

5.2.2.3. Conformational Energy

The steric potential energy function was adapted from the program Charmm. The total energy of a protein in solution is the sum of the expressions for E_{bond} (bond length potential), E_{phi} (bond angle potential), E_{tor} (torsion angle potential), E_{impr} (improper torsion angle potential), E_{vdW} (van der Waals pair interactions), E_{el} (electrostatic potential), E_H (hydrogen bonds), and of two expressions for interaction with the solvent, E_{cr} and E_{cphi} :

$$E = E_{bond} + E_{phi} + E_{tor} + E_{impr} + E_{vdW} + E_{el} + E_H + E_{cr} + E_{cphi}.$$

Here we assume constant bond lengths and bond angles. The expressions for E_{bond}, E_{phi} and E_{impr} are therefore constant for different conformations of the same protein. The expression E_H was omitted because it would have required the exclusion of the effect of hydrogen bonds from the expressions for E_{vdW} and E_{el}. This, however, was not done by the authors of Charmm in their version v.21 of the program. In all runs, folding was simulated in vacuum with no ligands or solvent, i.e. E_{cr} and E_{cphi} are constant. This is certainly a crude simplification of the real situation but nevertheless more detailed than the 2-D protein model in the previous section. Thus, the potential energy function simplifies to:

$$E = E_{tor} + E_{vdW} + E_{el}.$$

Test runs showed that if only the three expressions E_{tor}, E_{vdW} and E_{el} are used there would not be enough force to drive the protein to a compact folded state. An exact solution to this problem requires the consideration of entropy. The calculation of the entropy difference between a folded and unfolded state is based on the interactions between protein and solvent. Unfortunately, it is not yet possible to routinely calculate an accurate model of those interactions. It was therefore decided to introduce an *ad hoc* pseudo entropic term E_{pe} that drives the protein to a globular state. The analysis of a number of globular proteins reveals the following empirical relation between the number of residues and the diameter:

$$\textit{expected diameter} = 8 \cdot \sqrt[3]{\textit{length}}\ [\text{Å}].$$

The pseudo entropic term E_{pe} for a conformation is a function of its actual diameter. The diameter is defined to be the largest distance between any Cα atoms in one conformation. An exponential of the difference between actual and expected diameter is added to the potential energy if that difference is less than 15Å. If the difference is greater than 15Å a fixed amount of energy is added (10^{10} kcal/mol) to avoid exponential overflow. If the actual diameter of an individual is smaller than the expected diameter E_{pe} is set to zero. The net result is that extended conformations have larger energy values and are therefore less fit for reproduction than globular conformations.

$$E_{pe} = 4^{(\textit{actual diameter - expected diameter})} \quad [\text{kcal/mol}].$$

Occasionally, if two atoms are very close the E_{vdW} term can become very large. The maximum value for E_{vdW} in this case is 10^{10} kcal/mol and the expressions for E_{el} and E_{tor} are not calculated. Runs were performed with the potential energy function E as described above where lower fitness values mean fitter individuals and with

a variant, where the four expressions E_{tor}, E_{vdW}, E_{el} and E_{pe} were given individual weights. The results were similar in all cases. Especially, scaling down the dominant effect of electrostatic interactions did not change the results (see below).

5.2.2.4. Genetic Operators

In order to combine individuals of one generation to produce new offspring nature as well as genetic algorithms apply several genetic operators. In the present work, individuals are protein conformations represented by a set of torsion angles under the assumption of constant standard binding geometries. Three operators are invented to modify these individuals: MUTATE, VARIATE and CROSSOVER. The decision about the application of an operator is made during run time and can be controlled by various parameters.

MUTATE. The first operator is the MUTATE operator. If MUTATE gets activated for a particular torsion angle, this angle will be replaced by a random choice of one of the ten most frequently occurring values for that type of residue. The decision whether a torsion angle will be modified by MUTATE is made independently for each torsion angle in a protein. A random number between 0 and 1 is generated and if this number is greater than the MUTATE parameter at that time, MUTATE is applied. The MUTATE parameter can change dynamically during a run. The torsion angle values that MUTATE can choose from come from a statistical analysis of 129 proteins from PDB. The number of instances in each of 36 10°-intervals was counted for each torsion angle. The ten most frequent intervals, each represented by its left boundary, are available for substitution.

VARIATE. The VARIATE operator consists of three components: the 1°, 5° and 10° operator. Independently and after application of the MUTATE operator for each torsion angle in a protein two decisions are made: first, whether the VARIATE operator will be applied and, second, if so which of the three components shall be selected. The VARIATE operator increments or decrements (always an independent random chance of 1:1) the torsion angle by 1°, 5° or 10°. Care is taken that the range of torsion angles does not exceed the [-180°, 180°] interval. The probability of applying this operator is controlled by the VARIATE parameter which can change dynamically during run time. Similarly, three additional parameters control the probability for choosing among the three components. Alternatively, instead of three discrete increments a Gaussian uniformly distributed increment between -10° and +10° can be used.

CROSSOVER. The CROSSOVER operator has two components: the *two point crossover* and the *uniform crossover*. CROSSOVER is applied to two individuals independently of the MUTATE and VARIATE operators. First, individuals of the parent generation, possibly modified by MUTATE and VARIATE, are randomly grouped pairwise. For each pair, an independent decision is made whether or not to apply the CROSSOVER operator. The probability of this is controlled by a CROSSOVER parameter which can change dynamically during run time. If the decision is "no", the two individuals are not further modified and added to the list of offspring. If the decision is "yes", a choice between the two point crossover and the uniform crossover must be made. This decision is controlled by two other parameters that can also be changed during run time. The *two point crossover* randomly selects two

residues on one of the individuals. Then the fragment between the two residues is exchanged with the corresponding fragment of the second individual. Alternatively, *uniform crossover* decides independently for each residue whether or not to exchange the torsion angles of that residue. The probability for an exchange is then always 50%.

Parameterization. As mentioned in the previous paragraphs, there are a number of parameters that control the run time behaviour of this genetic algorithm application. The parameter values used for the experiments that will be presented in the "Results" section below are summarised in Table 4. The main chain torsion angle ω was kept constant at 180°. The initial generation was created by a random selection of torsion angles from a list of the ten most frequently occurring values for each angle. Ten individuals are in one generation. The genetic algorithm was halted after 1000 generations. At the start of the run, the probability for a torsion angle to be modified by the MUTATE operator is 80%, at the end of the run it becomes 20%. In between the probability decreases linearly with the number of generations. In contrast, the probability of applying the VARIATE operator increases from 20% at the beginning to 70% at the end of the run. The 10° component of the VARIATE operator is dominant at the start of the run (60%), whereas it is the 1° component at the end (80%). Likewise, the chance of performing a CROSSOVER rises from 10% to 70%. At the beginning of the run mainly uniform CROSSOVER is applied (90%), at the end it is mainly two point CROSSOVER (90%). This parameter setting uses a small number of individuals but runs over a large number of generations. This keeps computation time low while allowing a maximum number of crossover events. At the beginning of the run MUTATE and uniform CROSSOVER are applied most of the time to create some variety in the population so that many different regions of the search space are covered. At the end of the run the 1° component of the VARIATE operator dominates the scene. This is intended for fine tuning those conformations that have survived the selection pressure of evolution so far.

Generation Replacement. There are different ways of selecting the individuals for the next generation. Given the constraint that the number of individuals should remain constant some individuals have to be discarded. Transition between generations can be done by *total replacement, elitist replacement* or *steady state replacement*. For *total replacement* only the newly created offspring enter the next generation and the parents of the previous generation are completely discarded. This has the disadvantage that a fit parent can be lost even if it only produces bad offspring once. With *elitist replacement* all parents and offspring of one generation are sorted according to their fitness. If the size of the population is n, then the n fittest individuals are selected as parents for the following generation. This mode has been used here. Another variant is *steady state replacement* where two individuals are selected from the population based on their fitness and then modified by mutation and crossover. They are then used to replace their parents.

5.2.2.5. *Ab initio* Prediction Results

A prototype of a genetic algorithm with the representation, fitness function and operators as described above has been implemented. To evaluate the *ab initio* prediction performance of the genetic algorithm the sequence of Crambin was given to

Parameter	Value
ω angle constant 180°	on
initialise start generation	random
number of individuals	10
number of generations	1000
MUTATE (start)	80%
MUTATE (end)	20%
VARIATE (start)	20%
VARIATE (end)	70%
VARIATE (start 10°)	60%
VARIATE (end 10°)	0%
VARIATE (start 5°)	30%
VARIATE (end 5°)	20%
VARIATE (start 1°)	10%
VARIATE (end 1°)	80%
CROSSOVER (start)	70%
CROSSOVER (end)	10%
CROSSOVER (start uniform)	90%
CROSSOVER (end uniform)	10%
CROSSOVER (start two point)	10%
CROSSOVER (end two point)	90%

Table 4: Run Time Parameters.

the program. Crambin is a plant seed protein from the cabbage *Crambe abyssinica.* Its structure was determined by W. A. Hendrickson and M. M. Teeter[40] to a resolution of 1.5Å (Figures 13 and 14). Crambin has a strong amphiphilic character which makes its conformation especially difficult to predict. However, because of its good resolution and small size of 46 residues it was decided to use Crambin as a first candidate for prediction. The protein structures are again displayed in stereo projection. If the observer manages to look cross eyed at the diagram in a way that superimposes both halves a 3-dimensional image can be perceived.

Figure 15 shows two of the ten individuals in the last generation of the genetic algorithm. None of the ten individuals shows significant structural similarity to the native Crambin conformation. This can be confirmed by superimposing the generated structures with the native conformation. Table 5 shows the r.m.s. differences between all ten individuals and the native conformation. All values are in the range of 9Å which rejects any significant structural homology.

Although the genetic algorithm did not produce native-like conformations of Crambin, the generated backbone conformations could be those of a protein, i.e. they have no knots or unreasonably protruding extensions. The conformational results alone would indicate a complete failure of the genetic algorithm approach to conformational search but let us have a look at the energies in the final generation (Table 6). All individuals have a much lower energy than native Crambin in the

40 W. A. Hendrickson, M. M. Teeter, *Structure of the Hydrophobic Protein Crambin Determined directly from the Anomalous Scattering of Sulphur*, **Nature**, vol 290, pp. 107, 1981.

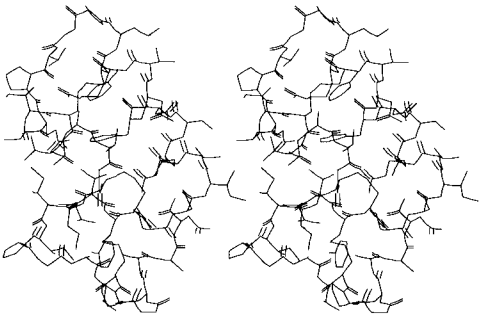


Figure 13: Stereoprojection of Crambin with Side Chains.

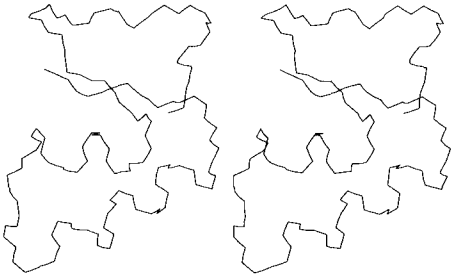


Figure 14: Stereoprojection of Crambin without Side Chains.

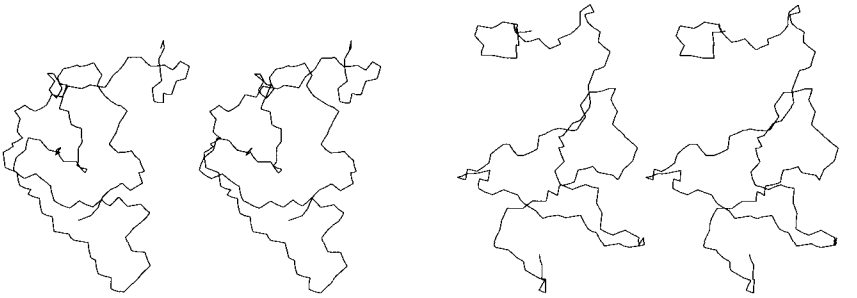


Figure 15: Two Conformations Generated by the Genetic Algorithm (Stereoprojection).

same force field. That means that the genetic algorithm actually achieved a substantial optimisation but that the current fitness function was not a good indicator of "nativeness" of a conformation.

It is obvious that all individuals generated by the genetic algorithm have a much higher electrostatic potential than native Crambin. There are three reasons for this.

- Electrostatic interactions are able to contribute larger amounts of stabilising energy than any of the other fitness components.
- Crambin has six partially charged residues that were not neutralised in this experiment.

Individual	R.m.s.	Individual	R.m.s.
P1	10.07 Å	P6	10.31 Å
P2	9.74 Å	P7	9.45 Å
P3	9.15 Å	P8	10.18 Å
P4	10.14 Å	P9	9.37 Å
P5	9.95 Å	P10	8.84 Å

Table 5: R.m.s. Deviations to Native Crambin.
The ten individuals of the last generation were measured against the native conformation of Crambin. R.m.s. values of around 9Å for a small protein as Crambin exclude any significant structural similarity.

Individual	E_{vdw}	E_{el}	E_{tor}	E_{pe}	E_{total}
P1	-14.9	-2434.5	74.1	75.2	-2336.5
P2	-2.9	-2431.6	76.3	77.4	-2320.8
P3	78.5	-2447.4	79.6	80.7	-2316.1
P4	-11.1	-2409.7	81.8	82.9	-2313.7
P5	83.0	-2440.6	84.1	85.2	-2308.5
P6	-12.3	-2403.8	86.1	87.2	-2303.7
P7	88.3	-2470.8	89.4	90.5	-2297.6
P8	-12.2	-2401.0	91.6	92.7	-2293.7
P9	93.7	-2404.5	94.8	95.9	-2289.1
P10	96.0	-2462.8	97.1	98.2	-2287.5
Crambin	-12.8	11.4	60.9	1.7	61.2

Table 6: Steric Energies in the Last Generation.
For each individual the van der Waals energy (E_{vdW}), electrostatic energy (E_{el}), torsion energy (E_{tor}), pseudo entropic energy (E_{pe}) and the sum of all terms (E_{total}) is shown. For comparison the values for native Crambin in the same force field are listed.

- The genetic algorithm favoured individuals with lowest total energy which in this case was most easily achieved by optimising electrostatic contributions.

The final generation of only ten individuals contained two fundamentally different families of structures (class 1: P1, P2, P4, P5, P6, P8, P9) and (class 2: P3, P7, P10). Members of one class have a r.m.s. deviation of about 2 Å among themselves but differ from members of the other class by about 9 Å.

Taking into account the small population size, the significant increase in total energy of the individuals generated by the genetic algorithm and the fact that the final generation contained two substantially different classes of conformations with very similar energies, one is led to the conclusion that the search performance of the genetic algorithm was not that bad at all. What remains a problem is to find a better fitness function that actually guides the genetic algorithm to native-like conformations. As the only criterion currently known to determine native conformation is the

free energy, the difficulty of this approach becomes obvious. One possible way to cope with the problem of inadequate fitness functions is to combine other heuristic criteria together with force field components in a multi-value vector fitness function. Before we turn to that approach let us first examine the performance of the current version for side chain placement.

5.2.2.6. Side Chain Placement

Crystallographers often face the problem of positioning the side chains of a protein when the primary structure and the conformation of the backbone is known. At present, there is no method that automatically does side chain placement with sufficiently high accuracy for routine practical use. Although the side chain placement problem is conceptually easier than *ab initio* tertiary structure prediction it is still too complex for analytical treatment.

The genetic algorithm approach as described above can be used for side chain placement. The torsion angles φ, ψ, and ω simply have to be kept constant for a given backbone. Side chain placement by the genetic algorithm was done for Crambin. For each five residues, a superposition of the native and predicted conformation is shown in stereo projection graphs in Figure 16. As can be seen, the predictions agree quite well with the native conformation in most cases. The overall r.m.s. difference in this example is 1.86Å. This is not as good as but comparable to the results from a simulated annealing approach[41] (1.65Å) and a heuristic approach[42] (1.48Å).

It must be emphasised that these runs were done without optimising either the force field parameters of the fitness function or the run time parameters of the genetic algorithm. From a more elaborate and fine-tuned experiment even better results should be expected.

5.2.3. Multi-Criteria Optimisation of Protein Conformations

In this section we will introduce additional fitness criteria for the protein folding application with genetic algorithms. The rationale is that more information about genuine protein conformations should improve the fitness function to guide the genetic algorithm towards native-like conformations. Some properties of protein conformations can be used as additional fitness components whereas others can be incorporated into genetic operators (e.g. constraints from the Ramachandran plot). For such an extended fitness function several incommensurable quantities will have to be combined: energy, preferred torsion angles, secondary structure propensities or distributions of polar and hydrophobic residues. This creates the problem of how to combine the different fitness contributions to arrive at the total fitness of a single individual. Simple summation of different components has the disadvantage that components with larger numbers would dominate the fitness function whether or

41 C. Lee, S. Subbiah, *Prediction of protein side chain conformation by packing optimization*, **Journal of Molecular Biology**, no 217, pp. 373-388, 1991.

42 P. Tuffery, C. Etchebest, S. Hazout, R. Lavery, *A new approach to the rapid determination of protein side chain conformations*, **J. Biomol. Struct. Dyn.**, vol 8, no 6, pp. 1267-1289, 1991.

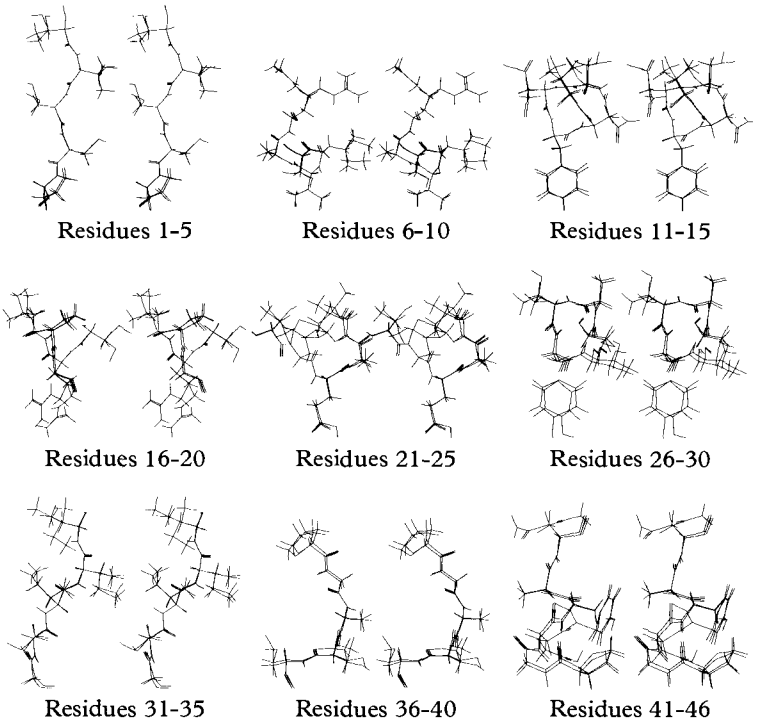


Figure 16: Side Chain Placement Results.
A spatial superposition in stereoscopic wire frame diagrams is show for every five residues of Crambin and the corresponding fragment generated by a genetic algorithm. The amino acid sequence of Crambin in one letter code is TTCCP SIVAR SNFNV CRLPG TPEAI CATYT GCIII PGATC PGDYA N.

not they are important or of any significance at all for a particular conformation. To cope with this difficulty individual weights for each of the components could be introduced. But this creates another problem. How should one determine useful values for these weights? As there is no general theory known for the proper weighting of each fitness component the only way is to try different combinations of values and evaluate them by their performance of a genetic algorithm on test proteins with known conformations. However, even for a small number of fitness components a large number of combinations of weights arises which requires as many test runs for evaluation. Also, "expensive" fitness components as the van der Waals energy need considerable computation time. In this work[43] two measures were taken to deal with this situation:

43 S. Schulze-Kremer, A. Levin, *Search for Protein Conformations with a Parallel Genetic Algorithm: Evaluation of a Vector Fitness Function*, Final Report and Technical Documentation of Protein Folding Application (Work package 1), **European ESPRIT project # 6857 PAPAGENA**, Brainware GmbH August 1994.

- Different fitness components are not arithmetically added to produce a single numerical fitness value but they are combined in a vector. This means that each fitness component is individually carried along the whole evaluation process and is always available explicitly.
- Parallel processing is employed to evaluate all individuals of one generation in parallel. For populations of 20 to 60 individuals this gave a speed-up of about 20 fold compared to small single-processor workstations.

5.2.3.1. Vector Fitness Function

In this application two versions of a fitness function are used. One version is a scalar fitness function that calculates the r.m.s.-deviation of a newly generated individual from the known conformation of the test protein. This geometric measure should guide the genetic algorithm directly to the desired solution but it is only available for proteins with a known conformation. R.m.s.-deviation is calculated as follows:

$$\text{r.m.s.} = \sqrt{\frac{\sum_{i}^{N} (|\vec{u}_i - \vec{v}_i|)^2}{N}}.$$

Here i is the index over all corresponding N atoms in the two structures to be compared, in this case the conformation of an individual ($\vec{u}_i$) in the current population and the known, actual structure ($\vec{v}_i$) of the test protein. The squares of the distances between the vectors $\vec{u}_i$ and $\vec{v}_i$ of corresponding atoms are summed, normalised and the square root is taken. The result is a measure of how much each atom in the individual deviates on average from its true position. R.m.s. values of 0 – 3 Å signify strong structural similarity; values of 4 – 6 Å denote weak structural similarity whereas for small proteins r.m.s.-values over 6 Å mean that probably not even the backbone folding pattern is similar in both conformations.

The other version of the fitness function is a vector of several fitness components which will be explained in the following paragraphs. This multi-value vector fitness function includes the following components:

$$fitness = \begin{pmatrix} r.m.s. \\ E_{tor} \\ E_{vdw} \\ E_{el} \\ E_{pe} \\ polar \\ hydro \\ scatter \\ solvent \\ Crippen \\ clash \end{pmatrix}$$

R.m.s. is the r.m.s.-deviation as described above. It can only be calculated in test runs with the protein conformation known beforehand. For the multi-value vector fitness function this measure was calculated for each individual to see how close the genetic algorithm came to the known structure. In these runs, however, the r.m.s.-

measure was *not* used in the offspring selection process. Selection was done *only* based on the remaining ten fitness components and a Pareto selection algorithm which will be described below.

E_{tor} is the torsion energy of a conformation based on the force field data of the Charmm force field v.21 with k and n as force field constants depending on the type of atom and ϕ as the torsion angle:

$$E_{tor} = |k_\phi| - k_\phi \cos(n\phi)$$

E_{vdw} is the van der Waals energy (also called Lennard-Jones potential) with A and B as force field constants depending on the type of atom and r as the distance between two atoms in one molecule. The indices i and j for the two atoms may not have identical values and each pair is counted only once:

$$E_{vdw} = \sum_{i<j} \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right)$$

E_{el} is the electrostatic energy between two atoms with $q_{i,j}$ as the partial charges of the two atoms i and j and r as the distance between them:

$$E_{el} = \sum_{i<j} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$

E_{pe} is a measure to promote compact folding patterns. The expected diameter of a protein can be estimated by a number of techniques. A penalty energy term is then calculated as follows:

$$E_{pe} = 4^{(actual\ diameter\ -\ expected\ diameter)}$$

Polar is a measure that favours polar residues on the protein surface but not in the core. Because all fitness contributions should be minimised a factor of minus one is required before the sum. The larger the distances $\vec{u}_i$ of polar residues to the centre of the protein, the better a conformation and the more negative the value of *polar*. If residue i is one of k polar residues (any of Arg, Lys, Asn, Asp, Glu, or Gln) in a protein of length N residues and with $\vec{s}$ as the centre of gravity, then the polar fitness contribution is calculated as follows:

$$polar = \frac{-\sum_i^N |\vec{u}_i - \vec{s}|}{k}$$

Hydro is a similar measure that favours hydrophobic residues (Ala, Val, Ile, Leu, Phe, Pro, Trp) in the core of a protein, whereas *scatter* promotes compact folds as it adds up the distances over all Cα atoms irrespective of amino acid type:

$$hydro = \frac{\sum_i^N |\vec{v}_i - \vec{s}|}{k}, \qquad scatter = \frac{\sum_i^N |\vec{v}_i - \vec{s}|}{N}.$$

Solvent is the solvent accessible surface of a conformation in Å^2. It is calculated by a surface triangulation method.

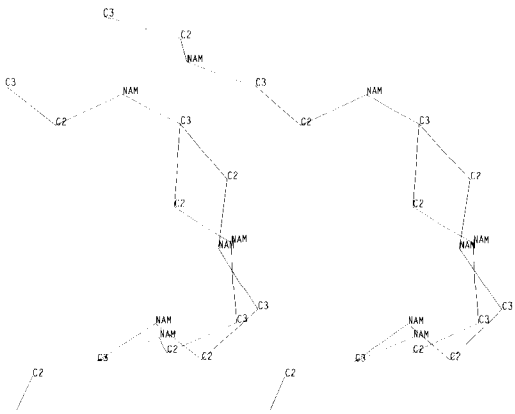


Figure 17: Backbone Conformation changed by LOCAL TWIST Operator. Stereoprojection of a portion of three residues and an alternative fold found by the LOCAL TWIST operator.

Crippen is an empirical, statistical potential developed by G. Crippen[44]. It is summed over all pairs of atoms that interact within a certain distance.

Clash is a term that counts the number of atomic collisions where any two atoms come closer than 3.8Å to each other. This fitness term can be used to approximate the effect of the van der Waals energy at small distances but at only a fraction of the computational cost:

$$clash = \sum_{i=1}^{N} \sum_{j=i+1}^{N} overlap(i,j) \text{ with } overlap(i,j) = \begin{cases} 0 & \text{if } dist(i,j) \geq 3.8\text{Å} \\ 1 & \text{if } dist(i,j) < 3.8\text{Å} \end{cases}$$

5.2.3.2. Specialised Genetic Operators

LOCAL TWIST Operator. The LOCAL TWIST operator introduces local conformation changes by performing the ring closure algorithm for polymers[45] of N. Go and H. A. Scheraga for three consecutive amino acid residues (Figure 17). This algorithm was originally implemented in the RING.FOR program[46] in a general way that operated on six adjacent dihedral angles to bridge a gap with bonds of defined length and bond angle. The application of this algorithm for a polypeptide required translation of the program into the C programming language and some alterations to the program to account for the intermitting rigid ω torsion angle.

The basic concept of the ring closure algorithm is to find suitable values for ϕ_1 that satisfy the following equation:

$$g(\phi_1) = u^{+} T_\alpha R_{\phi 1} T_\beta R_{\psi 1+\pi} T_\alpha R_{\phi 2} T_\beta R_{\psi 2+\pi} T_\alpha e_1 - \cos(\beta) = 0.$$

44 N. M. Maiorov, G. M. Crippen, *Contact potential that recognizes the correct folding of globular proteins*, **Journal of Molecular Biology**, vol 227, pp. 876 – 888, 1992.

45 N. Go, H. A. Scheraga, *Ring Closure and Local Conformational Deformations of Chain Molecules*, **Macromolecules**, vol 3, pp. 178-187, 1970.

46 Quantum Chemical Exchange Program (QCEP) program No. QCMP 046.

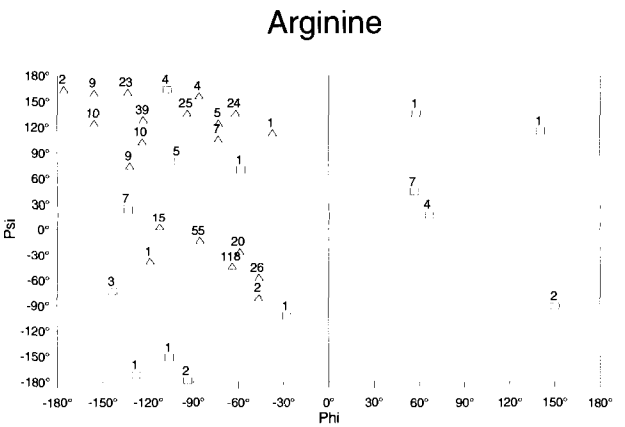


Figure 18: 34 ϕ/ψ Clusters for Arginine.
There are 14 small clusters of the first pass with less than 10 pairs (shown as boxes) and 20 large clusters of the remaining pairs in the second pass (triangles).

Here, u^+ (transposed) and e_1 are vectors and T and R are several translation and rotation matrices that define the constraints of a local conformation change. The angle β describes the rigid geometry of a peptide bond and ϕ_1 is the first backbone torsion angle in sequence to be modified. The search for suitable values of ϕ_1 involves repeated numerical approximations and is therefore rather time consuming. Hence, it was decided to distribute the LOCAL TWIST operator over several processors on a parallel computer[47] so that the calculations can be carried out in parallel for all individuals. In test runs with r.m.s.-deviation to the native conformation as the fitness function the LOCAL TWIST operator led to significant improvements in prediction accuracy and also to a substantial decrease in overall computation time.

Preferred Backbone Conformations. The MUTATE operator of the previous chapter is rather crude because it always uses the left boundary of one of the ten most frequently occurring 10°-intervals for a torsion angle. To improve the chance of selecting favourable values for the backbone torsion angles ϕ and ψ a cluster analysis with a modified nearest neighbour algorithm[48] was performed for the main chain torsion angles of 66 proteins:

1. Cluster all ϕ/ψ pairs for each amino acid until 21 clusters are formed.
2. Collect all clusters with less than 10 pairs and add the centre of each cluster to the set of ϕ/ψ pairs to be used by the MUTATE operator.
3. Repeat the clustering procedure with only the ϕ/ψ pairs from the clusters with at least 10 pairs in step 2 and let the clustering program run again until 21 clusters are formed. The centres of all new clusters complete the list of ϕ/ψ pairs that MUTATE uses when substituting individual torsion angles.

47 Intel Paragon with 98 x i860 processors owned by Parallab, University of Bergen, Norway.
48 S. Y. Lu, K. S. Fu, *A sentence-to-sentence clustering procedure for pattern analysis*, **IEEE Transactions on Systems, Man and Cybernatics SMC**, vol 8, pp. 381 – 389, 1978.

Secondary Structure	ϕ_l	ϕ_u	ψ_l	ψ_u	ϕ_{exact}	ψ_{exact}
α-helix (narrow interval)	-57°	-62°	-41°	-47°	-57°	-47°
α-helix (broad interval)	-30°	-120°	10°	-90°		
β-strand (narrow interval)	-119°	-139°	135°	113°	-130°	125°
β-strand (broad interval)	-50°	-180°	180°	80°		

Table 7: Boundaries for Main Chain Torsion Angles in Secondary Structures. ϕ_l, ψ_l and ϕ_u, ψ_u are lower and upper values of the main chain torsion angles in the respective secondary structure. ψ_{exact}, and ψ_{exact} are values for an idealised standard geometry. For β-strands the values are an average of parallel and antiparallel strands.

This algorithm first identifies small clusters with only few examples in more detail and then clusters more densely populated areas with a finer resolution than a single clustering would do in one pass. Figure 18 shows the centres of 34 clusters for arginine.

Secondary Structure. In addition to a more accurate selection of preferable main chain torsion angles predictions of secondary structure were used to reduce the search space. Two issues arise which must be considered:

1. Which secondary structure prediction algorithm should one rely on?
2. Which torsion angles should be used for the predicted secondary structures?

The first question was addressed by assembling a consensus prediction from two different methods: the PHD artificial neural network[49] and a statistical analysis which uses information theory[50,51]. For the second question there are two alternate solutions. One alternative is to use torsion angles of idealised α-helices and β-strands, another is to constrain torsion angles of the predicted secondary structures to an interval that includes the conformation with idealised geometry. The corresponding torsion angles are shown in Table 7.

5.2.3.3. Results

Using the genetic algorithm application as described in the previous section produced the following results. Figure 19 shows the best individual of the final generation of a run with a population of 30 individuals, the LOCAL TWIST operator in effect and r.m.s.-deviation as the only fitness component[52]. For Crambin, the final r.m.s.-deviation of the conformation generated by the genetic algorithm is 1.08 Å, which is well within the range of the best resolution from X-ray or NMR structure elucidation experiments. Another run with the same parameters produced an in-

49 B. Rost, C. Sander, *Prediction of protein secondary structure at better than 70% accuracy*, **Journal of Molecular Biology**, vol 232, pp. 584 – 599, 1993.

50 R.S. Cármenes, J.P. Freije, M.M. Molina, J.M. Martín, *PREDICT7, a program for protein structure prediction*, **Biochem. Biophys. Res. Comm.**, vol 159, pp. 687-693, 1989.

51 J. Garnier, D. J. Osguthorpe, B. Robson, **Journal of Molecular Biology**, vol 120, pp. 97-120, 1978.

52 This is only available for test runs with known protein conformations.

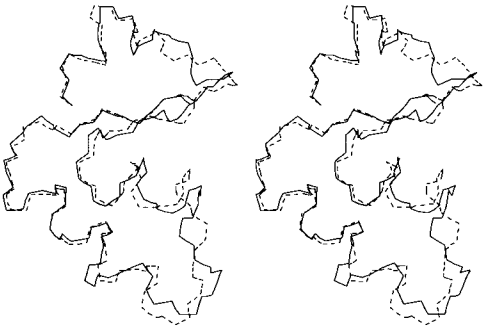


Figure 19: Crambin Predicted by R.M.S. Fitness Function.
This conformation (solid line) with a r.m.s.-deviation of 1.08 Å to native Crambin (dashed line) was obtained after 10,000 generations using the LOCAL TWIST, MUTATE, VARIATE and CROSSOVER operators and r.m.s.-deviation as the fitness function.

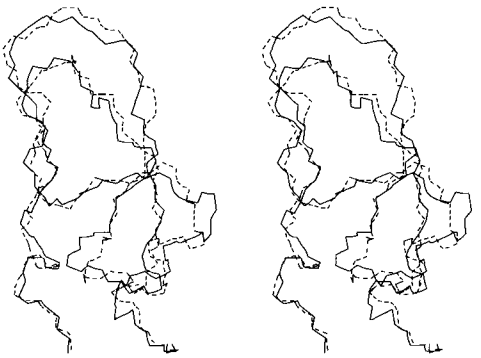


Figure 20: Trypsin Inhibitor Predicted by R.M.S. Fitness Function.
Stereoscopic superposition of the native conformation (dashed line) and one individual of the final generation (solid line). The r.m.s.-deviation is 1.48 Å.

dividual with a r.m.s.-deviation of 0.89 Å. This demonstrates the suitability of the genetic algorithm approach to protein folding. Given a reliable fitness function the genetic algorithm is able to successfully traverse the torsion angle search space.

Other proteins that were used for test purposes of the genetic algorithm with an r.m.s.-fitness function are the trypsin inhibitor protein (Brookhaven database code 5PTI; final r.m.s.-deviation 1.48 Å; Figure 20) and RNAse T1 (Brookhaven database code 2RNT, final r.m.s.-deviation 2.32 Å; Figure 21).

The fact that none of the structures produced in the runs with a r.m.s.-fitness function were completely identical to the native conformations is explained by the following three observations:

1. The use of standard binding geometries for reconstructing 3D coordinates from a set of torsion angles could cause structural alterations where the native conformation does not closely adhere to the theoretically derived ideal bond lengths

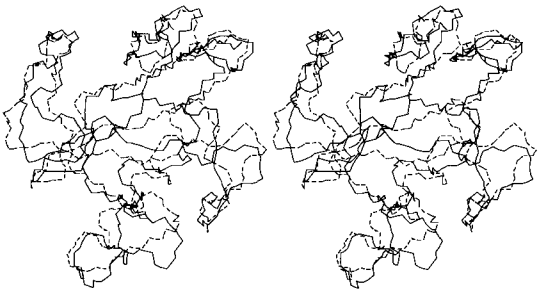


Figure 21: RNAse T1 Predicted by R.M.S. Fitness Function.
Stereoscopic superposition of the native conformation (dashed line) and one individual of the final generation (solid line). The r.m.s.-deviation is 2.32 Å.

and bond angles. In this case the best match will always have a r.m.s.-deviation of greater than zero.

2. The operators MUTATE, VARIATE and CROSSOVER in theory cannot produce an exact match even if the target structure is known in detail. This is a result of the representation formalism that these operators work on. If the current individual is already structurally similar to the desired protein then a single application of MUTATE or VARIATE is most likely to introduce mismatches of previously well fitting fragments and thus deteriorates the conformation. This happens because even if one bond becomes better aligned the rest of the protein towards the C-terminal swings away and increases the r.m.s.-deviation. CROSSOVER is not able to improve this situation for the same reason.
3. Only the LOCAL TWIST operator can improve a fit locally without disturbing well fitting fragments that surround the mutation site. However, the applicability of LOCAL TWIST is mathematically constrained: when starting from a less fitting conformation the optimal local improvement is not always found in one pass. Sometimes it is even impossible to improve a local conformation at all.

Hence, with an increasing number of generations it becomes more and more difficult to achieve any further improvement in the r.m.s.-fitness and the search stagnates at r.m.s.-deviation values between 0 – 2 Å (Figure 22).

Another conclusion to draw from the above experiments with the r.m.s.-fitness function is that the fitness function is the crucial topic. This is clearly an unresolved issue and the subject of ongoing research in protein engineering. Some aspects of the computational complexity have already been explained above. This situation led to the following experiments with the genetic algorithm and a multi-value vector fitness function.

Figure 23 shows the results of a run with the fitness components *polar*, E_{pe}, E_{tor}, E_{el}, *hydro*, *Crippen* and *solvent*. This individual had an r.m.s.-deviation of 6.27 Å from the native conformation of Crambin. The genetic algorithm did not use the r.m.s.-deviation as part of the fitness function. Only the fitness components listed above were used to guide the genetic algorithm. Over the whole run some of the fitness components decreased along with r.m.s.-deviation (E_{pe}, *hydro*, *Crippen*, *solvent*), as was expected. However, the other fitness components (*polar*, E_{tor}, E_{el}) ac-

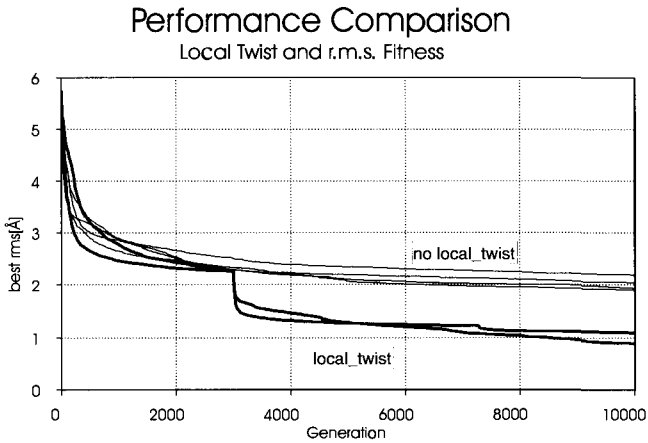


Figure 22: Performance Comparison for the LOCAL TWIST Operator.
This graph shows the course of six single experiments with the r.m.s-deviation as the fitness function. The individual with the best r.m.s.-deviation is plotted for each generation. The two thicker lines at the bottom have the LOCAL TWIST operator switched on after 3000 generations. Reproduction was done by the roulette wheel algorithm. The four runs without LOCAL TWIST had a population size of 54 individuals whereas the two runs with LOCAL TWIST had only 30.

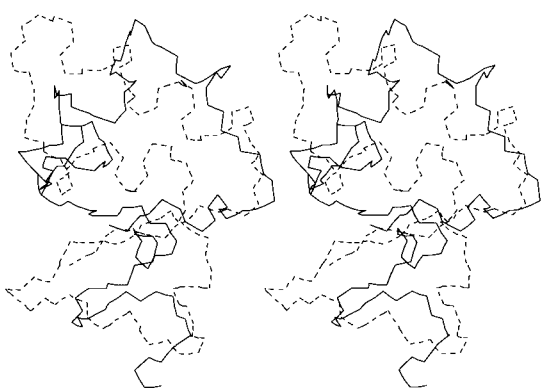


Figure 23: Individual of the Final Generation of a Multi-Value Fitness Run.
Only the fitness components *polar*, E_{pe}, E_{tor}, E_{el}, *hydro*, *Crippen* and *solvent* were used to guide the genetic algorithm in this run. There is a vague similarity (r.m.s. 6.27 Å) in the overall backbone fold of the generated individual (solid line) to native Crambin (dashed line).

tually drove the genetic algorithm to conformations with less similarity to the native Crambin indicating that these propensities were no good indicators for the "nativeness" of Crambin. In general, no r.m.s.-values better than around 6 Å were detected in similar runs.

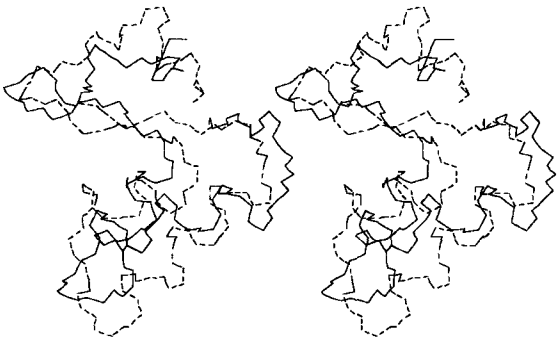


Figure 24: Folding Crambin with Secondary Structure Constraints.
The backbone of the predicted conformation (solid line) and Crambin (dashed line) have only a r.m.s.-deviation of 4.36 Å. For this run only the fitness components *Crippen*, *clash*, *hydro* and *scatter* were used in the multi-value vector fitness function.

The following conformations were generated with the fitness components *Crippen*, *clash*, *hydro* and *scatter*. In addition, constraints on the secondary structures of Crambin were imposed by limiting the backbone angles to intervals between the upper and lower values of Table 7. Torsion angle ω was constrained to 180°. For a general application the use of secondary structure constraints requires a highly accurate and reliable secondary structure prediction algorithm which unfortunately does not (yet) exist. Figure 24 shows the backbone of an individual generated by the genetic algorithm with the above mentioned fitness components and that has a r.m.s.-deviation from native Crambin of 4.36 Å.

Another run with the same fitness components was performed for trypsin inhibitor (Figure 25). The r.m.s.-deviation from native trypsin inhibitor is 6.65 Å. This is worse than the result for Crambin in Figure 24 because the lower content of secondary structure in trypsin inhibitor implies less rigid constraints on the conformation. This means there are more degrees of freedom and therefore a larger search space to traverse.

Summarising these findings and those of the previous subsections we are led to the following conclusions.

1. Genetic algorithms proved to be an efficient search tool for both 2-D and 3-D representations of proteins. In a 2-D protein model the genetic algorithm outperformed Monte Carlo search in both the quality of the results and required computation time. For a 3-D protein model with a simple, additive force field as fitness function and using a rather small population the genetic algorithm produced *several* individuals (i.e. protein conformations) of dissimilar topology but each with highly optimised fitness values.
2. Given an appropriate fitness function (for test purposes the r.m.s-deviation from the *a priori* known conformation can be used) the genetic algorithm application described in this section finds the desired solution within only small deviations.

Figure 25: Backbone Folding of Trypsin Inhibitor.
The backbone of the predicted conformation (solid line) and trypsin inhibitor (dashed line) have a r.m.s.-deviation of 6.65 Å. For this run only the fitness components *Crippen*, *clash*, *hydro* and *scatter* were used. The comparatively bad performance of the genetic algorithm in comparison to the run on Crambin (Figure 24) is a result of the low content of secondary structure in trypsin inhibitor which increases the number of rotational degrees of freedom.

3. The major problem lies in the fitness function. If there were one or a set of indicators that return "1" for "*the object is a native protein conformation*" and "0" for "*the object is not a native protein conformation*" one could expect the genetic algorithm approach to deliver reasonably accurate *ab initio* predictions. However, neither mathematical models, empirical, semi-empirical or statistical force fields are yet accurate enough to reliably discriminate native from non-native conformations without additional constraints. Thus, the genetic algorithm produces (sub-)optimal conformations in a different sense than that of "nativeness".
4. Because secondary structure in nature and J. H. Holland's building blocks in the genetic algorithm are analogous fundamental components for the construction of the individual, it was hoped that secondary structures would emerge as the building blocks in a subset of the population. This has not yet happened. One possible explanation is that the fitness functions used are not sensitive enough to detect and account for the structural benefits in secondary structures.

Let us now turn to a variant of the protein folding problem, the question of how to dock a substrate to a protein and how simulated annealing can be used to address this problem.

5.2.4. Protein – Substrate Docking

Docking is the process by which a substrate attaches to a protein. The interaction between both may be temporarily as in the case for an enzyme and its substrate or

a receptor and its ligand, or it may be permanent as for prosthetic groups or co-factors. A substrate can also be another protein. In all cases interactions of surface atoms and / or protruding groups mediate and constrain the docking process. Only a substrate that fits the steric, hydrophobic and polar requirements of the receptor can establish a stable binding pattern.

Various docking methods have been tested. Manual docking with the aid of computer graphics is probably still the first method of choice because in the hands of an expert it quickly yields practical answers without many prerequisites. Automatic methods to simulate docking can be divided into two broad categories: *energy driven docking* and *distance constraint docking*. The former uses a force field to determine favourable contacts between the two molecules, possibly reducing the search space by constraining the two molecules to a grid. The latter tries to geometrically maximise the contact between two molecular shapes based on the optimal geometric complementarity of molecular surfaces.

5.2.4.1. Distance Constraint Docking

As an example of distance constraint docking by simulated annealing we will have a look at the work of Shi-Yi Yue[53]. The simulated annealing procedure used is the standard Metropolis algorithm as explained in section 5.1.4. A new configuration of protein and ligand is accepted if its energy E_{new} is less than the energy of the previous configuration E_{prev}. If not, the new configuration is accepted only with the probability:

$$p = e^{\frac{-(E_{new}-E_{prev})}{kT}},$$

where k is the Boltzmann constant and T the temperature.

A new conformation is generated by a random change in one of the six degrees of freedom of the ligand group (three rotational, three translational degrees). A first random number selects one of the six degrees to be modified and a second random number specifies the step size of the increment (in the range of -45° to 45° for rotation and -2.0 Å to 2.0 Å for translation). The three rotation matrices about the x, y and z axes are:

$$R_x = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{vmatrix}, \quad R_y = \begin{vmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{vmatrix},$$

$$R_z = \begin{vmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{vmatrix}.$$

The configuration of a complex is defined by the coordinates of an enzyme and its ligand. The constrained distances r_{ref} are numbered $i = 1, \ldots, N$ with bounds on specific pairs of atoms. These atoms define the binding site. In practice, the selection criterion for a constrained pair of atoms is that one atom must reside in the

53 Shi-Yi Yue, *Distance-constrained molecular docking by simulated annealing*, **Protein Engineering**, vol 4, no 2, pp. 177-184, 1990.

enzyme, the other in the ligand and that their distance has to be less than 3.3 Å. The information for r_{ref} can be gained from experimental data or through general considerations, e.g. about the optimal distance for a hydrogen bond if one knows that there is one in the binding site.

The cost function in this application, i.e. the energy function in the original Metropolis algorithm, is determined by the following equation:

$$E = \sum_{i}^{N} K(r_i)(r_i - r_i^{ref})^2, \quad \text{with } K(r_i) = \begin{cases} 500 & \text{if } r_i \leq 1.5\text{Å} \\ 20 & \text{if } r_i > 1.5\text{Å} \end{cases} .$$

The sum extends over all specified, constrained pairs of atoms. Very small distances (< 1.5 Å) represent excessive atomic overlap and a large repulsive force constant is returned for such an interaction. Larger distances result in a small effective energy constant. The exact values are not important but it is necessary to maintain a reasonable ratio of repulsive force to effective energy. The above formula identifies favourable configurations (minimum $E = 0$ for exact match) and eliminates bad van der Waals contacts.

The cooling schedule for these experiments is as follows. The temperature factor kT is set to 8.0 and decreases by 10% at each temperature step. Fifty temperature steps are taken per run and at each temperature step 100 configurations are generated. At the end of each step the acceptance rates for rotation and translation are examined. If any one of these is below 0.1 the corresponding increment size is reduced by 20% for the next temperature step. In addition to the current and previous configuration the program also saves the coordinates of the best configuration generated since the beginning of the run.

In test runs on eight serine protease complexes all available distance constraints were used. This was done to check whether the simulated annealing procedure can reproduce a known fit. All runs converged to a minimum of the average distance violation:

$$D_{error} = \frac{1}{N} \sum_{i}^{N} |r_i^{best} - r_i^{ref}| ,$$

in this case of less than 0.015 Å after 50 temperature steps. The average deviation *rmsd* of the final best configuration after 100 runs was less than 0.1 Å. The *rmsd* value is calculated as:

$$rmsd = \sqrt{\frac{1}{N_{ligand}} \sum_{i}^{N_{ligand}} \sum_{j}^{3} \left(C_{ij}^{a} - C_{ij}^{b}\right)^2} ,$$

where N_{ligand} is the number of atoms in the ligand and C^a and C^b are sets of three-dimensional coordinates for two different configurations of the ligand.

The next experiment was done to study the influence of the number of constraints on the convergence of the simulated annealing procedure. Only ligand atoms that participate in several distance constraints (distance < 3.3 Å) were considered. The average distance violation was below 0.03 Å and the average deviation *rmsd* also increased only slightly to values less than 0.3 Å.

In the last experiment, a set of hydrogen bonds and short contact distances was obtained from the X-ray structure of chymotrypsin (Brookhaven database code 1CHO) and the backbone of residues 14 – 20 of the third domain of ovomucoid inhibitor of Turkey (P5-P'2 sites; OMTKY3). The distance constraints derived from fitting OMTKY3 to 1CHO were used to dock three other, structurally similar ligands to 1CHO. Such computations would be performed if one has a set of intermolecular distance constraints, e.g. from NMR experiments, but lacks an experimental structure for the ligand. A model of the ligand structure would then be built based on a known inhibitor of a homologous enzyme. In these experiments the maximum average distance violations were 0.096 Å, 0.094 Å and 0.61 Å and the *rmsd* values are 0.93 Å, 0.91 Å and 4.76 Å. For the *rmsd* values reference values were computed by fitting the homologous ligands to OMTKY3 and then measuring the *rmsd* to 1CHO. This gave the values 0.71 Å, 1.01 Å and 3.34 Å. The simulated annealing procedure cannot be expected to surpass the reference *rmsd* values except by fortuitous coincidence. Simulated annealing came in these cases as close as 31%, 10% and 43% to the expected values. It is clear that this docking method only provides an approximate orientation and position of the ligand. Energy minimisation and / or molecular dynamics should be applied to further refine the relative configuration of enzyme and ligand.

5.2.4.2. Energy driven Docking

D. S. Goodsell and A. J. Olson devised another application of the simulated annealing procedure to the docking problem[54]. In addition to the work presented in the previous section, not only rotational and translational degrees of freedom are included but also rotation around individual torsion angles of the substrate. The energy function is a molecular affinity potential[55]. The protein is embedded in a three-dimensional grid and a probe atom is sequentially placed at each grid point. The energy of interaction of this single atom with the protein is assigned to that grid point. An affinity grid is calculated for each type of atom in a typical substrate (C, O, N, H) as well as a grid of electrostatic potential using a point charge of +1 as the probe. During simulation, these grids are used as look-up tables to evaluate the energy of interaction rapidly. The order of complexity of this implementation is proportional only to the number of atoms in the substrate, not to any function of the number of atoms in the protein. As the grids are calculated only once for a particular protein even computationally expensive force fields can be used without slowing down the simulated annealing process.

Each simulation was composed of 50 cycles containing a maximum of 30000 accepted or 30000 rejected steps, whichever limit was reached first. Simulations were started at a very high temperature ($kT = 100$ kcal/mol). The temperature was lowered by multiplication by 0.9 at each step, which yielded a final temperature of $kT = 0.5$ kcal/mol. Displacements at each step were a random fraction of 0.2 Å for translation and of 5° for rigid body and torsional rotations.

54 D. S. Goodsell, A. J. Olson, *Automated Docking of Substrates to Proteins by Simulated Annealing*, **Proteins**, vol 8, pp. 195-202, 1990.

55 P. J. Goodford, *A computational procedure for determining energetically favorable binding sites on biologically important molecules*, **Journal of Medical Chemistry**, vol 28, pp. 849-857, 1985.

Five systems were analysed. Immunoglobulin McPC 603 with phosphocholine (0.97 Å), chymotrypsin with N-formyltryptophan (1.32 Å), lysozyme with α-N-acetyglucosamine (3.96 Å), lysozyme with β-N-acetyglucosamine (0.97 Å), and aconitase with sulfate (0.95 Å). The r.m.s.-deviations given are the best values found for each complex. However, it was not always the final configuration of a run but sometimes the one ranked second, third or fourth by the energy function that had the best r.m.s.-deviation.

This concludes the current list of applications from evolutionary computation in molecular biology. As many biologists are becoming familiar with the algorithms presented here many more applications are expected to appear in core journals of molecular biology and relevant international conferences[56] over the next years.

56 See the **Journal of Molecular Biology** and **Protein Engineering**. Other relevant journals are **Proteins, Journal of Computational Biology**, and the **Journal of Computational Chemistry**. From the computational side watch for the announcements and proceedings of the **International Conference on Genetic Algorithms**, the **Parallel Problem Solving from Nature** Conference, and the **International Conference on Intelligent Systems in Molecular Biology**.

6. Artificial Neural Networks

Artificial neural networks are a class of computational algorithms that are based on the model of biological neural networks. In biological networks neurons collect and process excitatory and inhibitory stimuli and emit electro-chemical signals in response. If the incoming stimuli of a neuron reach a certain threshold the synapse of the neuron will fire. Neurotransmitters control the effective strength of a connection between neurons. Nodes and connective arcs in an artificial neural network are analogous to the neurons and synapses in a biological neural network. The simplest artificial neural network topology is an acyclic directed graph. This type of artificial neural network is referred to as *feed-forward network*. Other network topologies may have different and more complex configurations.

6.1. Methodology

Most neural network models that have been applied to problems in molecular biology use a supervised learning algorithm. These were first developed by F. Rosenblatt[1]. More recently, other algorithms for multi-layer networks have been developed[2]. In the *supervised learning mode* the strength or weight of an arc between nodes is initialised with a random value. Training examples are presented to the input and output nodes. The algorithm "learns" from the answer by modifying the weights inside the net according to one of several learning models. The objective is to minimise the difference between the desired and actual output of the network. Having completed the learning phase the weights of the artificial neural network will have converged in most cases and the net can be applied in a predictive mode to unknown examples.

In the *unsupervised learning mode*, the artificial neural network is presented with a set of unclassified examples. It then tries to learn a proper classification that reflects the similarities and differences among the examples. The trained network with the resulting classification can be used to identify the class of a new example based only on a subset of its features. The class assignment may then be taken to propose reasonable values for unknown or missing attributes of the new example. Two types of artificial neural networks will be presented in this section in more detail, one for supervised learning and another for unsupervised learning.

---

1 F. Rosenblatt, *The Perceptron: A probabilistic model for information storage and organization in the brain*, **Psychological Review**, vol 65, pp. 386-408, 1958.

2 D. E. Rumelhart, J. L. McClelland (eds.), **Parallel Distributed Processing**, Cambridge MA, MIT Press, 1986.

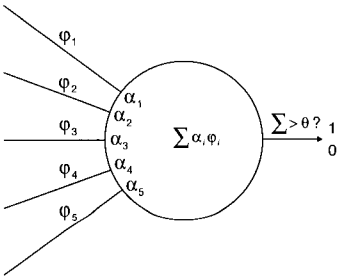


Figure 1: Perceptron.
The Perceptron gets real valued input on a number of input channels φ_i. The input values are weighted (weights α_i) and added. If the sum is greater than a threshold condition θ the Perceptron returns the value "1", or "0" otherwise. For each particular application suitable weights have to be learned in the training phase from a set of correlated input / output pairs.

6.1.1. Perceptron and Backpropagation Network

One of most widely used network architectures for supervised learning is the so-called backpropagation network[2]. Backpropagation networks are an extension of the first simple network model, the Perceptron[3]. Figure 1 shows the basic architecture of the Perceptron.

Let $\Theta = \{\varphi_1, \varphi_2, \ldots, \varphi_N\}$ be a set of predicates. In the predicates' simplest form, $\varphi_1 = 1$ denotes that this predicate is applicable for the current example whereas $\varphi_1 = 0$ says it is not. Each of the input predicates is weighted by a number α_i. The output of the Perceptron is 1 if the sum of the weighted inputs exceeds a threshold value θ. Otherwise the Perceptron returns 0. To train a Perceptron, a set of examples where the desired output (0 or 1) is known in advance is presented successively to the neuron and the weights are iteratively adjusted to match the desired output. The Perceptron output function is:

$$f_{perceptron} = \begin{cases} 1 & \text{if } \sum \geq \theta \\ 0 & \text{if } \sum < \theta \end{cases} \quad \text{or, for a neuron with 2 inputs:}$$

$$f_{perceptron} = \begin{cases} 1 & \text{if } \alpha_1\varphi_1 + \alpha_2\varphi_2 \geq \theta \\ 0 & \text{if } \alpha_1\varphi_1 + \alpha_2\varphi_2 < \theta \end{cases} .$$

We now consider the task of a Perceptron learning the XOR problem. The variable bindings and the desired output for the XOR problem are shown in Table 1.

Given the values for φ_1 and φ_2 the Perceptron should return the desired output value. After experimenting with a few combinations of α_1, α_2 and θ in the Perceptron output function above the reader may grow suspicious whether there is any solution at all. – In fact, there is none for this task. Geometrically speaking, to solve the XOR problem with the basic Perceptron a straight line would be needed to separate the points (0, 0) and (1, 1) from the points (0, 1) and (1, 0) in a two-

3 M. Minsky, S. Papert, **Perceptrons**, MIT Press, Cambridge MA, 1969, 1988 (expanded edition).

φ_1	φ_2	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: XOR Problem.
The input values of φ_1 and φ_2 to the Perceptron should produce the desired output value. This truth table defines the logical "exclusive-or" operator.

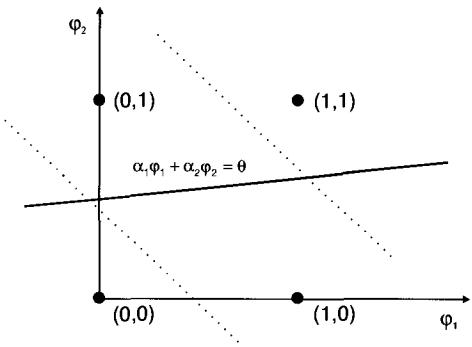


Figure 2: Geometric Representation of the XOR Problem.
At least two straight lines (dotted) are needed to separate the points (0, 0) and (1, 1) from the points (0, 1) and (1, 0).

dimensional Cartesian coordinate system. The equation $\alpha_1\varphi_1 + \alpha_2\varphi_2 = \theta$ divides the (φ_1, φ_2)-plane into two regions but cannot separate both points (0, 0) and (1, 1) from the points (0, 1) and (1, 0) (Figure 2).

To overcome this inherent limitation of the Perceptron one can introduce so-called *hidden layers*. These are additional artificial neurons connected between the input and output layer. The topology of a general three-layer feed-forward network is shown in Figure 3.

The backpropagation algorithm works as follows:

1. Pass the input vector $\vec{x}_p = (x_{p1}, x_{p2}, \ldots, x_{pN})^t$ to the input units. p is the index of the current example.
2. Calculate the node values of the hidden units $net^h_{pj} = \sum_{i=1}^{N} w^h_{ji} x_{pi} + \theta^h_j$. i is the index to the dimension of the input vector and input node, j is the index of the hidden units. The term θ^h_j reflects optional bias units.
3. Calculate the output values for the hidden layer $r_{pj} = f^h_j(net^h_{pj})$. Choices for the node function $f^{h,o}_q$ are $f^{h,o}_q(net^{h,o}_q) = net^{h,o}_q$ or $f^{h,o}_q(net^{h,o}_q) = \frac{1}{1+e^{-net^{h,o}_q}}$. The superscripts "h" and "o" denote hidden unit and output unit, respectively. The index q can be used for hidden and output units.

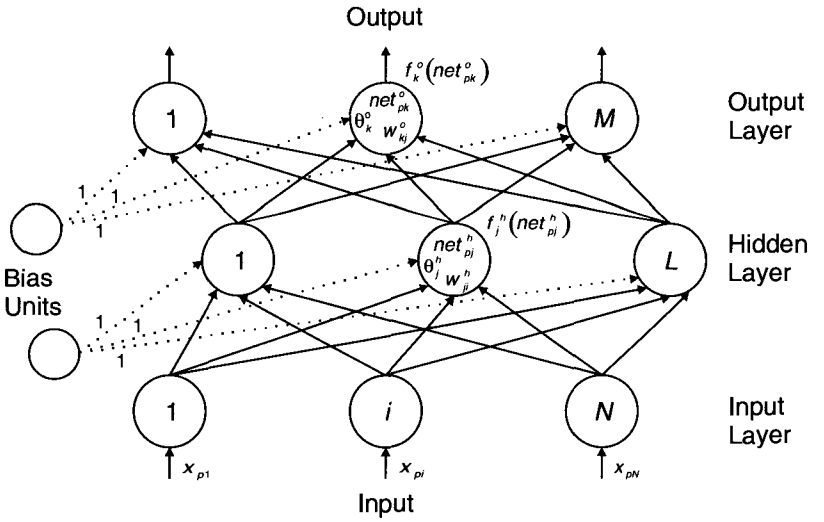


Figure 3: Three-Layer Feed-Forward Backpropagation Network.
Circles symbolise artificial neurons. There are N input units (index i for nodes, index p for examples) with input values x_{pi}, one hidden layer with L hidden units (index j) and M output units (index k). The superscripts "h" and "o" refer to hidden and output unit, respectively. Bias units are optional and can be used to adjust the learning process.

4. Calculate the node values of the output units $net_{pk}^o = \sum_{j=1}^{L} w_{kj}^o r_{pj} + \theta_k^o$. As before, the term θ_k^o reflects optional bias units.
5. Calculate the output values for the output layer $s_{pk} = f_k^o(net_{pk}^o)$.
6. Calculate the error terms δ_{pk}^o for the output units. k is the index for the output units, y_{pk} are the known, expected output values and s_{pk} are the computed values from step 5. The error δ_{pk}^o is then calculated by $\delta_{pk}^o = (y_{pk} - s_{pk}) f_k^{o'}(net_{pk}^o)$ with $f_k^{o'} = 1$ *or* $f_k^{o'} = f_k^o \cdot (1 - f_k^o) = s_{pk} \cdot (1 - s_{pk})$.
7. Calculate the error terms δ_{pj}^h for the hidden units (before updating the weights connecting hidden and output units) $\delta_{pj}^h = f_j^{h'}(net_{pj}^h) \sum_k^M \delta_{pk}^o w_{kj}^o$.
8. Update the weights w_{kj}^o that lead from the hidden layer to the output layer in cycle t as follows: $w_{kj}^o(t+1) = w_{kj}^o(t) + \eta \delta_{pk}^o r_{pj}$, with a learning rate $0 \leq \eta \leq 1$. The learning rate η can be adjusted during run time.
9. Update the weights w_{ji}^h of cycle t leading from the input layer to the hidden layer by $w_{ji}^h(t+1) = w_{ji}^h(t) + \eta \delta_{pj}^h x_{pi}$.
10. Calculate the overall error term $\Delta_p = \frac{1}{2} \sum_{k=1}^{M} \left(\delta_{pk}^o\right)^2$. This quantity is the measure of the learning performance of the network. When the overall error is acceptably small for all training examples, training can be terminated and the net can be applied to new examples in the prediction mode.

There are a number of commercial software packages that allow the user to adapt the backpropagation algorithm to his / her specific purposes. There is also some useful public domain software available in this area[4].

6.1.2. Kohonen Network

T. Kohonen's model[5] of unsupervised learning can be used for feature reduction or clustering and belongs to a more general set of algorithms known as *competitive learning*. The basic principle of competitive learning is the use of a contest between individual nodes (or processors) to best approximate the current example. The winner node and its nearest neighbours are modified to better fit the current example. There are two phases in using a Kohonen network. In the training phase, examples are repeatedly presented to the network in random order. For each input one node wins the competition of having a state vector most similar to the current example. The net will reorganise itself to assimilate the winner neuron to even better match the incoming example. After this has been repeated many times similar examples will reinforce the same node. In the classification phase, examples are put through the network and their winner nodes are recorded but the state vectors remain unchanged.

The basic algorithm of a Kohonen network is hence as follows:

1. A Kohonen network consists of an interconnected layer of nodes. The connectivity between nodes can be in one, two, three or more dimensions. A neighbouring scheme defines which nodes are considered direct neighbours. In two dimensions neighbours of a node could be those nodes within a Hamming distance of 1.
2. Each node has a state vector $node_i$ with d vector components. d is called the dimension of the application and is invariable during the run.

$$node_i = \begin{pmatrix} v_1^i \\ \cdots \\ v_d^i \end{pmatrix}$$

3. Each component v_k of a state vector corresponds to a feature (or attribute) of an example. The values of the attributes should be normalised.
4. Initially, the values of the state vectors are randomised numbers between 0 and 1.

4 Try, in this order: 1) Internet archie / gopher (Veronica) search for the keywords "backprop", "neural", "net", etc. 2) Search WWW catalogues for the same keywords. 3) Scan Usenet Netnews groups for neural computing and ask for references on public domain software. – Two public domain packages that I found very useful are *Fast Backpropagation* by D. R. Tveter (email drt@chinet.chi.il.us) and the *Stuttgarter Neuronaler Netz Simulator (SNNS)*, University of Stuttgart, Department of Computer Science, Germany (email mamier@informatik.uni-stuttgart.de; anonymous ftp server ftp.informatik.uni-stuttgart.de or 129.69.211.2, directory /pub/SNNS), both available in C source code for a number of platforms, including DOS and Unix.

5 T. Kohonen, *Self-Organization and Associative Memory*, **Springer Series in Information Science**, vol 8, Springer, New York, 1984.

5. There is a winner function which is the minimum function. The node with the state vector that is most similar (e.g. in terms of an Euclidean distance measure) to the input vector of the current example is declared winner. In ambiguous cases, the node with the lower index is chosen.

$$node_i \text{ is winner with min } = \left(\left| \begin{pmatrix} input_1 \\ \cdots \\ input_d \end{pmatrix} - \begin{pmatrix} v_1^i \\ \cdots \\ v_d^i \end{pmatrix} \right| \right)$$

6. There is an update function which transforms the state vector of a winner node and its nearest neighbours (as defined in step 1) such that it becomes more similar to the input vector of the current example. Usually, the update function increments the state vector by the product of a learning rate μ (e.g. 0.01 or 0.5) and the difference of input vector and the (old) state vector.

$$node_i^{t+1} = node_i^t + \mu \cdot \left(\begin{pmatrix} input_1 \\ \cdots \\ input_d \end{pmatrix} - \begin{pmatrix} v_1^i \\ \cdots \\ v_d^i \end{pmatrix} \right)$$

7. All examples are randomly presented to the net several times (e.g. between 50 to 1000 times) before a net is considered trained. It can then be used to classify the training set or previously unseen examples. For classification, the examples are presented to the net and their winner nodes are recorded. In contrast to the learning mode state vectors are not updated.
8. Each node can be interpreted to represent one class. Its state vector describes the features of that class.

In the following we will examine a simple geometric application of a Kohonen network. The net has 8×8 nodes, each with a two-dimensional state vector and a planar topology in two dimensions. The neighbour-function returns all orthogonally and diagonally adjacent nodes; i.e. nodes in the middle of the net have nine neighbours and nodes in a corner only three. The state vectors of all nodes are initialised with the same pair of values (0, 0). The training set is a sequence of pairs of uniformly distributed random numbers between -1 and 1. This setting implies that the net should learn the concept of a two-dimensional rectangle. Figure 4 shows six snapshots of the net during a run.

At the beginning, all nodes have similar state vectors. Incoming training examples that lie close to the edges of the rectangle [(-1, -1), (-1, 1), (1, 1), (1, -1)] drag individual nodes away from the centre (0, 0). The reinforcement of neighbouring nodes causes the net to spread out and eventually to evenly cover the (x, y)-plane. This result can be interpreted in two ways.

1. The state vectors of each node can be taken as the centre of a *cluster* of a subset of (x, y)-training pairs. As the training examples were uniformly distributed the rectangular arrangement of nodes represents the arrangement with optimal resolution. This interpretation of the result is analogous to statistical cluster analysis.
2. The net adopts a *structure* that is *topologically* equivalent to the distribution of the training examples. Figure 5 shows a comparison of one net learning a uniform distribution and another that is given a biased distribution. The biased

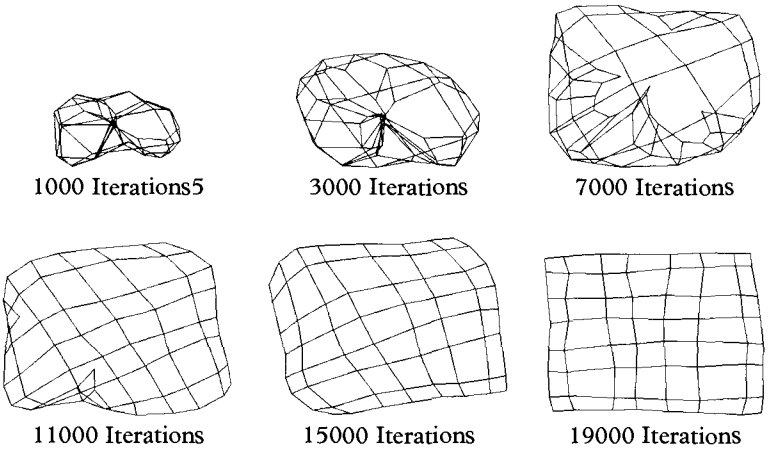


Figure 4: Evolving Kohonen Net.
Six snapshots of a Kohonen net learning the concept of a 2-dimensional rectangle.

distribution was computed identically to the uniform distribution except that in 80% of the examples the interval was limited to the smaller, rectangular area [(-0.3, -0.3), (-0.3, 0.3), (0.3, 0.3), (0.3, -0.3)], instead of [(-1, -1), (-1, 1), (1, 1), (1, -1)]. The net responded with an arrangement that spends more nodes in the centre of the rectangle to discriminate the majority of examples in this area more accurately.

For each application, the following preparations must be taken.

1. Decide how to encode the training examples. The representation involves a decision on whether it is more desirable to discover clusters or a topologically equivalent net configuration. This also determines the dimension of state vectors.
2. What is a reasonable upper limit of the number of nodes required? Less nodes are processed faster and are less likely to tangle up the net whereas more nodes give a better resolution for classification.
3. What type of connectivity should be used? The topology can be in one, two, three or more dimensions.
4. Which nodes are considered neighbours? The neighbour-function can change during run time and towards the end of the run the neighbourhood may become smaller for fine tuning.
5. Which value should be used for the learning parameter? One way could be to automatically adjust the learning parameter to decrease with the number of training cycles.
6. When should one stop training the net? Optimally, there would be a criterion that can be used to measure the performance of the net (e.g. training examples with known classification). If the net reaches a certain performance level train-

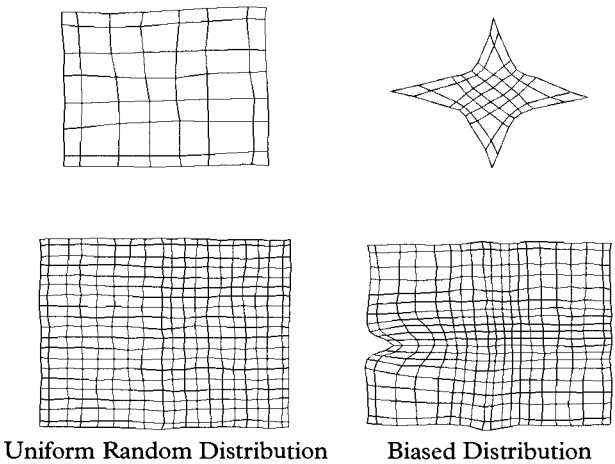


Figure 5: Topological Equivalency.
The final arrangement of nodes corresponds to the distribution of training examples. The upper two nets are made of 8 x 8 nodes, the lower ones of 20 x 20 nodes. For a biased distribution, more nodes are spent on areas with higher density of training examples to enhance overall resolution. The "dent" in the lower right net probably results from imperfections of the random number generator.

ing can be stopped. Otherwise, one could stop training after a predetermined number of training cycles (number of times the complete set of training examples are presented to the net) or if the changes of the net arrangement become smaller than a predefined limit.

It should be noted that the result of a run with a Kohonen net highly depends on all the parameters listed above. Especially, if one wants to use a Kohonen net for clustering purposes it is recommended to try several combinations with different parameters for net size, net topology, learning parameter and neighbour-function. Otherwise, there is a danger of arriving at misleading results.

6.2. Applications

There have been manifold applications of artificial neural networks in biosciences. Most of them can be divided into two groups:

- *pattern recognition*, to correlate biological features with biochemical or physical processes and
- *prediction*, to apply rules and principles which are yet hidden in sets of data to new, previously unknown cases.

A few examples for each category are listed in Table 2.

Pattern Recognition	Prediction
Design of Media for Bacterium Growth	Biotransformation
DNA Fingerprinting	Fermentation Variables
DNA Sequences	Harvest Times
Drug Screening	Human Responses
Experimental Design	Operational Scheduling
Fault Detection	PID Loop Tuning
Interpretation of Flow Cytometry	Predicting Project Time
Microbial Identification	Protein Structure
Protein Structures	Sensor Interpretation
Spectral Data Matching	Unmeasurable Quantities
Structure-Activity Relationships	
Taxonomy	

Table 2: Applications of Neural Nets in Biosciences.
Pattern recognition tasks are mostly done by supervised learning artificial neural networks. Prediction tasks require a network that can be trained in supervised or unsupervised mode.

In this chapter, we will examine three supervised artificial neural network learning tasks: exon-intron boundary recognition, secondary structure prediction, main chain torsion angle prediction and one unsupervised learning task: the detection of super-secondary structures formed by long-range interactions.

6.2.1. Exon-Intron Boundary Recognition

In eucaryotic organisms proteins are usually not encoded by a single, contiguous DNA strand but are translated from a concatenation of a small number of separate DNA fragments. Those fragments are called *exons* and the non-coding intervening sequences between exons are termed *introns*. Information on intron sequences is not used for assembling proteins but it may have other, less obvious regulatory functions. Eucaryotic cells contain enzymes that catalyse the so-called *splicing* process in which exons are excised from a transcript of genomic DNA, the so-called messenger RNA. Splicing is directed by DNA patterns at exon-intron boundaries. This section explains how a backpropagation network can be used to detect these exon-intron boundaries.

First, let us examine the choices for representing DNA sequences in a backpropagation network. Ideally, a DNA sequence would be encoded as a string composed of letters from the alphabet {A, G, T, C}, representing the four nucleotides adenine, guanine, thymine, and cytosine. However, a backpropagation network accepts only numbers as input to its nodes. There are a number of ways to transform a DNA string into a series of numbers which are acceptable as input for a backpropagation network.

1. If we decide to map one nucleotide on one input node we could define the number 1 to stand for adenine, 2 for guanine and so on. This *nominal* representation has the disadvantage that the different amounts in each of the four codes do not reflect a particular property of the respective nucleotide. The backpropagation network, however, might process the information in a sense of "cytosine is four times weaker than adenine".
2. Four objects require two bits to label them uniquely. Thus, one could use two input nodes per one nucleotide and assign the values (0, 0) for adenine, (0, 1) for guanine, (1, 0) for thymine and (1, 1) for cytosine. This leads to the same problem as in version 1 above. Cytosine excites the backpropagation network twice as much as adenine.
3. So, why not use four bits to represent one nucleotide? Adenine would get (1, 0, 0, 0), guanine (0, 1, 0, 0), thymine (0, 0, 1, 0) and cytosine (0, 0, 0, 1). This way the main difference between nucleotides is not an amount of excitation but a difference in input excitation patterns.
4. There is another means to process raw DNA sequences into an acceptable input format for a backpropagation network. If we use fragments of standardised length one can count the percentage of each of the four nucleotides and give these four numbers to the net together with the information whether this fragment contains a splice site and if so of which type. Or, in a more refined version, the relative occurrence of each of the 16 nucleotide pairs could be determined. Then, 16 numbers corresponding to 16 real-valued input nodes would be required to specify one DNA fragment. Similarly, the relative occurrence of all 64 nucleotide triplets can be used.
5. Finally, one can use a combination of the previous representation models for all six reading frames[6] of a DNA fragment.

The following experiments were done with a standard implementation of the backpropagation algorithm as explained in section 6.1.1, except the additional use of a so-called *momentum* parameter α. When updating a weight, the fraction α of the previous weight change is added to the last weight together with the product of learning rate, error and input values:

$$w_{ji}^{h,o}(t+1) = w_{ji}^{h,o}(t) + \eta\delta_{pj}^{h,o}x_{pi} + \alpha\Delta w_{ji}^{h,o}(t-1) \ .$$

In these experiments α was set to 0.9. A total of 3190 DNA sequences with either an intron-exon site (768 IE examples i.e. 25%; these are called donors), an exon-intron site (767 EI examples i.e. 25%; also called acceptors) or neither (1655 N examples i.e. 50%) were collected from Genbank[7] v. 64.1. There are 61 attributes for each example: the first attribute has one of the three values N, EI, or IE, indicating the class. The remaining 60 fields are the sequence, starting at position -30 and ending at position +30, relative to the splice site. Each of the fields is filled by one of A, G, T, or C. In a few exceptions, other characters occur in a sequence that

6 On each strand, there are three reading frames, starting with the first, second and third nucleotide. There are altogether six reading frames for a given DNA fragment: three on the coding strand and three on the anti-sense strand.

7 ftp site on Internet: genbank.bio.net. Log in as anonymous and use your email as password.

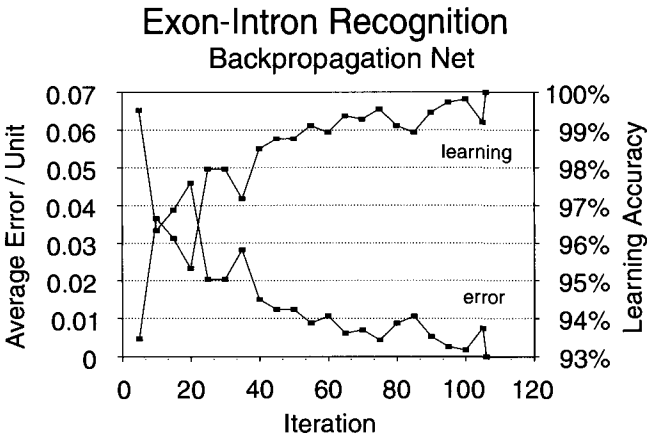


Figure 6: Exon-Intron Recognition with Backpropagation Net.
The net learns to discriminate between DNA fragments with and without exon-intron sites with 100% accuracy after about 100 cycles.

indicate ambiguity among the standard nucleotides. D can be A, G or T; N can be A, G, C or T; S can be C or G; and R can be A or G.

For training, 567 examples of class EI and 567 examples of class N were presented to a backpropagation network with 120 input nodes and one output node. Class EI should produce the output "1", class N "0". The weights were randomly initialised with values between 0 and 1. The learning rate μ was set to 0.5. Figure 6 shows the learning progress during training.

After training, the remaining 1088 examples of class N and 200 examples of class EI were presented to the net for classification. The net predicted 1225 examples accurately (95.1%). Of the 63 falsely predicted examples 6 are false negative and 57 falsely predict the occurance of exon-intron boundaries.

For learning to discriminate intron-exon recognition sites, 567 examples of class IE and 567 examples of class N were presented to a backpropagation network with 120 input nodes and one output node. Class IE should produce the output "1", class N "0". Figure 7 shows the learning progress during training. After training, the remaining 1088 examples of class N and 201 examples of class IE were presented to the net for classification. The net predicted 1220 examples of the test set accurately (94.6%). Of the 69 falsely predicted examples are 11 false negative and 58 falsely predict the occurance of intron-exon boundaries.

The reliability of this simple experiment is corroborated by work done by R. Farber, A. Lapedes and K. Sirotkin[8]. The authors break exon and intron sequences into fragments of lengths 5, 15, 30, 45, 60 and 90 nucleotides. These fragments do not contain any exon / intron boundaries. Sequences are represented to the net

8 R. Farber, A. Lapedes, K. Sirotkin, *Determination of Eucaroytic Protein Coding Regions Using Neural Networks and Information Theory*, **Journal of Molecular Biology**, vol 226, pp. 471-479, 1992.

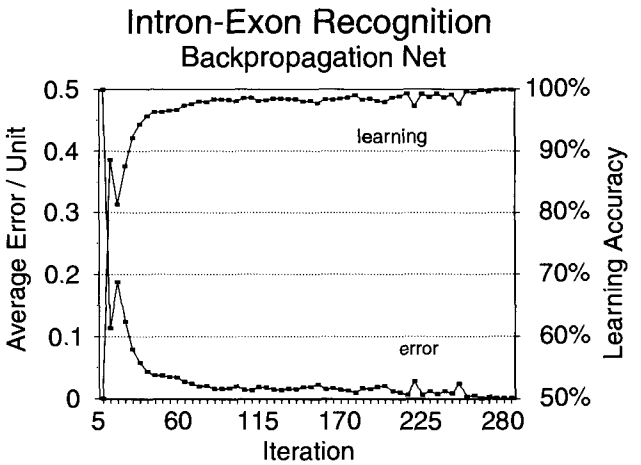


Figure 7: Intron-Exon Recognition with Backpropagation Net.
Learning progress is slower than for exon-intron sites (Figure 6) but reaches the same maximum accuracy of 100% after about 300 iterations.

not as strings of single nucleotides but as chains of codons[9] (dicodons). There are 64 (4096) input nodes for each set of three nucleotides. Alternatively, a frequency representation of codons (dicodons) was used. 64 (4096) normalised frequencies are required to describe the codon (dicodon) distribution of a DNA fragment.

Other information that can be fed into a backpropagation network to characterise DNA fragments are:

- *Frame bias.* In exons and introns there is a distinct, non-random frequency distribution of each of the four nucleotides at the three positions of a codon. Statistical trends and deviations of those can be measured and presented to the net.
- *Word preferences.* The frequency of nucleotide "words" of a given length is different in exons and introns. Introns use extremely common words but exons use relatively rare words. The normalised frequencies of a set of characteristic words and any deviation from the expected values can be used as additional information for the net.

Frequency information from all six reading frames on both strands was used in this application. The best results were achieved with a dicodon frequency representation on all six reading frames. The average prediction accuracy for introns and exons of length 5, 10, 15, 30, 45, 60 and 90 codons is 71%, 79%, 86%, 94%, 98%, 99% and 99%, respectively. Obviously, longer fragments are easier to classify than short ones. Unfortunately, there was no statement made on the homology between training and test set sequences. It is also important to realise the difference between the rate of coverage (i.e. "How many of the true exon positions are among those predicted to occur in exons?") and prediction accuracy (i.e. "How many of the po-

9 Three consecutive nucleotides in an exon code for one amino acid and are termed a *codon*. A *dicodon* contains two codons or six consecutive nucleotides.

sitions predicted to occur in exons actually are found in exons?"). The authors did not report any values for the latter (false positives).

It should be pointed out that it is a different objective to establish an exon / intron boundary and to discriminate between DNA fragments from exons and introns. The latter is mainly a function of the nucleotide distribution whereas the former is primarily based on positional patterns. Yet a more complex question is which putative exons are concatenated in what order to produce the final transcript. This question is of relevance because as yet no prediction method uniquely assigns exon / intron boundaries. Hence, there are usually a number of combinations to evaluate before an accurate prediction can be given to the protein engineer for further analysis to determine the biochemical function of a protein. Further work on this subject was done by Y. Xu, J. R. Einstein, R. J. Mural, M. Shah and E. C. Ueberbacher[10] and by R. Guigo, S. Knudsen, N. Drake and T. Smith[11].

6.2.2. Secondary Structure Prediction

The work of N. Qian and T. J. Sejnowski has been the impetus for much additional experimentation with artificial neural networks for protein secondary structure prediction[12], some of them later reviewed by L. H. Holley and M. Karplus[13] and by S. R. Presnell and F. E. Cohen[14]. The first mentioned authors used a backpropagation network with 40 hidden nodes and an input window of 13 contiguous amino acid residues each represented by a 21 dimensional binary vector. These vectors have only zeros except at one position to indicate the presence of the corresponding amino acid. The 21st dimension stands for a "spacer" residue which substitutes missing residues beyond the C and N termini in order to have a constant input window size at both ends of the sequence. The output unit has three nodes, each representing one of the secondary structures α-helix, β-strand or coil (i.e. no defined secondary structure). Their peak prediction performance on non-homologous proteins is:

$$Q_3 = \frac{p_\alpha + p_\beta + p_{coil}}{N} = 62.7\%$$

10 Y. Xu, J. R. Einstein, R. J. Mural, M. Shah, E. C. Ueberbacher, *An improved system for exon recognition and gene modelling in human DNA sequences*, in **Proceeding of the 2nd International Conference on Intelligent Systems for Molecular Biology** (Eds. R. Altman, D. Brutlag, P. Karp, R. Lathrop, D. Searls), AAAI Press, California, pp. 376-384, 1994.

11 R. Guigo, S. Knudsen, N. Drake, T. Smith, *Prediction of Gene Structure*, **Journal of Molecular Biology**, vol 226, pp. 141-157, 1992.

12 N. Qian and T. J. Sejnowski, *Predicting the Secondary Structure of Globular Proteins Using Neural Network Models*, **Journal of Molecular Biology**, vol 202, pp. 865-884, 1988.

13 L. H. Holley and M. Karplus, *Neural Networks in Protein Structure Prediction*, **Methods in Enzymology**, vol 202, pp. 204-224, 1991.

14 S. R. Presnell, F. E. Cohen, *Artificial neural networks for pattern recognition in biochemical sequences*, **Annu. Rev. Biophys. Biomol. Struct.**, vol 22, pp. 283-298, 1993.

with p_α, p_β and p_{coil} as the correctly predicted amino acid positions for α-helix, β-strand and coil and N as the total number of positions in the prediction set. Another performance measure is B. W. Matthews' correlation coefficient[15]:

$$Q_\alpha = \frac{p_\alpha \cdot n_\alpha - u_\alpha \cdot o_\alpha}{\sqrt{(n_\alpha + u_\alpha) \cdot (n_\alpha + o_\alpha) \cdot (p_\alpha + u_\alpha) \cdot (p_\alpha + o_\alpha)}},$$

with p_α (n_α) as the number of positive (negative) cases that were correctly predicted and o_α (p_α) the number of overpredicted cases i.e. false positives (underpredicted cases, i.e. misses). Similar expressions hold for β-strands (C_β), and coil (C_{coil}). The best performance values achieved are $C_\alpha = 0.35$, $C_\beta = 0.29$ and $C_{coil} = 0.38$. The authors also experimented with varying network topologies (with and without hidden layers; varying input window size) and various input representations (biophysical properties of amino acid residues) but none gave a significantly better performance. One important observation is that adding hidden layers did not improve prediction. This means that in the data used for training there were no second-order rules present but only first-order patterns that can be learned by a simple Perceptron type network. L. H. Holley and M. Karplus independently did similar experiments[16] which confirmed the results of Qian and Sejnowski ($Q3$ = 63%). So did P. Stolorz, A. Lapedes and Y. Xia[17]. Although the prediction accuracy of artificial neural networks is significantly better than that of other classical, statistical secondary structure prediction methods it is still far too low for practical routine use. The main reason for the poor performance is the negligence of long-range interactions that also contribute to secondary structure formation.

An improvement in the neural network secondary structure prediction approach comes with more constraints imposed on the training examples. If only proteins of type α- or β-domain are presented to the net, learning and prediction accuracy rises because the training examples are more homogeneous and folding patterns more easily identified. Still better results are obtained when the training set is not a collection of single protein sequences but a set of alignments of reasonably homologous proteins[18]. In that work, all protein sequences homologous to each of the known protein structures had to be collected from various DNA and protein sequence databases. Depending on the length "alignments of reasonably homology" means that at least 25% to 30% of the residues in secondary structures must be identical. An alignment of homologous sequences provides information about which residues are important for the secondary structure and which may change without disrupting secondary structure formation. Not only one amino acid as in the previous approaches but the relative distribution of permissible residues for each position is given as input to an artificial neural net. A first net learns a three-state secondary structure prediction from a distribution matrix for the middle residue in the in-

15 B. W. Matthews, *Comparison of the predicted and observed secondary structure of T4 phage lysozyme*, **Biochim. Biophys. Acta**, vol 405, pp. 442-451, 1975.

16 L. H. Holley and M. Karplus, *Protein secondary structure prediction with a neural network*, **Proceedings of the National Academy of Science, USA**, vol 86, pp. 152-156, 1989.

17 P. Stolorz, A. Lapedes, Y. Xia, *Predicting protein secondary structure using neural nets and statistical methods*, **Journal of Molecular Biology**, vol 225, pp. 363-377, 1992.

18 B. Rost, C. Sander, *Prediction of protein secondary structure at better than 70% accuracy*, **Journal of Molecular Biology**, vol 232, pp. 584-599, 1993.

put window. A second net follows that learns from the predictions for surrounding residues. Thus, the second net accounts for neighbouring effects and promotes long, contiguous secondary structures over short, fragmented ones. Averaging over nine runs such a cascaded net was able to produce an overall three-state prediction accuracy of $Q3 = 70.8\%$. Although this is a substantial improvement over networks learning from individual sequences this figure is still too low for reliable, routine secondary structure prediction. The program can be accessed via email[19] for prediction.

If one specialises on a single type of secondary structure, e.g. α-helix, prediction accuracy can increase to 78% correctly predicted residues[20] or for prediction of β-turns to 71%[21].

6.2.3. ϕ / ψ Torsion Angle Prediction

It is an attractive idea to make an artificial neural network learn the proper ϕ / ψ main chain torsion angles from primary structure. If there is an immediate correlation between local sequence patterns and backbone conformation a neural network should be able to detect and predict these correlations. The input for the net is the primary structure, e.g. encoded as 20-dimensional bit vectors or distribution matrices as explained in the previous section. The values of the output units are interpreted as ϕ / ψ torsion angles.

A simple, straightforward implementation of this approach was done with a backpropagation network with 60 real valued input units, 72 real valued output units and no hidden nodes. One residue is described by a feature vector of 10 dimensions based on an Eigenvector analysis of 188 biochemical and physicochemical amino acid residue properties[22]. The 60 input nodes thus represent six successive residues, each characterised by 10 real numbers. Each of the 72 output nodes represents one interval for the torsion angles to be learned, e.g. node one is "on" for torsion angle values between 0° and $\frac{360°}{72} = 5°$, node two is "on" for values between 5° and 10°, and so on. Only one corresponding output unit should be activated ("on" i.e. output value 1) for each example whereas all other output nodes should be "off" (output value 0). This setting configures a backpropagation network to learn the main chain torsion angle of the fourth residue in a fragment of six consecutive residues to an accuracy of $\pm$ 5°. For training, 3536 hexa-tuples selected from the core of 104 broadly non-homologous proteins together with their ϕ (or ψ, respectively) values were presented to the net. The best learning performance during training for either ϕ or ψ angles did not exceed 6%, i.e. the net could not

19 Send an email with the word "help" in the body to the address predictprotein@embl-heidelberg.de and follow the guidelines returned.

20 S. Hayward and J. F. Collins, *Limits on A-Helix Prediction With Neural Network Models*, **Proteins**, vol 14, pp. 372-381, 1992.

21 M. J. McGregor, T. P. Flores, M. J. E. Sternberg, *Prediction of β-turns in proteins using neural networks*, **Protein Engineering**, vol 2, no 7, pp. 521-526, 1989.

22 A. Kidera, Y. Konishi, M. Oka, M. Ooi, H. A. Scheraga, *Statistical Analysis of the Physical Properties of the 20 naturally Occurring Amino Acids*, **Journal of Protein Chemistry**, vol 4, no 1, pp. 2355, 1985.

significantly relate a main chain torsion angle to the sequence of its six neighbouring residues. Taking the relatively short monitoring window into consideration this result is not surprising. After all, a window of 13 residues was just shown to be informative enough to predict a three-state secondary structure at only 63% accuracy and here fewer residues are used in an attempt to accurately discriminate many more (72) classes of output values.

J. Vanhala and K. Kaski experimented with this approach somewhat more successfully[23]. They divided the ϕ / ψ map into 100 slightly larger squares, each spanning 36° by 36°. Here, the input is a 22-dimensional bit vector. Two additional "null" residues are used, one for the end of sequences and another for cis-proline. In the smallest net, 154 input nodes were used corresponding to a 7 residue fragment, plus 10 hidden nodes and 100 output nodes. Their training set contains 46 strands from 43 proteins that have a total of 6964 residues. For a test on bovine pancreatic trypsin inhibitor (BPTI; Brookhaven database code 5PTI) with 58 residues their backpropagation network predicts the ϕ / ψ regions of about half of the residues correctly. Both α-helices and the β-strand are properly predicted as are the four ends of the three sulphur-bridges. Because a few angles are predicted with the maximum error of 180° the suggested conformation of BPTI is elongated and not globular. Another test was done on human lysozyme (Brookhaven database code 1LZ1) with 130 residues. Again, the predicted overall structure is elongated instead of globular. Only the closely packed core structures are reasonably similar to the native conformation. Finally, their result for avian pancreatic polypeptide (Brookhaven database code 1PPT) with 36 residues and one α-helix and one β-strand is given. Both secondary structures are predicted correctly but their relative positions and packing are wrong.

The problem with ϕ / ψ prediction remains that in the real case of predicting the conformation of an unknown protein one does not known which angles are predicted reliably and which ones are not. To overcome this difficulty one could try to establish a confidence measure based on the amount of the signal of the output units. Further, biochemical and physico-chemical amino acid residue properties could be used to represent the input fragments which might enhance the results. Alternatively, backbone conformation can be represented by distance matrices[24], although in that particular work the number of weights in the network was so large compared to the number of training examples that the net only memorised the examples but did not actually generalise or learn them.

23 J. Vanhala and K. Kaski, *Protein Structure Prediction System based on Artificial Neural Networks*, in **Proceeding of the 1st International Conference on Intelligent Systems for Molecular Biology** (Eds. L. Hunter, D. Searls, J. Shavlik), AAAI Press, California, pp. 402-410, 1993.

24 H. Bohr, J. Bohr, S. Brunak, R. M. J. Cotterill, H. Fredholm, B. Lautrup, S. B. Peterson, *A novel approach to prediction of the 3-dimensional structures of protein backbones by neural networks*, **FEBS Letters**, vol 261, pp. 43-46, 1990.

Nodes	Topology	
6	1D	6 x 1
8	1D	8 x 1
9	1D	9 x 1
	2D	3 x 3
16	2D	4 x 4
27	3D	3 x 3 x 3
64	3D	4 x 4 x 4

Table 3: Kohonen Network Topologies used for Super Secondary Structure Classification. The connection pattern between nodes can be linear, planar, three- or more dimensional.

6.2.4. Super-Secondary Structure Detection

A Kohonen net was used to learn classifications of small super-secondary structures (pairs of α-helices) with common geometrical, topological and chemophysical properties from the PRL database described in section 4.2.1. The following node topologies were used (Table 3).

The remaining parameters for the Kohonen network were as follows.

```
#define VECTORSIZE 32      /* dimension of input vector */
#define R1 0.05            /* 1st ratio parameter */
#define R2 0.50            /* 2nd ratio parameter */
#define L1 0.10            /* 1st learning rate */
#define L2 0.05            /* 2nd learning rate */
#define L3 0.01            /* 3rd learning rate */
#define D1 3.1             /* 1st distance */
#define D2 2.1             /* 2nd distance */
#define D3 1.1             /* 3rd distance */
#define MAX_ITER   100     /* maximum number of cycles */
#define INIT_DIST  1.4     /* initial neighbor distance */
#define LEARN_RATE 0.2     /* initial learning rate */
```

There is a discrete time dependence for the learning rate and neighbor function which is controlled by the 1ST and 2ND RATIO PARAMETERs. For the first 5% all learning instances presented to the net (i.e. the product of training examples and learning cycles) the learning rate is 0.1 and the neighbour distance threshold 3.1. This means that the state vectors of all neurons with a Hamming distance less than 3.1 (corresponding to a 5 x 5 square of nodes in two dimensions) are adjusted by 10% of the difference between their original state vector and the one of the current example. The following 45% of all learning instances are trained with a learning rate of 5% and a neighbor distance threshold of 2.1, corresponding to a 3 x 3 square of nodes in two dimensions. The last 50% of all learning instances are subject to a learning rate of 1% and a neighbour distance threshold of 1.1 which means that only the four orthogonally adjacent neighbours in two dimensions are updated with the winner node. The purpose of this parameter setting is to train a Kohonen net fast and coarsely at the beginning and slowly fine-tune it towards the end of a run. Usually there were no major changes in the clustering after 50 generations.

Node	1	2	3	4	5	6	Ratio
1	–	–	–	–	0.03	0.97	0.21
2	–	–	–	0.15	0.85	–	0.09
3	–	–	–	1.00	–	–	0.13
4	–	–	–	–	0.88	0.12	0.12
5	–	–	–	0.32	0.68	–	0.13
6	–	–	0.89	0.11	–	–	0.06
7	–	–	–	–	0.31	0.69	0.11
8	50	–	–	0.25	0.25	–	0.03
9	71	0.29	–	–	–	–	0.12

Table 4: Cross-Reference of Two Kohonen Clusterings.
This comparison of two runs with a 1D (6 x 1) and 2D (3 x 3) topology reveals that cluster 1 of the 2D run corresponds closely to cluster 6 of the 1D run. Altogether, 23 cross-reference tables were analysed to assemble a consensus classification of 16 runs.

As with other clustering techniques it would be unwise to take the classification of a single run because the result is likely to depend on the initial configuration (net topology, number of nodes, initial weights, etc.). Hence a number of runs were performed with the parameter settings described above using the net topologies of Table 3. To be able to compare runs with different net topologies the clustered examples were cross-referenced as in Table 4.

The final consensus classification of 145 helix-pair super-secondary structures based on 16 Kohonen runs with the net topologies of Table 3 is shown in Table 5.

This Kohonen consensus classification is in good agreement with the IPSA consensus classification of section 4.2.1.3. Similarly, Kohonen networks can be applied to other classification problems in molecular biology.

One interesting extension of this method is the combination of a Kohonen network and a backpropagation network which is called *counter propagation*:

- a Kohonen network is used to "spread out" and classify a set of objects and then
- a backpropagation network receives the Kohonen classification output as its input and tries to predict the Kohonen classification based on the original data.

This concludes the current list of applications of artificial neural networks in molecular biology. As one can see, artificial neural networks are versatile, general purpose algorithms. Even in the presence of noise they are capable of detecting hidden rules and patterns if any are present in the data to be analysed. Literature search (e.g. with Medline or on the Internet) will guide the reader to many variations of the work presented here and other recent work using artificial neural networks in molecular biology.

Kohonen Cluster	Helix-Helix Pairs (IPSA Consensus Classification)	IPSA Cluster
a	9 (A), 14 (A), 26 (A), 28 (A), 30 (A), 31 (A), 33, 47 (A), 55 (A), 59 (A), 60 (A), 61 (A), 65 (A), 66 (A), 67 (A), 71 (A), 72 (A), 73 (A), 76 (A), 77 (A), 87 (A), 111 (A), 138 (A)	A
b	13 (B), 24 (B), 25 (B), 39 (B), 42 (B), 43 (B), 57 (B), 63 (B), 69 (B), 75 (B), 83 (B), 85 (B), 92 (B), 94 (B), 95 (B), 101 (B), 102 (B), 109 (B), 133 (B), 134 (B)	B
c	1 (C), 17 (C), 22 (C), 48 (C), 54 (C), 80 (C), 82 (C), 88 (C), 106 (C), 119 (C)	C
d	20 (D), 51 (D), 56 (D), 62 (D), 68 (D), 74 (D), 118 (C), 121 (C), 128 (C), 142 (D), 143 (D), 144 (B)	B, C, D
e	40 (B), 108 (C), 113 (D), 117 (D), 120 (D), 139 (D), 145 (B)	B, C, D
f	55 (A), 59 (A), 60 (A), 61 (A), 65 (A), 66 (A), 67 (A), 71 (A), 72 (A), 73 (A), 76 (A), 77 (A)	A
g	83 (B), 85 (B), 92 (B), 101 (B), 102 (B)	B
i	9 (A), 26 (A), 28 (A), 30 (A), 111 (A), 131 (A)	A
j	33, 84, 86, 87 (A), 97, 103 (A)	A, nc
k	27, 29, 110 (A), 123	A, nc

Table 5: Helix Pair Kohonen Consensus Classification.
The numbers of the helix pairs correspond to Table 2 in section 4.2.1.3. Capital letters refer to IPSA clusters (section 4.2.1), lower case letters denote Kohonen consensus clusters. "nc" stands for "no class". Some examples occur in more than one cluster because some nets had more nodes and could therefore create and fill more clusters with less examples. For this consensus classification those smaller clusters are listed individually instead of having them merged with larger Kohonen clusters.

7. Summary, Conclusion & Prospects

By now the attentive reader will have encountered five important, modern information processing technologies:

- Symbolic knowledge processing (chapter 2).
  Symbols represent complex entities or concepts and can be manipulated according to their semantics.
- Automatic logic inference (chapter 3).
  The laws of logic are used to infer and specialise new knowledge from established rules and facts.
- Machine learning and induction (chapter 4).
  Inductive algorithms generalise classified or unclassified observations and induce regularities and interdependencies from them.
- Evolutionary computation (chapter 5).
  Letting models evolve in a way analogous to natural evolution can produce solutions to otherwise intractable optimisation problems.
- Artificial neural networks (chapter 6).
  As an easy and practical alternative to classic, statistical analysis artificial neural networks can detect hidden correlations in analytically intractable sets of data.

The reader has also seen a number of information processing issues in molecular biology being approached by the aforementioned techniques, e.g. :

- Computerised semantic storage, manipulation and retrieval of information on various levels in molecular biology.
- Computational analysis of protein purification techniques, protein folding, protein structure and the biochemical function of proteins.
- DNA sequence analysis and pattern recognition.

In the recent past the focus in biosciences has been extended from experimental work in the laboratory to increasing use of computerised information processing and simulation tools. In terms of the history of science the information processing techniques described in this book have only quite recently been invented and are still being constantly refined. However, as we can see from some of the applications described here those new ways of refining our knowledge can already deliver results.

Symbolic knowledge processing will eventually lead to a revolution in our way of handling and organising the myriad pieces of information which are continuously produced by the scientific community world wide. Much more computerised information and more intelligent search and retrieval options will gradually supplant traditional paper copy journals and books. In fact, parts of this book have been

made electronically accessible for the first international course on biocomputing[1] of the Globewide Network Academy in the Virtual School of Natural Sciences on the Internet. People from all over the world are able to participate as students and instructors, sitting comfortably in their offices or at home and browsing hypertext material over the Internet with a few mouse clicks. A true realisation of the world as a global village!

Machine learning will help to analyse and organise the abundance of new raw data emerging from large scale sequencing projects and link this kind of information with other sources. A more detailed and grounded overall picture on the composition and performance of biological systems will emerge from this. – Let us use our new instruments wisely and efficiently.

1 For more information (participation, registration) contact the author, who is one of the authors on biocomputing at the GNA, VSNS. Email: steffen@chemie.fu-berlin.de, steffen@mpimg-berlin-dahlem.mpg.de, or steffen@mycroft.rz-berlin.mpg.de.

8. Appendix

8.1. Internet Entry Points for Biocomputing

The easiest way to contact professionals in biocomputing world wide and to look for interesting new developments and *software* that is often *freely available* for non-commercial use is via the Internet. To participate, do the following:

- Get access to a computer which must be linked to the Internet. This could be done with a PC and a modem. Usually you have to apply for your own user id from the system administrator of your local institution (company, service provider, research institute, university).
- Once logged in, start a world wide web browser like WWW, MOSAIC or NETSCAPE. At least one of these should be available for all major operating systems (Unix, DEC, Mac, Windows, DOS). If your system manager has not yet installed such a program ask him / her to do so.
- Connect to one of the entry points mentioned below. The actual procedure will be explained in the help facility of your web interface or simply ask someone who knows. In general, look for a command like "Goto", "Open" or "Connect". From there you can continue your journey through the World Wide Web resources per mouse click. There are also searchable catalogues and indices of other computers and material connected to the World Wide Web to guide you on your journey through the Internet.
- The author's home page can be found at http://www.chemie.fu-berlin.de/user/steffen or at http://mycroft.rz-berlin.mpg.de/~steffen.

8.1.1. Information, Literature, References

FTP Site Springer Verlag	file://trick.ntp.springer.de/pub
Table of Contents Springer Verlag	file://trick.ntp.springer.de/pub/TOC
BIOSCI and Bionet Electronic News	gopher://gopher.bio.net/
Human Genome News	gopher://merlot.welch.jhu.edu/11/Genome/hgnews
Classes of GNA, VSNS in BioMOO	http://bioinformatics.weizmann.ac.il:70/1s/biomoo
Biological Journals	http://golgi.harvard.edu/journals.html
Principles of Protein Structure Course	http://seqnet.dl.ac.uk:8000/vsns-pps
Evolutionary Computation Journal	http://www-mitpress.mit.edu/jrnls-catalog/evolution.html
The Genetics Algorithm Archive	http://www.aic.nrl.navy.mil/galist
The Hitchhiker's Guide to Genetics Algorithms	http://www.cis.ohio-state.edu/hypertext/faq/usenet/ai-faq/genetic/top.html

Computer Science Laboratory and Artificial Intelligence Centre, CA	http://www.csl.sri.com/ and http://www.ai.sri.com/aic
List of Libraries on the Internet	http://www.laum.uni-hannover.de/iln/bibliotheken/
Cinema in Berlin	http://www.netcs.com/Kino/
Human Genome Project Information	http://www.ornl.gov/TechResources/Human_Genome/home.html
Virtual Library	http://www.w3.org/hypertext/DataSources/bySubject/Overview2.html

8.1.2. Institutions Dealing with Molecular Bioinformatics

Bioinformatics at the Weizmann Institute, Israel	gopher://bioinformatics.weizmann.ac.il/1
Jobs at the European Bioinformatics Institute	gopher://bioinformatics.weizmann.ac.il/11m/researchers/EBI
Artificial Life at the Santa Fe Institute	http://alife.santafe.edu/
Harvard Physics Department	http://string.harvard.edu/
Information Technology Branch of NIH	http://text.nlm.nih.gov/wwwitb/itbhome.html
The Baylor College of Medicine Home Page	http://www.bcm.tmc.edu/
European Commission Research Programme Information	http://www.cordis.lu/ or http://www.echo.lu/
Faces and Names of EBI Personnel	http://www.ebi.ac.uk/ebi_docs/staff.html
European Computer-Industry Research Centre	http://www.ecrc.de/
John Hopkins University Bioinformatics WWW Server	http://www.gdb.org/hopkins.html
Institute of Molecular Biotechnology Jena Home Page	http://www.imb-jena.de/
National Library of Medicine of the US (NLM, NIH)	http://www.nlm.nih.gov/
Computer Science Branch of NLM (NIH)	http://www.nlm.nih.gov/lhc.dir/csb.dir/csb.html

8.1.3. Databases

SWISS-PROT Protein Sequence Database	http://expasy.hcuge.ch/sprot/sprot-top.html
SWISS-PROT Protein Sequence Database	http://expasy.hcuge.ch/www/disclaimer.html
Genome Database (Repository for HUGO)	http://gdbwww.gdb.org/
Integrated Genomic Database (IGD) Documentation	http://genome.dkfz-heidelberg.de/igd-docs/start_igd_doc.html
Brookhaven Protein Data Bank PDB (USA)	http://www.pdb.bnl.gov/

8.1.4. Search Tools for World Wide Web and FTP Sites

Index of Biology Related Servers (BOING)	gopher://gopher.gdb.org:3005/7
Veronica (Search Gopher Space)	gopher://gopher.harvard.edu/11/ .othergophers/Veronica
Search Gopher Space by Titleword(s)	gopher://veronica.sunet.se:2347/7
Finding Topics on the Internet	http://alpha.acast.nova.edu/findtopic.html
Searchable WWW Catalogue	http://cui_www.unige.ch/w3catalog
Searchable WWW Catalogue	http://cuiwww.unige.ch/cgi-bin/w3catalog
WWW Search Engines	http://cuiwww.unige.ch/meta-index.html
Searchable Index for the WWW VL Biosciences	http://golgi.harvard.edu/htbin/biopages
World Wide Web Servers	http://info.cern.ch/hypertext/DataSources/ WWW/ Servers.html
Index of Biology Servers from USGS	http://info.er.usgs.gov/network/science/ biology/index.html
ArchiePlex	http://web.nexor.co.uk/archie.html
Infoseek Netsearch	http://www2.infoseek.com/
Yahoo Netsearch	http://www.yahoo.com/
Web Crawler Search	http://www.webcrawler.com/

8.1.5. Software Resources

Artificial Life Software	ftp://alife.santafe.edu/pub/SOFTWARE
Software for the IBM PCs	gopher://ftp.bio.indiana.edu/11/IUBio-Software%2bData/chemistry/ibmpc
Molecular Biology	gopher://ftp.bio.indiana.edu/11/IUBio-Software%2bData/molbio
Other Gopher Sites for Molecular Biology	gopher://ftp.bio.indiana.edu/11/Other-Bio-Gophers
Biology Internet Resources	gopher://genome-gopher.stanford.edu/11/topic
List of More Biogophers	gopher://gopher.gdb.org/11/biogophers
FTP Sites in Biology	gopher://gopher.uni-kl.de/11/Fachbereiche/Biologie/bioftp
Other BioGophers	gopher://gopher.uni-kl.de/11/Fachbereiche/Biologie/biogopher.andere
FTP Sites For Biology	gopher://merlot.welch.jhu.edu/11/bioftp
Medical and Biology Related Gophers	gopher://merlot.welch.jhu.edu/11/biogophers
Medical and Biology Related Gophers	gopher://odie.niaid.nih.gov/11/gophers/bio
Molecular Biology Computational Resource	http://condor.bcm.tmc.edu/home.html
Evolution Strategies Software	http://lautaro.fb10.tu-berlin.de/ or ftp://ftp-bionik.fb10.tu-berlin.de/
Related Papers And Technical Reports On Logic Programming Tools	http://www.ecrc.de/eclipse/html/reports.html
Search On-line Descriptions of DOS Software Archive	http://www.gdb.org/bio/search/FILT/simtel.html
Software Archives	http://www.gdb.org/Dan/softsearch/biol-links.html
Data Mining Tools of the University of Birmingham	http://www.cs.bham.ac.uk/~anp/TheDataMine.html
Software Agents of the University of Birmingham	http://www.cs.bham.ac.uk:80/~amw/agents
Machine Learning at the University of Vienna	http://www.ai.univie.ac.at/oefai/ml
More Software Archives	http://www.gdb.org/Dan/softsearch/graph-links.html
Biological Software and Data FTP Archives	http://www.gdb.org/Dan/softsearch/softsearch.html
Molecules 'R' US	http://www.nih.gov/htbin/pdb
The NIH Guide to Molecular Modelling	http://www.nih.gov/molecular_modeling/gateway.html#apps_anchor
Molecular Modelling Tools and Databases	http://www.nih.gov/molecular_modeling/mmhome.html

Index

This index should be used in combination with the table of contents.

λ-phage 122
1/5 rule 219
1d-NMR 100
2D gel electrophoresis 72
2d-NMR 100

abduction 32
active site 11
affinity chromatography 73
algebraic differentiation 17
alphabetic variant 114
AM 5
amino acids 64
ammonium sulfate precipitation 68
analogy 117
and-introduction 144
antecedent 112
antibiotic 50
antibiotic resistance 50
antibiotic therapy 29
antiparallel β-sheets 130
artificial intelligence 3
artificial life 5, 212
artificial neural network 5
AutoClass 159

backpropagation network 273
backward chaining 32
Bayes theorem 156, 160
best-first search 33
biochips 9
biocomputing 292
biocomputing course IX, 292
biosensors 9
biotechnology 8
blackboard architecture 104, 108
breadth-first 33
breadth-first tree search 140
building blocks 213

capping 87
Cartesian coordinates 247
category utility 155
certainty factor 27
charge 67
Checkers 7
chemotaxis 43
classifier systems 8
Classit algorithm 152
clause 116
Cobweb 152
cohesive-end method 49
competitive learning 276
Computer Database 35
conceptual clustering 152
conjunction 112
consequent 112
constraint propagation 54
constraint satisfaction algorithm 140
constraints 54
cooperative revision 94
COSY 100
counter propagation 289
crossover 212, 216, 237

data mining IX
database 9
decision tree 6
declarative programming 122
deduction 32
depth-first 33
diabetes mellitus 48
disjunction 112
DNA / RNA 11
DNA ligase 48
docking 267
downstream process 62
drug design 149

E. coli genetics 25
elitist generation replacement 214
elitist replacement 252
Eliza 5
email X, 292
endonuclease 48
enhancer element 11

entropy 30
enzyme activity 64
evolution strategy 6, 219
evolutionary computation 211
existential quantifier 113
exon 11, 87, 280
exonuclease 48
expert system 3, 26
expression 37
extraction, symbolic 17

feed-forward network 272
first order predicate calculus 111
fitness function 212
Foil, ILP System 147
forward chaining 32
forward pruning 140
frame problem 25
frames 6, 44
function optimizer 219
functions 21
functions in programs 6

gel electrophoresis 70
gel filtration 72
gene expression 37
gene regulation 37
gene technology 48
generation replacement 252
Genesis 237
genetic algorithm 6, 212
genetic programming 8, 220
genome mapping 11
genomic information 11
genotype 219
Global Network Academy X, 292
Golem 146
greek key motif 129
ground instance 114

hairpin motif 129
handedness 130
heat treatment 69
hepatitis serology 29
hidden layer 274
homology 285
hybrid approach 247
hydrophobicity 65

ID3 30, 200
ILP 142
implication 112
induction 29
inductive logic programming 142
inference 29
insertional inactivation 50
insulin 47
internal medicine 29
Internet X, 292
intron 11, 87, 280
ion exchange chromatography 72
IPSA 162
isoelectric focusing 72

Jakob-Monod model 37
jelly roll motif 130

knowledge acquisition 28
knowledge base 6
knowledge engineering 21
knowledge engineering bottleneck 28
knowledge processing 29
knowledge representation 6

Laboratory Notebook 41
lac operon 39
least-general generalisation 145
left-handed conformation 135
ligation 49
Lisp 5, 220
literals 113
local twist operator 260
logical and 112
logical or 112
long range interactions 12, 185

machine learning 7
Macsyma 5
meander motif 129
Medical Subject Headings 44
Medline 35
member predicate 118
Mendelian theory 21
message passing 24
methods 24
Mobal, ILP System 147
modus ponens 32, 114
molecular dynamics X
molecular mechanics X
molecular modelling X
molecular pathology 25
Molgen 46

Monte-Carlo search 236
most general unifier 115
mRNA 37
mutate 212
mutation 216
mutation rate 219

nanotechnology XI
negation 112
neural network 5
NOESY 101
nomenclature 41
non-monotonic reasoning 29
nuclear Overhauser effect 101

object-oriented 9, 24
object-oriented representation 41, 44
oncology 29
operator 38
operators in Molgen 52
operon 37
or-introduction 144

PaperChase 35
parallel α/β-sheets 134
partial evaluation 132
partitioned semantic net 24
pattern recognition 279
pBR322 plasmid 50
perceptron 273
pH 68
phenotype 219
pK 67
plasmids 50
polyadenylation 87
potential energy function 250
predicate calculus 7
predicate invention 144
predicate logic 111
predicate-based database 9
predicates 6
prediction 279
prediction of a-helices 150
primary structure 12
probability calculus 7
probability matching 156
probability maximising 156
processes, reasoning on 25
production rules 6
production system 27
Prolog 5, 111
promoter 37
promoter site 11
proof 114
prophage 124
propositional calculus 113
Protean 104
protein folding problem 12
Protein NMR Assistant 108
protein purification 62
Protein Purification Program 63
protein structure analysis 111
protein targeting 199
protein topology 133
proteins 11
pulmonary function 29

QSAR 147
qualitative process theory 25

r.m.s. fit 185
reasoning by analogy 117
recursion 16
relational database 9
representation, functional 21
representation, object-oriented 24
reproduction 212, 216
resolution 116
restriction enzymes 48
ribosomes 37
right-handed conformation 135
RNA polymerase 37, 50
rules 26
rules of inference 114

salting-in 69
salting-out 68
schema 42
schemata 213
schemata theorem 214
scripts 6, 25
search 33
secondary structure 12, 101
secondary structure prediction 12
secondary structure prediction performance measure 285
selection operator 251
self-modifying programs 14
semantic net 6, 41
semantic nets 22
Senex 44
sequence alignment 11

SGML 41
shape of protein 67
side chain 64
signal sequences 199
simulated annealing 7, 230
size of protein 65
Skolem function 116
softbots 5
software robots 5
splice site 11
splicing 87, 280
statistics 7
steady state replacement 214, 252
strand topology 133
strategy operator 52
supervised learning 30
supervised learning mode 272
symbolic computation 3, 15
symbolic programming 13
syntax 111

tertiary structure 12
theorem 114
thermolysin inhibitors 149
titration curve 68
torsion angle 247
torsion angle search space 249
total generation replacement 214
total replacement 252
transcription 37
transformation 56
translation 37
travelling salesman 231
triplets VIII
truth maintenance 28
two point crossover 251

ultracentrifugation 73
ultrafiltration 74
unification 114
uniform crossover 251
universal quantifier 113
universal specialisation 114
unsupervised learning 30
unuspervised learning mode 272

Virtual School of Natural Sciences X, 292

well-formed formula 111
World Wide Web Servers X, 293

XOR problem 273

This book attempts to provide an overview of advanced computer applications derived from and used in biosciences and in doing so to help define and promote the newly emerging focus on molecular bioinformatics. It is intended to serve as a guidebook for both biologists who are about to turn to computers and for computer scientists who are looking for interesting application areas. The reader will find a variety of modern methodological approaches applied to a number of quite diverse topics. The book is also intended to inspire further research in this exciting and vastly expanding research area.

ISBN 3-11-014113-2