Heuristic Algorithms in Bioinformatics

Jakob Vesterstrøm

PhD Thesis

Bioinformatics Research Center (BiRC)
Department of Computer Science
University of Aarhus
Denmark

Heuristic Algorithms in Bioinformatics

Ph.D. Thesis
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
Ph.D. Degree

by
Jakob Vesterstrøm
November 25, 2005

Abstract

Many bioinformatics problems are very hard to solve optimally and within polynomial time of their size. This motivates the use of heuristics in bioinformatics. The present thesis constitutes research into the design and analysis of efficient heuristic algorithms of practical usage for solving biological problems.

The research made in this thesis can be related to three main objectives: 1) The design efficient tools for solving relevant bioinformatics problems, 2) The exploration of stochastic search algorithms, and 3) The use of bioinformatics tools get an increased knowledge about the world from a bioinformatics perspective.

The research relates to two bioinformatics problems: Comparison of 3D protein structures also known as structural comparison, and physical mapping, which is a combinatorial problem that originates from the determination of whole genome sequences.

The thesis is structured in two parts: Part I, in which the background is introduced for the algorithms used and the problem domains investigated. Part II, in which the four papers I have produced are presented. One paper has been accepted for publication in the *Journal of Computational Biology*, two have been published in the proceedings of the *Congress on Evolutionary Computation*, and one is currently in submission.

The research on structural comparison resulted in a journal paper, written with William R. Taylor, and a tool, called FASE (Flexible structural Alignment from Secondary structure), for comparison, superposition, and alignment of protein structures. The chosen focus for structural comparison is reflected in the features of FASE: FASE can find non-sequential structure similarities with a novel, local application dynamic programming, FASE uses a new idea to search the space of structural alignments by making initial alignments of the protein structures according to their SSE definitions. This improves the runtime, while still enabling discovery of very subtle and remote similarities. A novel consensus scoring method, combining five measures for structural similarity, was introduced in order to increase robustness in similarity assessment. The significance of the found similarity values was determined using the SCOP classification as reference. In conclusion, FASE was shown to be robust with defaults parameters, and have a better time-complexity than competitive methods. FASE has been tested on

standard problems and demonstrated the ability to consistently find weak similarities and circular permutations (CPs). In all areas tested, the performance was equal to or better than the best alternative method.

In a subsequent research paper, FASE was applied to the detection of novel pairs of proteins related by a CP. This required the construction of measures for the detection of CP alignments and for finding similarity to the identity alignment. These measures was used to detect CPs, and to assess their likelihood. The novel CPs were presented, analysed, and evaluated, and many plausible cases were found. An extensive comparative analysis with already known and classified CPs in SCOP further confirmed these findings.

Stochastic search algorithms were further explored in the research on the physical mapping (PM) problem. This research was published in a conference paper. By applying evolutionary algorithms, simulated annealing, and local search (LS) on the PM problem, it was found that 1) The algorithms had comparable performance. 2) Considering the ambiguity of the problem and experimental errors, the performance of the algorithms was close to optimum. 3) Many existing fitness functions correlated badly with optimal solutions. 4) The choice of fitness function and low experimental error rates were the most important factors in obtaining good results. Moreover, since many problems have search spaces with properties similar to PM, it was concluded that LS approaches, somewhat surprisingly, will perform as well as evolutionary algorithms on many new problems.

With Rene Thomsen I made a comparative investigation of the performance of differential evolution, particle swarm optimisation (PSO), and a standard evolutionary algorithm. In this study the focus was entirely on the exploration of stochastic search algorithms, rather than a bioinformatics problem. This research was a continuation of the algorithmic part of the previous study, and of the research made in my masters thesis on PSO. The resulting paper was published in a conference paper, was it was the first reported study comparing these three well-known algorithms. The results of our experiments demonstrated that differential evolution clearly outperformed the other methods regarding performance and robustness on a diverse suite of 34 well-known, high-dimensional, highly multi-modal test problems.

Acknowledgements

Thanks to René Thomsen, William R. Taylor, Christian Nørgaard Storm Pedersen, Wouter Boomsma, Thiemo Krink, Rasmus Ursem, EVALife, BiRC, and Mathbio people, and not least Kristine.

Jakob Vesterstrøm,
Århus, November 25, 2005.

Contents

Abbreviations

CATH	class, architecture, topology, homology
CLS	classic local search
CPs	circular permutations
DNA	deoxyribo-nucleic acid
DE	differential evolution
DP	dynamic programming
EAs	evolutionary algorithms
EC	evolutionary computation
EP	evolutionary programming
ES	evolution strategies
EXS	exhaustive search
FASE	**F**lexible structural **A**lignment from **S**econdary structure **E**lements
FSSP	**F**old classification based on **S**tructure-**S**tructure alignment of **P**roteins
GA	genetic algorithm
GP	genetic programming
HGP	humane genome project
HC	hill-climbing
LS	local search
NP	nondeterministic polynomial time
PDB	protein data bank
PDB ID	PDB identification code
PM	physical mapping
PBSMs	population based search methods
PSO	particle swarm optimisation
RMSD	root mean square deviation
RNA	ribo-nucleic acid
mRNA	messenger RNA
RS	random search
SCOP	structural classification of proteins
SA	simulated annealing
STS	sequence tagged site
us	unit square
WGS	whole genome sequencing

Part I.

Overview

Chapter 1

Introduction

In computer science, it is a very fundamental challenge to make algorithms that find provably good solutions using a provably bounded amount of computation time. A *heuristic*, in contrast, is an algorithm that gives up one or both of these goals [166, 124]. The term bioinformatics means using "*techniques from applied mathematics, informatics, statistics, and computer science to solve biological problems*" [165, 152]. The title and goals of this thesis are in accordance with the above definitions: It constitutes research into the design and analysis of efficient heuristic algorithms of practical usage for solving biological problems. In this process, increased knowledge about the algorithms used and the problems investigated is also anticipated.

The motivation for resorting to heuristics is the fact that many problems – and this includes many bioinformatics problems – are very hard to solve optimally and within polynomial time of their size; stated more popularly: they are hard to solve fast *and* well. Examples are molecular docking [103], multiple sequence alignment [160], and structural alignment [84]. The theoretical and computational reason for this is that these problems are NP-hard [113] (docking is conjectured but not proved to be NP-hard [152]), and such problems are believed to be unsolvable in polynomial time.

It lies in the nature of most efficiently designed algorithms in general, and heuristics in particular, that they make extensive use of any structure or knowledge that can be extracted from the problem domain. This is how a good heuristic may find near-optimal solutions even for large NP-hard problem instances relatively fast. However, the more complex the problem, the harder it is to integrate it into the heuristic, and since many bioinformatics problems are indeed complex, much work goes into analysing and understanding the problem, when solving bioinformatics problems heuristically. Simply defining the problem is not sufficient for this purpose; rather, it must be understood more extensively. Moreover, because heuristics make no guarantees on solution quality and computation time, much effort must also be devoted to benchmarking their performance by empirical means to obtain reliable knowledge of solution time and quality.

1.1. Research Objectives

The research made in this thesis can be related to a set of main objectives. Below, these objectives are outlined along with the research that was conducted to achieve them.

An objective of the research presented in my thesis was to ***explore stochastic search algorithms***. This was done in work together with René Thomsen from the Department of Computer Science, University of Aarhus, which was published in the paper *A Comparative Study of Differential Evolution, Particle Swarm Optimisation, and Evolutionary Algorithms on Numerical Benchmark Problems* [159]. The performance of the mentioned methods as well as simulated annealing (SA) was investigated on a wide range of numerical benchmark problems. Indeed, possessing an overview of the relative performance of a set of algorithms is central if attempting to improve some of them. Surprisingly, prior to our study, no thorough comparisons had been made of these heuristic optimisation methods. In particular, differential evolution (DE) as a new promising optimisation method, had not been compared to evolutionary algorithms (EAs) on a wide range of problems. This work was a natural extension of the analysis and improvements suggested in my masters thesis for particle swarm optimisation (PSO) [157]. One of the algorithms proposed in the thesis, the arPSO [126], was included in the comparison.

The exploration of stochastic search algorithms was also the topic of the study made on physical mapping (PM), which was published in the paper *Physical mapping using simulated annealing and evolutionary algorithms* [154]. PM is a bioinformatics problem that originates from the determination of whole genome sequences; the task is to order a set of genetic markers along a genome, based on their presence in or absence from a set of long and overlapping DNA-segments. Local search (LS), SA, and EAs were applied to solve the PM problem, and their performance were analysed and discussed.

The basic aspects of the relevant optimisation algorithms are described in chapter 2, and the paper with the comparative study of the algorithms is presented in appendix E. Moreover, PM is described in detail in chapter 4; the above mentioned paper on optimisation strategies for PM is presented in appendix D.

Another objective of the research was to ***design efficient tools for solving relevant bioinformatics problems***. During the latter part of my Ph.D. project, I have developed the algorithm and program FASE (Flexible structural Alignment from Secondary structure Elements) for structural comparison of proteins. Obviously, this product is the most significant contribution in fulfilling the above mentioned objective. In chapter 5 on structural comparison, the method is introduced. In addition, the method has been published in the paper *Flexible Secondary Structure Based Protein Structure Comparison Applied to the Detection of Circular Permutation*, which has been accepted for the *Journal of Computational Biology* [158] with William Taylor from The National Institute for Medical

Research, Mill Hill, London as co-author. In this paper, FASE is tested regarding various aspects of its performance: It is compared to related algorithms, its statistical significance is tested, and its structural alignments are analysed through a comparison of the effectiveness of several scoring schemes for similarity. In addition to those results and experiments, methods for finding the so-called circular permutations (CPs) are described and subsequently used to find and present novel examples of proteins possibly related by such CPs. A CP is an evolutionary modification of a protein, in which its residues are circularly permuted. The overall 3D structure (architecture) often remains unaltered, with the only changes being alterations in the connectivity between some secondary structure elements (SSEs).

The FASE program has been compiled for several computer platforms, and made available for public use; further details regarding the practical usage of FASE and how to obtain it may be found in appendix B of the thesis. Structural alignment of proteins is the problem of finding maximal similarities between a pair of proteins — a similarity that can be verified visually by superposing their 3D structures. The resulting alignment of their amino acid sequences is hoped to reflect the evolutionary divergences that (if the pair did originate from a common ancestor) have caused their divergence. The widespread interest in structural alignment is motivated by the fact that evolutionary events may easier and more reliably be determined by structural rather than sequential analysis of proteins, since protein structure is more conserved than sequence during evolution. In addition to FASE, the algorithms developed for the PM research – to a lesser degree – are efficient and useful tools for PM. Specifically, I have implemented an artificial problem generator for PM, as well as the mentioned optimisation procedures for dealing with the problem.

To summarise, FASE is described in chapter 5 on structural comparison, and further details can be found in appendix B. The journal paper on FASE is presented in appendix F. The problem generator for PM is described in the PM paper, which may be found in appendix D.

Finally, an objective was to ***use bioinformatics tools to learn about the surrounding world***. In this thesis, the surrounding world is the nature at the molecular level, more precisely the protein level. This somewhat boldly sounding goal was pursued by applying the developed program FASE to finding CPs. This was done most noticeably in the second paper related to FASE, but was also dealt with in first paper on FASE. The second paper, *Detection and evaluation of novel CPs using FASE*, is included in appendix G. The CPs that were detected between proteins of known structure using FASE were evaluated and discussed. In evaluating these CPs, methodologies for making decisions concerning homology between proteins are discussed and presented. By documenting discovery of new and surprising similarities, a previously perhaps underestimated element of protein evolution, namely the CPs, is highlighted. Such insights of possibly new mechanisms, in combination with other findings, may change the way in which

protein evolution is regarded, and might (in the long run) help researchers to find answers to how proteins evolve and fold.

CPs are introduced in chapter 6, related concepts of protein evolution and homology discrimination are discussed as well, and the paper on detection and evaluation of novel CPs is presented in appendix G.

1.2. Chapter Outline

This thesis has been structured into two parts: *Part I* begins with a general introduction to the research made (this chapter). In the following five chapters, the necessary background knowledge is introduced.

- In chapter 2 on heuristic optimisation, problem solving in the context of optimisation problems is defined and discussed. Next, locality and a generic local search scheme are introduced, and a list on well-known search methods is derived from it. This is followed by a description of population based search methods. Specifically, EAs, PSO, and DE are introduced.

- Continuing in chapter 3, the topics of DNA, transcription and translation, proteins in general, and protein structure are introduced. This comprises the representation of structure at different levels, and division into domains that are suitable for further analysis and computation.

- In chapter 4 the PM problem is introduced. First, an introduction to different approaches for whole genome sequencing is given. Following this, PM is introduced and defined, and more practical and experimental issues — that affect the computational approach – are described. Then, scoring functions and heuristic optimisation methods for PM are presented and discussed. Finally, the main findings and research contributions made to PM are summarised.

- Structural comparison of proteins is introduced in chapter 5. This includes an exploration of the problem definition, a review of methods for structural comparison, as well as a thorough description of the program FASE for structural comparison. The chapter concludes with a description of some principles for significance assessment of similarity scores.

- Finally, in chapter 6 an overview of protein evolution is presented, a framework for distinguishing between homology and analogy is developed, and an introduction to CPs and their detection is given.

Part I ends with the conclusion (chapter 7) followed by the appendices. The first appendix, appendix A, defines basic, technical background concepts from computer science and computational geometry. The concepts NP-completeness

and NP-hardness that are used at various places in the thesis are defined, and the routine used for least squares superposition in FASE is presented as well. The second appendix, appendix B, includes an overview of FASE and practical aspects such as usage, availability, and parameter settings. The third and final appendix, appendix C, includes running time analysis of FASE and experiments with FASE that illustrate the running time of the method, and the effects on running time and solution quality of varying different parameters.

Part II consists of the three papers I have published during my Ph.D. study, and a fourth paper, which is currently in submission for a journal. As mentioned, one of the published papers has been accepted for publication in the *Journal of Computational Biology*, while the two other papers have been published in the *Proceedings of the Congress on Evolutionary Computation*, and presented at the conferences in Canberra, Australia, December 2003 and in Portland, USA, June 2004.

Chapter 2

Heuristic Optimisation

In this chapter, an introduction to heuristic optimisation will be given. We start by discussing some general issues of solving and modelling hard problems, thus introducing some trade-offs between model complexity and expected solution quality. This indirectly motivates the use of heuristics. Next, we look at some more specific issues of problem solving (representation, objective, and evaluation function), whereafter we formally define a search problem. Also, we define the concepts of locality and neighbourhoods in search problems, which serves as an introduction and prerequisite for the general description of local search that follows hereafter. This generic skeleton allows us to describe various variants of local search (hill-climbing, stochastic hill-climbing, simulated annealing) by specialisation of the parameters of the general skeleton. The (few) prerequisites for setting up local search are described, and the disadvantages of local search are highlighted as well. These problems are typical for the class of greedy algorithms that we also briefly survey.

The following section contains an introduction to population based search methods. In particular, we introduce evolutionary algorithms and the related derivable methods such as genetic algorithms and evolutionary strategies. Finally, we discuss the matter of general purpose heuristics vs. heuristics for specific problems.

2.1. Motivation for the Use of Heuristics

Among the infinite number of problems in this world, there is a subset of problem instances, for which the following properties are true:

- These problems have large search spaces; so large, in fact, that they cannot realistically be enumerated or searched exhaustively.
- There are no known methods for finding the best solution to these problems that do not employ a strategy that is fundamentally similar to exhaustive search.

For those problems it seems as a hopeless endeavour to try to find with certainty the optimal solution within reasonable time. The main obstacle is not the large search space in isolation; in fact, many problems with large search spaces can be solved quickly. Rather, the difficulty is caused by lack of structure in the problem and its search space, which makes them difficult to check efficiently. This is what makes the hard problems harder than the easy ones.

An excellent example of this difference emerges by comparing the Hamiltonian Path problem [29, page 925] with the Eulerian Path problem [29, page 496]. In the former problem one has to find a cycle in an undirected graph, while visiting each *node* exactly once. In the latter, one must find a cycle in an undirected graph, while visiting each *edge* exactly once. The formulations of the problems sound intriguingly similar, and furthermore their search spaces are of the same magnitude for most normal, non-contrived graphs. However, whereas the Eulerian Path problem is very simple to solve (a connected, directed graph has an Eulerian circuit if and only if the in-degree is equal to the out-degree for all nodes [29, page 496] — which is equivalent to stating that a connected, undirected graph has an Eulerian circuit if and only if it has no nodes of odd degree), surprisingly the Hamiltonian Path seems very hard (NP-complete [29, page 926]), and the best known way to decide if a graph has a Hamiltonian Path is basically to perform an exhaustive search.

The existence of such hard problems makes it relevant to use heuristics. As mentioned in the introduction, a *heuristic* is a problem solving method that does not guarantee to find the optimal solution, or alternatively does not guarantee a bounded running time – it may not even promise either one [166]. In another, but similar definition [124] it is stated that *"A heuristic is a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is"*, which essentially is to say the same thing except that "not guarantee a bounded runtime" is substituted with "at a reasonable computational cost".

Reflecting over the definition of a heuristic might lead to the hasty conclusion that the term heuristic merely defines a useless class of algorithms: No guarantees on quality nor time usage are given. The reality, however, is different: An approximate!solution (delivered after a slightly uncertain but reasonable amount of computation time), is far better than no solution at all (or one that will be delivered after an aeon of computation), which often is the alternative if no compromises are made. More precisely, the alternative to using heuristics is to employ an exact method; sadly, one must typically simplify the problem in order to obtain an "optimal" solution in reasonable time. For instance, in sequence alignment there are a number of requirements that the model (score function) must satisfy for dynamic programming (DP) to be applicable [37, chapter 1]. Thus, in such cases it is not a suboptimal solution to the problem formulation or an uncertain runtime that makes the approach an overall heuristic, but the fact

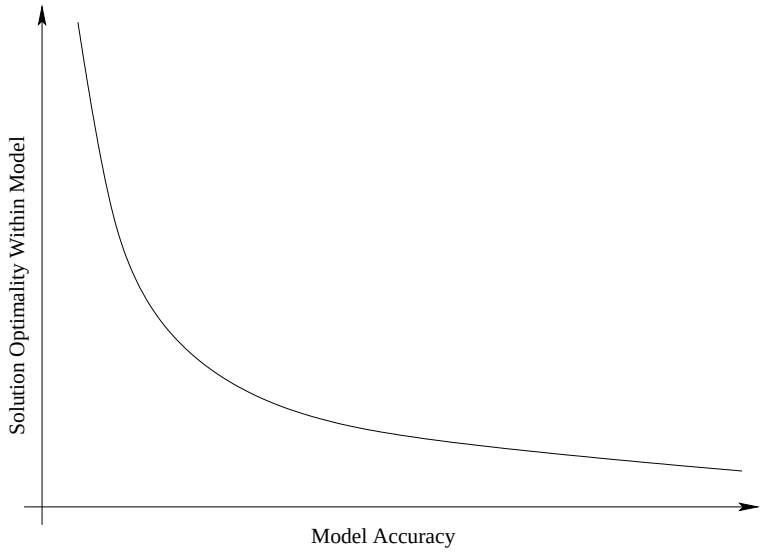


Figure 2.1.: Illustration of the trade-off that exists for many hard problems; given limited resources, the choice stands between model accuracy, x-axis, and solution optimality (achievable within the model), y-axis.

that the problem has been simplified and that, therefore, the "optimal" solution might not be optimal at all.

The above discussion highlights a central choice when solving hard problems: One can search for an optimal solution to an approximate!model of the problem, or one can search for an approximate solution to a precise model of the problem [101, page 18]. This somewhat simplified statement deserves some comments. Firstly, in principle one could do both things, but that leaves us bouncing back to the case of trying to solve a NP-hard problem [113] efficiently, which is neither realistic, nor advisable. Secondly, in reality, what is being faced is not a choice between two options. Rather, it is a continuum of choices, constituting a trade-off between the realism of the model and the ease of finding optimal solutions to the model. In the end, the usefulness of the solution to the *real* problem must be decisive for the choices made. This trade-off has been illustrated in figure 2.1. Somewhere on the trade-off curve lies an approach that achieves the best solutions to the actual problem. The FASE method (see section 5.6 and appendix F) is an example of a method, where extensive effort has been devoted to finding the right trade-off between creating an appropriately realistic model and the possibility to discover satisfyingly optimal solutions (within the model).

A different but related technique than that of creating a simplified model, is to assume that certain solutions to the model are unrealistic; this makes the effective search space smaller, and thus the search becomes faster. Many examples of this exist: One is the BLAST search [5] that assumes that interesting hits must start

with a certain number of consecutive matching residues (or nucleotides), and further do not contain gaps above a certain size. Another example is (again) the FASE method, which sets some assumptions on what can be expected from an interesting protein structure alignment and superposition; thus certain solutions – that are hoped and assumed to be uninteresting biologically if investigated – are excluded from the search to make it faster.

2.2. Problem Solving

When solving a problem, one has to choose a method – an algorithm. Regardless of this choice, some elements are always part of problem solving. This includes the choice of data structure for representing the problem, and it requires the definition of objectives and measures of quality. We discuss these issues below.

2.2.1. Representation

In principle, the problem representation is independent of the algorithm, but often there is a connection. Some algorithms require (or favour) certain representations, and some problems may only be represented in certain ways. Thus, the chosen representation may be biased towards the method used, and the problem may restrict the range of applicable methods.

In protein structure comparison, for instance, the problem can be represented in several ways, of which two are the most prevalent, namely the explicit coordinate description and distance matrices (see chapter 5). The two representations contain the same amount of information (except for a possible chirality loss for distance matrices, see section 3.3). However, they imply differences in the optimisation criteria and algorithmic methods that can most naturally be applied. Thus, in this particular problem domain, one has the freedom to choose the representation that best conforms to ones ideas for algorithmic approaches and optimisation criteria.

2.2.2. Objective and Measure of Quality

Distinction Between Objective and Measure of Quality

Every problem being solved must have an objective that a solution should fulfil. Similarly, for the computational formulation of the problem, there must be some measure of quality (also known as fitness function, evaluation function, score function, etc.) to guide the used computational method towards fulfilling the objective [101, chapter 2]. In particular for artificial benchmark problems, it can be hard to separate objective and score function (e.g. when optimising a continuous, real-valued function). For problems originating from real-world situations, the distinction is less subtle. For instance, in protein folding (see page

40) and in protein-ligand docking, the objectives are to predict the conformation of a protein structure and to predict the interactions between proteins and smaller compounds, respectively. The measure of quality, however, is in both cases typically an energy function to be minimised [103, 73].

Multiple Objectives

In some problems, the task is to optimise with respect to multiple objectives instead of just one. In this case, we are dealing with multi objective optimisation [33]. A classic example is the design of cars, where the task may be to maximise the performance of the engine, while minimising its weight and size. This trivial example illustrates that two or more objectives often are conflicting in multi objective optimisation. A more relevant example is structure alignment of proteins, where it is obviously interesting to find as large similarities between protein structures as possible, while simultaneously fitting the similarities as well as possible. These two goals are also conflicting. Thus structural alignment can naturally be formulated as a multimodal problem, but is often solved by optimising a single measure of quality that combines the objectives into one.

Correlation Between Objective and Measure of Quality

In structure comparison, it is very important that solutions with optimal values for the measure of quality also return biologically interesting alignments – solutions that fulfil the original objective. This might sound as a trivial consideration and an obvious requirement that is easily satisfied; it can however be rather complex to get these two concepts to correlate well. Using docking as an example again, the interaction with the lowest energy value is not necessarily the true conformation between protein and ligand, and obtaining an good correlation between objective and measure of quality is alpha and omega in docking [151]. We deal with this issue of correlation in chapter 5 and in the research paper *Flexible Secondary Structure Based Protein Structure Comparison Applied to the Detection of Circular Permutation* in appendix F; specifically we investigate the correlation between the objective of finding similarities between protein structures[1] and several measures for assessing the goodness of the structure alignment. The best measure (subsequently used in FASE) turned out to be a robust consensus score, which combined the five tested measures.

2.2.3. Definition of an Optimisation Problem

Definition 2.1 (An optimisation problem) *Let $\mathcal{S}$ be a search space and $\mathcal{F} \subseteq \mathcal{S}$ be the feasible part and f be a fitness function[2]. An optimisation problem is to*

[1] The objective was quantified using the SCOP classification of proteins as a "gold" standard.
[2] In this definition, *fitness function* is similar to the previously used *measure of quality*

find an $x \in \mathcal{F}$ *such that*

$$f(x) \leq f(y) \qquad \textit{for minimisation problems} \tag{2.1}$$
$$f(x) \geq f(y) \qquad \textit{for maximisation problems} \tag{2.2}$$

for every $y \in \mathcal{F}$*. The solution,* x*, is called the* global optimum*. The fitness function* f *is either* numerical *(*$f : \mathcal{S} \rightarrow \mathbb{R}$*) or* ordinal *(*$f : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$*). If* f *is numerical, this definition describes a* numerical optimisation problem [101, page 39]*.*

2.3. Locality

In this section we will introduce, define, and discuss the concept of *locality*. For this, we will first define a *neighbourhood*. Locality will be useful when discussing local search methods later in this chapter, and when reflecting about search methods in general.

Below we give two definitions of a neighbourhood. They complement each other: In the case where it is inconvenient to define a distance measure on the search space, the second definition is "handy"[3].

Definition 2.2 (Neighbourhood I) *Define a distance measure between two solutions:* $dist : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$*. Then, for all* $x \in \mathcal{S}$*, the neighbourhood of* x*,* $N(x)$*, is defined as*

$$N(x) = \{y \in \mathcal{S} | dist(x, y) \leq \epsilon\}$$

where ϵ *is a real number larger than 0.*

Definition 2.3 (Neighbourhood II) *Define a neighbourhood function,* m*, from the search space to the set of subsets of the search space:* $m : \mathcal{S} \rightarrow 2^{\mathcal{S}}$*. Then, for all* $x \in \mathcal{S}$*, the neighbourhood of* x*,* $N(x)$*, is defined as*

$$N(x) = m(x).$$

Definition 2.4 (Local minimum) *An element* $x \in \mathcal{F}$ *is a local minimum if* $f(x) \leq f(y)$ *for all* $y \in N(x)$ *and a local maximum if* $f(x) \geq f(y)$ *for all* $y \in N(x)$*. A local optimum is a local minimum or a local maximum.*

2.3.1. Relation Between Locality and Research Topics

In the following section we will spend some time and space discussing methods and concepts of local search (LS). The main reason is that these issues are central to much of my research: In the physical mapping (PM) research (chapter D), and

[3]The neighbourhood definitions are inspired by [101, pages 36–43]

in the extensive investigation of the different optimisation methods (chapter E), LS based methods was used to assess the performance of other and more complex methods (namely simulated annealing, evolutionary algorithms, and differential evolution). The greedy LS methods also helped to characterise the search spaces that were investigated; e.g. in the PM paper (chapter D), the correlation between neighbouring solutions in the search space was investigated by use of LS.

In the design of the program FASE for structural comparison of proteins, there are a number of LS based elements: The construction of alignments in the first phase is greedy and iterative (thus resembling LS). The search for alignments in the second phase (after the rough alignments have been made) is also local search. For further details and for an introduction to FASE, we refer to section 5.6.

2.4. A Generic Local Search Approach

In this section we present a generic description of the local search (LS) algorithm. The purpose of doing so is to highlight the similarities between many existing single solution optimisation methods such as hill-climbing (HC), stochastic hill-climbing (SHC), simulated annealing (SA), etc. In fact, they are all special cases of the general method, and they can be obtained by fixing parameters or restricting some choices that describe the more general method. For more background on LS, we refer to [101, section 2.6, 3.2, and chapter 5] and [112].

The LS method requires only one more aspect than those always required – namely representation and measure of quality – in general when solving a search problem. This extra aspect is the definition of a neighbourhood. Once the neighbourhood has been defined, all of the following local search algorithms should be applicable.

The generic local search method is given in figure 2.2. The following aspects of it can be parameterised:

- Number of restarts, `NrRestarts`.
- Number of iterations per temperature, `LocalTempIterations`.
- The way in which the start point is selected in `SelectStartPoint()`.
- The way in which the temperature is initialised, updated, and used as termination criterion, implemented through `TempStart`, `DecreaseTemp()`, and `TempEnd`.
- The principle for accepting the next current solution, implemented through the `SelectNew()` function. The straightforward approach is to select the best of the new and current solutions, `n` and `c`; the other extreme is random

```
Method Generic Local Search
    N = NrRestarts
    while (N > 0)
        T = TempStart
        c = SelectStartPoint()
        while (T ≥ TempEnd)
            i = LocalTempIterations
            while (i > 0 and not LocalOptimum(c))
                n = SearchNeighbourhood(c)
                c = SelectNew(c,n)
                i = i − 1
            T = DecreaseTemp(T)
        N = N − 1
    return BestFound
```

Figure 2.2.: The generic local search algorithm

selection. In between the two lies probabilistic, biased selection as used in SA.

- The function `SearchNeighbourhood()` gives rise to two parameters: The first is the degree to which the local neighbourhood is searched for improvements. Here, one extreme is to check only one solution from the neighbourhood, while the other is to check them all. The range in between is occupied by strategies checking some percentage of solutions. The second parameter of `SearchNeighbourhood()` is its selection of a return value, which must be chosen amongst the searched values from the neighbourhood. Often this is simply the best solution found; however, it might also be chosen at random, or biased more or less towards solutions with good scores.

The function `LocalOptimum()` returns true if all solutions in the neighbourhood was checked in the last iteration, and none of them were better than the current solution.

2.4.1. Derivable Algorithms

Let us review some of the algorithms that can be derived from the generic LS algorithm. For all the algorithms, the selection of the start point (implemented in `SelectStartPoint()`) can be random, based on the output of another method, or biased in some other way.

Basic Hill-Climbing

Hill-climbing (HC) is typically not attributed to any single individual researcher. The first approaches using HC strategies emerged during the 1950s and 1960s [85]. Initially, HC was mostly applied to discrete combinatorial problems such as the travelling salesman problem (TSP), but today HC is being applied to continuous optimisation problems as well.

In basic HC, one is just interested in executing the inner loop in figure 2.2 (where iteration over `i` takes place). Thus, we set `NrRestarts = TempStart = TempEnd = 1`, and use some arbitrary decreasing function in `DecreaseTemp()`. This causes the two outer loops to be executed only once. `SearchNeighbourhood()` should inspect the whole neighbourhood and select the best solution from it, just as `SelectNew()` also must select the best of its two arguments. In pure HC, we do not stop until a local optimum has been found; to achieve this we set `LocalTempIterations` "infinitely" high, such that the inner loop stops only when the `LocalOptimum()` function returns true; that is, when a local optimum has been found.

Iterated Hill-Climbing

Quite simply, to produce iterated HC (IHC), the only change needed in comparison to HC, is to change the value of `NrRestarts` to a value above 1. This will cause the HC method to restart each time a local optimum has been found, and the result is IHC.

Stochastic Hill-Climbing

Stochastic HC (SHC) — also known as the Metropolis algorithm — was introduced by Metropolis [98]. It is a modification of the HC scheme, where the most significant difference is that SHC may move to a worse solution than the current one. This is not beneficial for very simple problems (e.g. a problem with only one optimum, i.e. a unimodal problem), but is a great improvement for more difficult problems, on which basic HC may get stuck on local optima — a phenomenon known as premature convergence. Below, the SHC will be described in the context of the generic LS scheme presented in figure 2.2.

SHC does not search the whole neighbourhood. On the contrary, it checks only a single, random solution. The next current solution is selected by probabilistically selecting between the current and the new candidate in the `SelectNew()` function, using the same scheme as in SA (see below).

We can set `NrRestarts = 1`. The fixed temperature and the SA selection function determines the degree to which good solutions are preferred relative to bad ones. In SHC the temperature is constant. Thus, we should set `TempStart = TempEnd`, and use some arbitrary decreasing scheme in `DecreaseTemp()` (to

achieve just one iteration over `T`). The above implies that the parameter `LocalTempIterations` determines the overall number of iterations of the algorithm. `LocalOptimum()` will never return false, because only one point is searched in the neighbourhood, and hence we can never be sure to detect a local optimum.

Simulated Annealing

SA was introduced by Kirkpatrick *et al.* in 1983 [81]. SA is an extension of SHC.

The difference between SHC and SA is that the temperature is annealed instead of being constant during the optimisation process. Thus, the number of iterations is not only determined by `LocalTempIterations`, but also by `TempStart`, `TempEnd`, and `DecreaseTemp()`.

In its most basic form, we might set `NrRestarts = LocalTempIterations = 1`, and choose settings for `TempStart`, `TempEnd`, and `DecreaseTemp()` that anneals the temperature as wished, and gives the desired number of overall iterations. Typically, `DecreaseTemp()` multiplies `T` with a factor less than one to decrease the temperature from `TempStart` to `TempEnd`, eventually causing termination. In this lies the main idea of SA: As iterations pass, less good solutions will with decreasing probability be accepted relative to better ones due to the nature of the selection scheme (see below).

SA may be extended by using `LocalTempIterations` to get more iterations for each temperature (instead of just one). This helps to balance the number of evaluations made against the annealing scheme. Further, we may increase `NrRestarts`, to get several totally independent runs.

The probabilistic selection scheme used in SA and in the SHC is based on the following probability value p (for minimisation problems!) for selecting the new solution `n`:

$$p = \frac{1}{1 + e^{(\frac{Fc - Fn}{T})}}$$

where Fc are Fn are the fitness values of `c` and `n`, respectively [101, page 120]. This policy is carried out by the `SelectNew()` function. As a final remark, in most variants of simulated annealing, if $Fn < Fc$, solution `n` is always returned.

Random Search

Random search (RS) comes, perhaps surprisingly, in several variants. All of them should, however, have temperature parameters set as in HC.

In a fully RS, where the next solution is not dependent on the fitness of the previous one, the `SelectNew()` should always select the new solution regardless of its fitness. `SearchNeighbourhood()` checks and returns one random solution from the neighbourhood. If the neighbourhood is equal to the search space, we have a total uniform sampling from the search space in each iteration. If the neighbourhood is smaller, the search becomes a kind of random walk.

Classic Local Search

As for RS (above) and exhaustive search (below), classic local search (CLS) is a relatively simple method, and is therefore not attributed to any single reference. Of course, many interesting extensions have been proposed over the years, but describing these would be outside the scope of this chapter. Instead, for a good introduction to CLS and many aspects of it, the following reference can be recommended [112].

Using the above settings from RS, if the smaller neighbourhood is combined with `SelectNew()` always selecting the best solution, the result becomes CLS. The following CLS algorithm emerges: 1) Select some random point `c` 2) Find a new, random point `n` from the neighbourhood of `c` 3) If `n` is better than `c`, set `c = n` 4) Goto 2 or stop if `i` iterations have been performed, or another termination criterion has been fulfilled.

Exhaustive Search

Exhaustive search (EXS) is a quite degenerate version of the generic LS algorithm, where all three main loops are only executed once, where the neighbourhood is equal to the whole search space, where `SearchNeighbourhood()` searches the whole neighbourhood, and where `SelectNew()` selects the best solution. Of course it is mostly a theoretical observation (and of less practical relevance) that EXS can be constructed from the general LS scheme, since EXS is too time consuming to be relevant for many interesting problems.

2.4.2. Exemplification of Local Search Algorithms

Figure 2.3 on page 18 illustrates the application of six of the different LS algorithms from section 2.4.1 on an optimisation problem (see below). The left side graphs of figure 2.3 depict (in the search landscape — the response surface) the sample points chosen by each algorithm. Lines are drawn between consecutive samples. A *sample* is the input value given by the algorithm to a call of the evaluation function. The sampled points are also depicted via an orthogonal projection onto the plane that is defined by the equation $z = 7$. On the right side graphs, the fitness of the samples are shown along the y-axis, with samples ordered along the x-axis.

The six algorithms were described in section 2.4.1; they are: 1) RS, 2) IHC, 3) Iterated CLS (ICLS), 4) SHC, 5) SA, and 6) EXS.

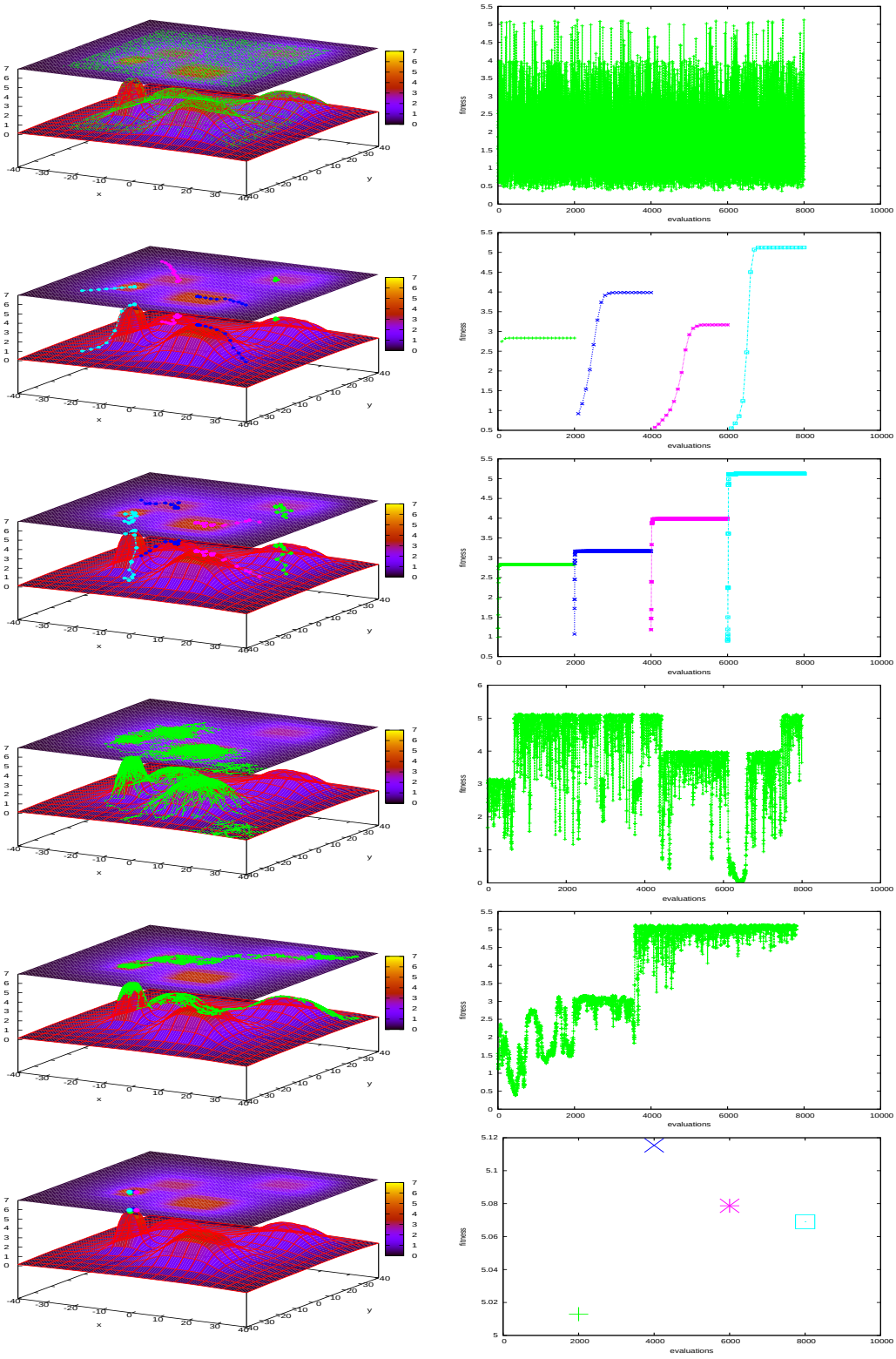


Figure 2.3.: See section 2.4.2 on page 17 for details.

The Optimisation Problem

The search space, $\mathcal{S}$, is a subset of $\mathbb{R}^2$ and is spanned by the rectangle $\mathcal{S} = ((-30, 30), (-30, 30))$. The fitness function, $g(x, y)$, is defined as:

$$\begin{aligned} g(x,y) = {} & f(x,y,20,20,10,2) + f(x,y,-10,3,8,2) + \\ & f(x,y,10,-17,10,3) + f(x,y,-16,-13,4,4) + f(x,y,0,0,30,1) \end{aligned}$$

$$\text{where } f(x,y,a,b,c,d) = d\left(\left(\frac{x-a}{c}\right)^4 + \left(\frac{y-b}{c}\right)^4 + 1\right)^{-1}$$

Thus, for the function f, the optimum is at (a, b), c is the width of the peak, on which the optimum lies, and d is the value of the optimum — the height of the peak. The optimisation problem is a sum of four inverted parabolic surfaces, the f function, and has (see figure 2.3) three local optima plus one global optimum (which, in principle, also is a local optimum). The four optima are in approximate values:

$$\begin{aligned} g(17.1145, 17.0669) &= 2.8343, \\ g(8.1568, -14.4785) &= 3.9848, \\ g(-7.8364, 0.7931) &= 3.1670, \text{ and} \\ g(-15.2356, -12.1192) &= 5.1233. \end{aligned}$$

Algorithm Performance

The parameter settings of the six variations of the generic LS scheme are given in figure 2.4. The neighbourhood N differs for the algorithms, and is also specified in figure 2.4. For all the algorithms the initial point is picked randomly and uniformly from $\mathcal{S}$. Each algorithm is now examined in turn; this order corresponds to the order of the graphs in figure 2.3, starting from the top.

RS) RS simply picks 8000 random points from $\mathcal{S}$. Although `SelectNew()` in the general version of RS always should select the new solution, `n`, it does not matter in this implementation, because $N = \mathcal{S}$, and therefore the output of `SearchNeighbourhood(c)` is independent of the current solution, `c`. RS samples points across the search space and also near the highest peak, but lacks the ability to fine-tune its solutions.

IHC) In all restarts, IHC moves nicely and smoothly along the gradients of the fitness landscape, because the neighbourhood, given by the unit square (us), is sampled 100 times; this setting simulates exhaustive search of the neighbourhood. IHC finds all four peaks.

ICLS) The graphs for the ICLS are similar to IHC, but there are some important and subtle differences: ICLS does not proceed along the gradient in a straight ascent towards the top; rather, it moves in a zig-zag movement towards

	RS	IHC	ICLS	SHC	SA	EXS
`NrRestarts`	1	4	4	1	1	4
`LocalTempIterations`	8000	20	2000	8000	300	1
`SearchNeighbourhood()`	1	100	1	1	1	2000
`TempStart`	-	-	-	0.6	1.5	-
`TempEnd`	-	-	-	0.6	0.1	-
`DecreaseTemp()`	-	-	-	-	$\times$ 0.9	-
N	$\mathcal{S}$	us	4·us	4·us	us	30·us
`SelectNew()`	new	best	best	SA	SA	best

Figure 2.4.: Parameter settings and parametrisation choices for the six algorithms discussed in section 2.4.2 and used in figure 2.3.

the top, while finding better solutions. The distinction between IHC and ICLS is the two types of LS known as steepest ascent and first-improvement, respectively (see section 2.4.3). Also ICLS finds all four peaks.

SHC) SHC visits — in a stochastic manner — the high fit regions of the search space. It is basically a random walk, but with a bias (determined by the temperature) towards selecting the better solution of the current and new one. As it can be seen from the graph on the right, it visits the highest region on two occasions, but makes exploratory walks during the whole process.

SA) SA is as mentioned very much like SHC, but the likelihood of escaping optima decreases over time. This enables more exploration in the initial phase, and more fine-tuning in the terminal phase of the optimisation, as it can be observed on the left-side graph in the figure, where the path of the SA becomes much more focused — and less stochastic. This is also reflected on the graph on the right, where the exploration gets less and less pronounced. The algorithm finds the optimal peak and stays there for the rest of the run.

EXS) In each restart, EXS picks an initial starting point from $\mathcal{S}$, and from this point 2000 improvements are sought in the neighbourhood of the initial solution (the neighbourhood is spanned by ((-30,30)(-30,30))). EXS is, in fact, very similar to ICLS, it just has a much larger neighbourhood (see table 2.4). Moreover, because of the large neighbourhood, the new sought solutions are essential independent of the current one, and thus EXS also ends up resembling a RS. This makes fine sense, because the "exhaustive" behaviour, as it is clear from above, is implemented (or simulated) through a high number of random samples from the neighbourhood. All four restarts find a solution near the optimal peak.

Discussion of the Behaviour of the Algorithms

On this simple, low dimensional problem, six LS algorithms were tested. They all found the optimum region, although they were capable (or incapable) of fine-

tuning to varying degrees; fine-tuning requires a decreasing neighbourhood size over time (and was not part of the simple generic LS model presented in section 2.4). Rather, the point was to illustrate the different ways in which the LS algorithms operate in finding solutions to a problem.

For instance it could observed that IHC appears more stable and "deterministic" that ICLS, which when starting from a valley may select either one of two peaks, whereas IHC will consistently select the peak with the steepest initial ascent.

It was illustrated how the most stochastic methods, SH and SA, operate between the extremes RS/EXS and IHC/ICLS, and how they differ in levels of stochasticity. SA exhibits this variation over time, as it (in the extremes) can go from RS to HC during a run through a variation of the temperature from ∞ down to 0 (strictly speaking: $\epsilon > 0$).

Finally, it was seen how RS and EXS were very similar, and actually appeared very non-greedy and exploratory due to the uniform sampling of the search space. This observation leads to the idea of combining RS with subsequent HC or LS starting from the best randomly found points; such approaches are also known as hybrid or memetic algorithms (see e.g. [101, chapter 14]).

2.4.3. A Discussion of Greedy Local Search Algorithms

A greedy algorithm is an algorithm that makes choices based on what looks best locally in time and/or space. According to some definitions [101, page 87], greediness only applies to the process of building up a solution from partial solutions. Other definitions of the concept, however, acknowledge that an algorithm might also be greedy simply because it moves around in a search space based solely on promising local information [17].

Below we will discuss a greedy algorithm — namely the local search (LS) algorithm introduced in several variations above.

Advantages and Disadvantages

In general, the greedy LS methods have some obvious advantages: They are conceptually simple and also simple to implement: In addition to what is needed when solving *any* problem, namely a representation and a measure of quality, only a neighbourhood definition is required to make LS applicable.

However, as all heuristics they make no guarantees on computation time nor quality of the found solutions. Furthermore, they are dependent on the initially chosen solution, and consequently many restarts are sometimes needed (as it is evident from the `NrRestarts` parameter in the generic LS algorithm).

The more greedy an algorithm is, the more prone it is to converging to a local optimum instead of a global one (premature convergence) and the faster it converges in general. Of course, this can in principle be solved by enlarging the

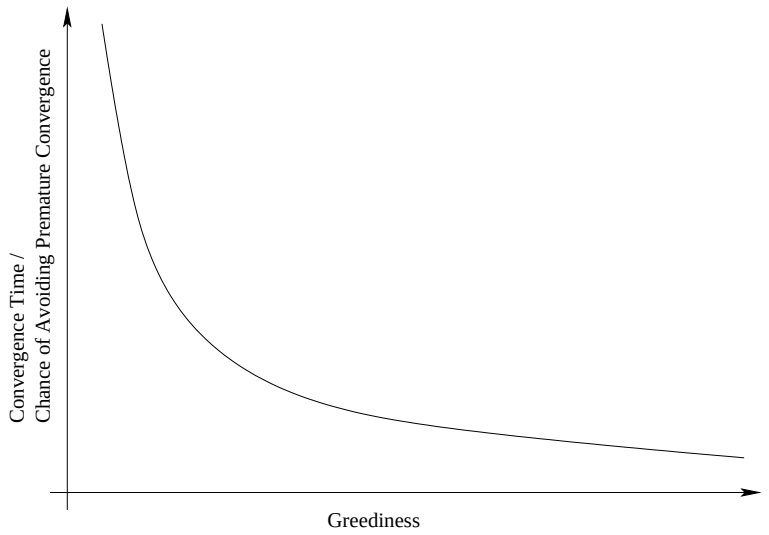


Figure 2.5.: Illustration of the trade-off for greedy LS algorithms: The more greedy, the faster is the convergence, but the lower is the chance of avoiding premature convergence.

neighbourhood, but if a large neighbourhood is to be searched well, it becomes unfeasibly costly. There are no easy solutions. The "secret" is to find the right degree of greediness – or to trade off the two concepts exploration (global search) and exploitation (local search) [101, page 44]. We have illustrated these two issues below in figure 2.5.

For a discussion of the reason why it is impossible (in general) to be very greedy and achieve fast convergence while avoiding premature convergence, we refer to section 2.6.

Means for Decreasing Greediness

We have just been discussing LS and the greedy idea of the methods based upon it. In summing up, it has become clear that the degree of LS can be modified by two approaches:

- Do not always use the best solution in the neighbourhood, but make randomised choice. The two extremes are also known as the *steepest ascent* (use the best solution in the neighbourhood) and *first-improvement* (use a random solution from the neighbourhood) strategies, respectively [112].

- One can increase the neighbourhood size – this makes search more global, less greedy, and more random. The *steepest ascent* and large neighbourhood becomes a global search (but a potentially very slow one), while *first-*

improvement and large neighbourhoods results in increasingly randomised search method.

2.5. Population Based Search Methods

A population based search method (PBSM) is a method that maintains a population of solutions to a problem. This is in contrast to the methods described in this chapter so far, where only one solution was maintained. When defining a PBSM it is difficult to be more specific than stating that they are methods where a population of solutions is maintained and updated in a series of iterations by one or more operators. To formalise and define this, we may write:

$$P(t+1) = T(P(t))$$

where $P(t)$ is the population at time t, and T is a transformation function, $T : \mathcal{P} \rightarrow \mathcal{P}$, that takes one population as input and outputs yet another population [8]. We may observe that a population belongs to the set $\mathcal{F}^{|P|}$, where $\mathcal{F}$ (as defined earlier) is the feasible region of the search space, and $|P|$ is the population size. According to the definition above, the only difference between any two PBSMs, per definition, is the transformation function T.

The idea of maintaining a population might seem quite new, if compared to the greedy LS methods discussed previously in this chapter. However, it is not so surprising as it might seem: The PBSM is the consequence of the occasional convergence to local optima for single solution methods. As a means to deal with this, in the generic LS, the `NrRestarts` parameter was introduced, in order to be able to make several independent runs of the method.

PBSMs might thus be seen as parallelised local searches with exchange of information between the search processes during the runs. Indeed, this idea is well-known, for instance as parallel SA [54, 15], and the similarity between parallel SA and evolutionary algorithms (EAs) is obvious, and has been recognised [15]. The assumption behind (and thus motivation for using) most PBSMs (in particular EAs) is that this parallel execution with continuous information exchange is more efficient than a similar number of serial and independent executions of local searches, because the population provides multiple simultaneous solutions to a problem and thus better possibilities for efficient search (see e.g. [152, page 20]).

2.5.1. Evolutionary Algorithms

EAs is a class of iterative PBSMs that are inspired by the natural evolution of the species. The origin (to use a central term from Darwin's theory about natural evolution) of EAs as a computational method can be traced back to the 1950s, where researchers started to experiment with stochastic, PBSMs; a development

```
Method Evolutionary Algorithm
    t = 0
    Initialise(P(t))
    Evaluate(P(t))
    while (not done)
        t = t + 1
        SelectNewFrom(P(t−1))
        Transform(P(t))
        Evaluate(P(t))
    return best
```

Figure 2.6.: The evolutionary algorithm

that was made possible by the advent of computers. Several people independently worked on the same ideas around the same time. Some of this original research is regarded as the foundation of the current branches in evolutionary computation (EC) (e.g. Schwefel: Evolution Strategies [130] and Fogel: Evolutionary Programming [45]); much has happened since then, and most current methods are results of a continuous mix of original ideas, and they do therefore not resemble the original ideas much. For a review of the origins of EAs and EC, we refer to [44].

The three most significant concepts borrowed from evolutionary theory are *mutation*, *recombination*, and *selection* (survival of the fittest). Like creatures in nature, solutions to a problem in an EA are subjected to random mutation, they are designed to mate by recombination, they continue to live if they are fit enough, and they die if they are not fit enough. The idea is to "simulate the evolutionary process of competition and selection and let the candidate solutions in our population fight for room in future generations" [101, page 140]. However, the biological realism of this simulation is not – and is not believed to be, either – particularly high. As stated later in the same source: "The algorithms that we will explore are just that – algorithms – mathematical procedures based on analogies to natural evolutionary processes" [101, page 141].

Continuing in more mathematical terms, if we denote selection by S, recombination by R, and mutation by M, we can express the process of an EA by the following equation (analogous to the general definition of PBSMs):

$$P(t+1) = S(M(R(P(t))))$$

where P and t are defined as in the previous equation, and R, M, and S are all functions of the form $\mathcal{P} \to \mathcal{P}$. The pseudo-code for the EA is given in figure 2.6.

The differences between various branches of EAs are caused by differences in representation of solutions, the specific implementation of R, M, and S, and the order of their execution. Also the termination criteria can vary, although this

aspect is not a distinguishing factor for any of the variants.

A big advantage of EAs is that they can operate on many types of objects; among the known examples are real-values, integers, matrices, graphs, artificial neural networks, and finite state machines [152, page 16].

Often these objects are simply represented in the most convenient way. However, some types of EAs require the objects to encoded using binary strings, e.g. the classic genetic algorithm (GA) uses this approach [50].

Operators – Mutation, Crossover, and Selection

In this subsection we review, very briefly, the most common operators of EAs, and some common views of their purpose; this is done to give the reader an overview of the basic components of an EA. In the two first papers made during my Ph.D. project (see appendices D and E), variations of a standard EA were used; an overview of basic EA operators is therefore given in this section. These issues can, however, be discussed in much greater detail, and a vast body of variations of operators, modifications, and likely improvements have been suggested in the history of EAs. However, such depth regarding these topics is beyond the scope of the thesis, and instead we refer – for a more elaborate treatment – to [11] and [100].

The ***mutation*** operator takes one solution and modifies it (typically) to some new solution in the neighbourhood. Using the Darwinist analogy, it is the purpose of mutation to introduce new genetic material [152] (whereas crossover just mixes available material) – a notion that makes most sense for discrete representations such as the bit-string encoding (see below). When operating on numerical domains, the operator is responsible for exploration of new areas in the beginning of the search and for fine-tuning the parameters in the end of the process. To achieve this effect, the magnitude of mutation is often annealed over time.

An efficient mutation operator for real-valued numerical optimisation problems is the annealed Gaussian mutation, which adds a number from the $N(0, \sigma^2)$ distribution to each solution coordinate with a given probability. The annealing is implemented by defining σ as a decreasing function of time; standard choices are either

$$\sigma^2(t) = \frac{1}{1+t} \quad \text{or} \quad \sigma^2(t) = \frac{1}{\sqrt{1+t}}.$$

Alternatively, the Cauchy distribution can be used; it has been shown to be efficient in several studies [173, 151]. We refer to the subsection on the *Simple Real-Valued EA* later in this section 2.5.1 for a definition of the Cauchy mutation operator.

When a bit-string representation is used, the same considerations do not apply, because the smallest mutation (one bit flip) does not always correspond to a minimal change in the decoded value. Still, the concept of reducing the overall amount of mutation as the process progresses, is common to nearly all EA

variants, because fine-tuning would be very hard to ensure otherwise.

Recombination (or crossover) operators come in variety of forms, and it would be outside the scope of this thesis to give a full description. The design is heavily dependent of the representation and the problem structure. Usually, problem specific knowledge is incorporated into the EA through the recombination operator.

The formal requirement for an operator to be a recombination operator is that it takes two or more solutions, and combines them to produce one or more new solutions. The crossover operator actually implements the population idea of the EA, because without it, the individuals would be independent and could have been evolved sequentially – as e.g. 50 local searches. Thus, the implicit assumption and purpose of crossover is that the newly combined solutions can overall help the EA to find better solutions [152].

The ***selection*** operator implements the concept of "survival of the fittest" in EAs, a concept that is intended to drive the population of solutions towards highly fit (near-optimal) solution areas. A more biological interpretation of this operator would be to state that it forces the population to adapt to the environment (the fitness function) over time.

The selection operator simply selects a set of solutions from the population; however, this selection can be done both with and without replacement. The most well-known and used schemes are the roulette-wheel selection (in particular connected to the GA), the ranking based selection from evolution strategies (ES), and tournament selection, which was introduced in evolutionary programming (EP). Descriptions of GA, ES, and EP, as well as descriptions of the selection schemes follow below.

Genetic Algorithms

The classic GA [50, 65] uses a fixed length bit string to encode solutions. The standard operators are bit-flip mutation, one-point crossover, and roulette-wheel selection [50, 101].

In bit-flip mutation, each bit is changed with the probability p_m. In one-point crossover, two solutions (parents) and a common random crossover-index in the bit-strings is selected. Then, two new solutions (children) are created by combining first and last parts of the two parents. In roulette-wheel selection, the new population is filled up by iteratively selecting solutions from the old population (one solution may be selected several times). At each selection, the probability p_i that solution i will be selected is equal to its fraction of the total fitness: $p_i = f_i / \sum_{j=1}^{|P|} f(j)$.

Evolution Strategies

The first versions of ES [122] were similar to LS; they had a population of only one solution, and the only operator present was the mutation operator. However, ES matured and were developed over the years. The generalised versions are now called $(\mu+\lambda)$-ES and (μ,λ)-ES, where the differences in naming reflects two possible selection schemes. In both schemes λ new solution are created. However, in the $(\mu+\lambda)$-ES, the solutions of the new population are selected from a total of $\mu+\lambda$ old as well as new solutions, whereas the new population in the (μ,λ)-ES is selected from the λ new solutions only.

Mutation in ES is implemented by the Gaussian distribution. Thus, a solution x is updated as $x(t+1) = x(t)+N(0,\sigma)$, where $N(0,\sigma)$ is a vector of independent Gaussian numbers, and x is a vector of the same dimensionality, and σ is a vector encoding the magnitude of variation for each dimension [101, page 160]. Control of this magnitude was done by deterministic means (the 1/5 rule). Also, a self-adaptive scheme was introduced (in fact, ES was the first of the EA variants to so). Here, the σ vector was part of the solution, thus being subjected to the optimisation process as well.

Although the initial versions of ES did not include crossover, most contemporary implementation use it. Two of the most used variants are uniform crossover – where each coordinate of the offspring is selected at random between two parents – and arithmetic crossover – where the weighted average of two parents generate one offspring.

Evolutionary Programming

EP was first described in 1962 by Fogel *et al.* [45]. EP and ES are considered to have been discovered independently of each other, but share many similarities. Having described ES above, we shall restrict ourselves to describe the differences between ES and EP below.

As mentioned, most ES versions have used crossover to discover new solution areas, whereas EP seldom has been seen implemented with this operator; instead, this task of discovery has in EP been left to the mutation operator. There are also differences in the selection scheme used: EP typically uses stochastic tournament selection, whereas ES uses the deterministic (μ,λ) or $(\mu+\lambda)$ selection strategies described above [152]. In tournament selection, a number of matches are held between two or more solutions from the population. The winner of each match is transferred to the next population. Matches may be decided purely based on fitness or may be partially stochastic (akin to the SA selection scheme). As in roulette-wheel selection, one solution can be selected more than once for the next population, because competitors for the matches are selected at random.

Simple Real-Valued Evolutionary Algorithm

In the comparative study of optimisation algorithms (see appendix E) we used a simple EA (SEA) that was previously found to work well on real-world problems [151]. The SEA works as follows: First, all individuals are randomly initialised and evaluated according to a given fitness function. Afterwards, the following process will be executed as long as the termination condition is not fulfilled (e.g. the current number of fitness evaluations performed is below the maximum number of evaluations allowed). Each individual has a probability of being exposed to either mutation or recombination (or both). Mutation and recombination operators are applied with probability p_m and p_c, respectively. The mutation and recombination operators used are Cauchy mutation using an annealing scheme and arithmetic crossover, respectively. Further, individuals that were altered due to mutation or recombination are re-evaluated using the fitness function. Finally, tournament selection [11, section 2.3] is used.

The *Cauchy mutation* operator is similar to the well-known Gaussian mutation operator, but the Cauchy distribution has thick tails that enable it to generate considerable changes more frequently than the Gaussian distribution. The Cauchy distribution has the form:

$$C(x, \alpha, \beta) = \frac{\beta}{\pi\beta^2 + (x - \alpha)^2}$$

where $\alpha \geq 0, \beta > 0, -\infty < x < \infty$ (α and β are parameters that affect the mean and spread of the distribution). All of the solution parameters are subject to mutation and the variance is scaled with 0.1 times the range of the specific parameter in question.

Moreover, an annealing scheme was applied to decrease the value of β as a function of the elapsed number of generations t, and α was fixed to 0. In the comparison study we used the following annealing function:

$$\beta(t) = \frac{1}{1 + t}$$

2.5.2. Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) was introduced by Kennedy and Eberhart in 1995 [80]. It was inspired by the swarming behaviour as it is displayed by e.g. a flock of birds, a school of fish, or even human social behaviour being influenced by other individuals.

Basically, PSO consists of a swarm of particles moving in an n-dimensional, real-valued search space of possible problem solutions. Every particle has a position vector $\vec{x}$ encoding a candidate solution to the problem and a velocity vector $\vec{v}$. Moreover, each particle contains a small memory that stores its own best position seen so far $\vec{p}$ and a global best position $\vec{g}$ obtained through communication

with its fellow neighbour particles. In the study in appendix E, we used the fully connected network topology for passing on information (see [157] for more details).

Intuitively, in PSO, information about good solutions spreads through the swarm, and thus the particles tend to move to good areas in the search space. At each time step t, the velocity is updated and the particle is moved to a new position. This new position is calculated as the sum of the previous position and the new velocity:

$$\vec{x}(t+1) = \vec{x}(t) + \vec{v}(t+1)$$

The update of the velocity from the previous velocity to the new velocity is determined by the following equation:

$$\vec{v}(t+1) = \omega \cdot \vec{v}(t) + U(0,\phi_1)(\vec{p}(t) - \vec{x}(t)) + U(0,\phi_2)(\vec{g}(t) - \vec{x}(t)),$$

where $U(a,b)$ is a uniformly distributed number between a and b. The parameter ω is called the inertia weight [133] and controls the magnitude of the old velocity $\vec{v}(t)$ in the calculation of the new velocity, whereas ϕ_1 and ϕ_2 determine the significance of $\vec{p}(t)$ and $\vec{g}(t)$, respectively. Furthermore, at any time step of the algorithm v_i is constrained by the parameter v_{max}.

The swarm in PSO is initialised by assigning each particle to a uniformly and randomly chosen position in the search space. Velocities are initialised randomly in the range $[-v_{max}, v_{max}]$. When a particle exceeds the search space boundary, it is repositioned at the boundary and its velocity is set to zero. It will travel into the search space already in the next time step because of the attraction to $\vec{p}$ and $\vec{g}$.

2.5.3. Differential Evolution

The differential evolution (DE) algorithm was introduced by Storn and Price in 1995 [140] and in several subsequent publications with modifications and additional offspring creation schemes [141, 139, 120]. Basically, it resembles the structure of an EA, but differs from traditional EAs in its generation of new candidate solutions and by its use of a greedy selection scheme. DE works as follows: First, all individuals are randomly initialised and evaluated using the fitness function provided. Afterwards, the following process will be executed as long as the termination condition is not fulfilled (e.g. as long as the current number of fitness evaluations performed is below the maximum number of evaluations allowed): For each individual in the population, an offspring is created using the weighted difference of parent solutions (see figure 2.7 for details regarding the creation of offspring).

The idea of using differences was novel (compared to the existing PBSMs in 1995). The DE has been suggested in approximately 10 variants, with different

offspring creation schemes. Common to them all is that differences between individuals in the population are used (typically added) in the generation of new individuals. In the comparison study of DE, PSO, and EAs (appendix E), we used the *DE/rand/1/exp* scheme shown in figure 2.7. The "rand" and "1" in *DE/rand/1/exp* tell that the scaled difference between two *randomly chosen* individuals is added to only *one* individual, respectively. In other schemes, the difference may be added to the *best* individual, and it may be added to sums and differences of *more than one* (e.g. 2) individuals. Finally, "exp" indicates that the above updating only concerns are consecutive stretch of the variables, whose start and end points are randomly chosen. The updating may also be done uniformly and independently for each variable, and is then called "bin". The offspring replaces the parent if it is fitter. Otherwise, the parent survives and is passed on to the next iteration of the algorithm.

procedure create offspring $O[i]$ from parent $P[i]$ {
  $O[i] = P[i]$ // copy parent to offspring
  randomly select parents $P[i_1]$, $P[i_2]$, $P[i_3]$, where $i_1 \neq i_2 \neq i_3 \neq i$
  $n = U(0, dim)$
  for $(j = 0;\ j < dim \wedge\ U(0,1) < CR;\ j{+}{+})$ {
    $O[i][n] = P[i_1][n] + F \cdot (P[i_2][n] - P[i_3][n])$
    $n = (n + 1) \bmod dim$
  }
}

Figure 2.7.: Pseudo-code for creating an offspring in DE with the *rand/1/exp* offspring creation scheme. $U(0, x)$ is a uniformly distributed number between 0 and x. CR is the probability of crossover, F is the scaling factor, and dim is the number of problem parameters (problem dimensionality).

2.6. Relation to No Free Lunch Theorem

As mentioned in section 2.4.3, there is a trade-off between fast convergence and premature convergence. This was formalised in the so called *"No Free Lunch"* (NFL) theorem, which stated that when averaged over all conceivable problems, no optimisation method is better than random search [167]. In fact, all algorithms (thus including random search) perform exactly the same – again averaged over all problems.

Although the implications of the NFL theorem were generally known and accepted without proof by many researchers, this result was problematic, because

it might imply that it was useless to continue the large and growing research area dealing with modifications and slight improvements of the performance of existing probabilistic heuristics (we refer to section 2.5 above for a presentation of "the top of the iceberg"). Luckily, "all problems" is a very abstract notion and this whole class is rarely dealt with when solving practical problems. For instance, it is clearly non-sense to create an algorithm for performing well on a completely "random" problem instance. To exemplify such a problem instance, we might define the search space $\mathcal{S}$ as the unit square in two dimensions, and the fitness function as pre-calculated for each input value as a random, real-valued number between 0 and 1. On such a problem instance, clearly no systematic search for the best fitness value is meaningful (except for exhaustive search), and it is equally clear that random sampling is an unbeatable strategy.

For more structured and restricted problem classes, with correlation between input and output values and "some continuity", general improvements might be made (this is not impossible according to the NFL theorem). On the other hand, it is very hard to actually prove that such improvements have been achieved, because very little typically is known (and can be stated) about a general problem class. Only extensive testing can make it likely (but not prove) that improvements have been made over other methods.

EAs are good when no (or little) knowledge is present about the problem domain. However, sometimes a search space can be searched more efficiently by building custom made algorithms instead of pushing the knowledge into the framework of an EA. This is e.g true for structural comparison: By exploiting the nature of protein structures — and the types of solutions one is interested in — it is possible to make an almost full search with a deterministic approach (see sections 5.2 and 5.6.2). Random elements could be introduced, but when the domain knowledge is great and the problem is well-structured, it can be difficult to incorporate it into the EA operators.

Modern heuristics like EAs are general purpose optimisation algorithms; they perform well on a wide range of problems, but may not be as efficient on specific problems as an algorithm specifically designed for it. For instance, the Lin-Kernighan heuristic [91] for the travelling salesman problem (TSP) will probably find better solutions for the TSP problem than a modern heuristic will (such as an EA) [76]. Conversely, Lin-Kernighan is only applicable to TSP, whereas an EA is generally applicable to a wide spectrum of problems. This is major advantage of modern heuristics: Their efficiency or applicability is not tied to any specific problem-domain (see figure 2.8).

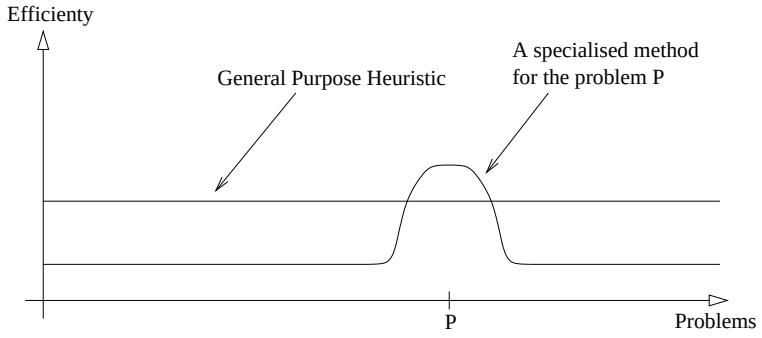


Figure 2.8.: Two efficiency spectra for a general purpose heuristic and a specialised method, respectively. The general purpose heuristic can always be used with some degree of success, while the specialised method will be very efficient for a narrow range of problems and be of little use on the majority of problems.

Chapter 3

DNA, Proteins, and Protein Structure

In this chapter, we will give an introduction to DNA, proteins and protein structure. We start by describing the genetic material DNA, and the processes by which it is transcribed into mRNA and translated into proteins using the genetic code. We then introduce the basic properties of proteins, their chemical components, and organisation. This is followed by a section with motivation for – and explanation of – the great interest that there is in proteins. Next, we describe standard representations of protein structure, and introduce the secondary structure elements (SSEs) of proteins. Continuing, we present the four levels on which proteins are usually analysed, whereafter we introduce the concept of domains. After this we introduce some descriptions of simplified protein structure representations, followed by a short discussion of available databases with structural content.

The concepts introduced in this chapter are important in the further discussion of protein structure comparison in chapter 5 and in relation to chapter 6 on protein evolution and homology detection. Moreover, both published papers on structural comparison, which may be found in appendices F and G, assume a significant amount of knowledge about proteins and protein structure. Thus, for the reader unfamiliar with these concepts, this background chapter is essential.

In particular the introductory sections of this chapter are based upon a set of textbooks on bioinformatics, molecular biology, proteins, and protein structure. We have chosen to list these books below with detailed annotations of the sections that have been used from the books, rather than citing them meticulously at every occasion throughout the chapter. The books and sections are: *The Cell* [1, Chapter 3], *Bioinformatics* [102, chapter 10], *Protein Bioinformatics* [38, pp. xiv-xvi and pp. 165-167], *Protein Geometry* [148, chapter 1], *Protein Architecture* [89, chapter 4], and *Protein Evolution* [114, chapter 1].

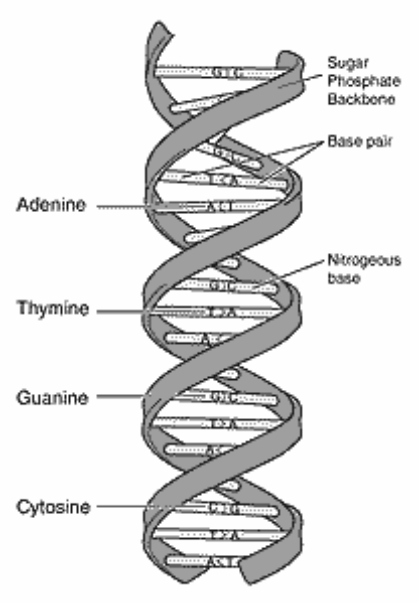


Figure 3.1.: A schematic drawing of the double helix, whose 3D structure was determined in 1953 by Watson and Crick. Note that adenine pairs with thymine, and that cytosine pairs with guanine (picture source: [71]).

3.1. From DNA to Proteins

Most organisms store genetic material in double-stranded DNA molecules, whose 3D structure (see figure 3.1 for a sketch) was famously determined in 1953 by Watson and Crick [162]. The genetic information is encoded in a sequence of nucleotides (adenine A, guanine G, cytosine C, and thymine T) along each strand. The nucleotide sequences on the strands are complementary and thus 100% redundant: The purines (A and T) pair with each other, while the pyrimidines (C and G) similarly pair with each other. Such a pairing is called a base-pair [114].

The parts of the genetic material (the DNA) that encode particularly essential and indispensable information are called *genes*. Conventionally, the term gene refers to the structural genes, which are the transcribed (see below) parts of the DNA. During transcription, one of the DNA strands is used as a template to create a pre-mRNA string, which contains the information to create proteins and which is an exact copy of a DNA strand, except for thymine (T) being converted into the chemically very similar uracil (U). Some structural genes only get transcribed (RNA-genes), while others, the protein-coding genes, are translated into proteins as well. The focus of this thesis – and the general focus in protein bioinformatics – is on the latter ones, the protein-coding genes, that we from here on shall refer to just as *genes*.

Returning for a short moment to the broader definition of a gene, genes consist of regions that get transcribed and some that do not, the non-transcribed regions. The transcribed regions are called the primary transcript. Transcription is controlled by promoters for initiation as well as signals for termination of transcription.

The primary transcript consists of introns and exons. Introns are removed later during splicing when pre-mRNA turns into messenger RNA (mRNA). Exons are the parts that remain in mature mRNA after splicing. The parts of exons that subsequently get translated, and subsequently may fold into a protein structure, are called the protein-coding regions [114].

We now take a look at translation in more detail. Following splicing, the mature mRNA moves from the nucleus to the cytoplasm (both of which are located inside the cell). The mRNA is subsequently translated by ribosomes into one or more polypeptide chains. The ribosome reads three nucleotides at a time, a codon, and then adds one amino acid (see section 3.2) to the growing polypeptide molecule. Each codon specifies one particular amino acid. This systematic and deterministic translation scheme is called the *genetic code* (see figure 3.2). There are 64 possible codons but only 20 amino acids, so several codons may specify the same amino acid; the code is highly degenerate. The initiation of translation requires the codon AUG to appear in the mRNA string.

3.2. Proteins

Polymers are sequential molecules of smaller recurring molecules. Proteins are made up from *amino acids* that are connected by *peptide bonds* to form polymers. This particular protein polymer consisting of amino acids is called a *polypeptide (chain)*. A protein may consist of one or more polypeptide chains.

An amino acid is a molecule that contains an *amino* functional group, -NH_2, and a *carboxylic acid* functional group, -C(=O)-OH. Moreover, it consists of a central alpha carbon (C_α) atom, with four covalent bindings: One to the amino group, one to the carboxylic acid group, one to a hydrogen (H) atom, and one to a so-called *side-chain*, which may be any organic molecule. The general structure of an amino acid is illustrated in figure 3.3. Since the side-chain of an amino acid as mentioned can be any organic molecule, there could in principle exist an infinite number of amino acids. In nature, over 100 different have been found. However, only 20 different amino acids appear in proteins, because only 20 amino acids are created by the ribosomes in protein synthesis.

When peptide bonds are formed between amino- and carboxylic acid groups from adjacent amino acids, a water molecule is released in the process; the remains of the amino acid is now called a *residue*. After all the amino acids have bonded to become residues, the *backbone* of the protein has come into existence. It consists of three atoms — namely the nitrogen (N) atom from the amino group,

Codon	Amino acid	Codon	Amino acid	Codon	Amino acid	Codon	Amino acid
UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys
UUC	Phe	UCC	Ser	UAC	Tyr	UGC	Cys
UUA	Leu	UCA	Ser	UAA	Stop	UGA	Stop
UUG	Leu	UCG	Ser	UAG	Stop	UGG	Trp
CUU	Leu	CCU	Pro	CAU	His	CGU	Arg
CUC	Leu	CCC	Pro	CAC	His	CGC	Arg
CUA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CUG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser
AUC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
AUA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
AUG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly
GUC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GUA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GUG	Val	GCG	Ala	GAG	Glu	GGG	Gly

Figure 3.2.: The genetic code

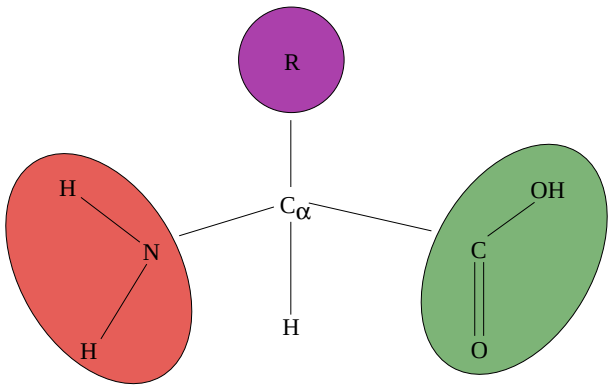


Figure 3.3.: General structure of the *amino acid*. The amino group is to the left, the carboxylic group is to the right, a hydrogen atom is in the bottom, and the side chain is in the top of the figure (denoted by the letter R).

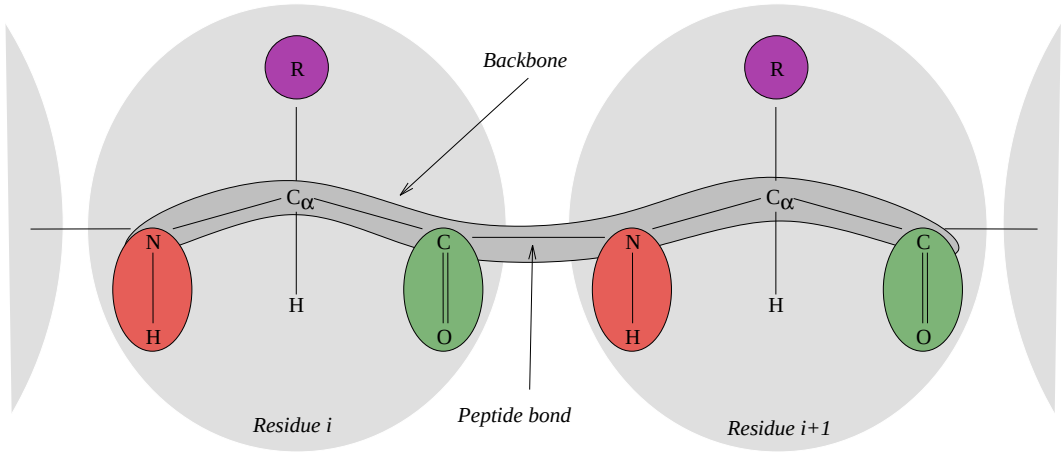


Figure 3.4.: The backbone of a protein

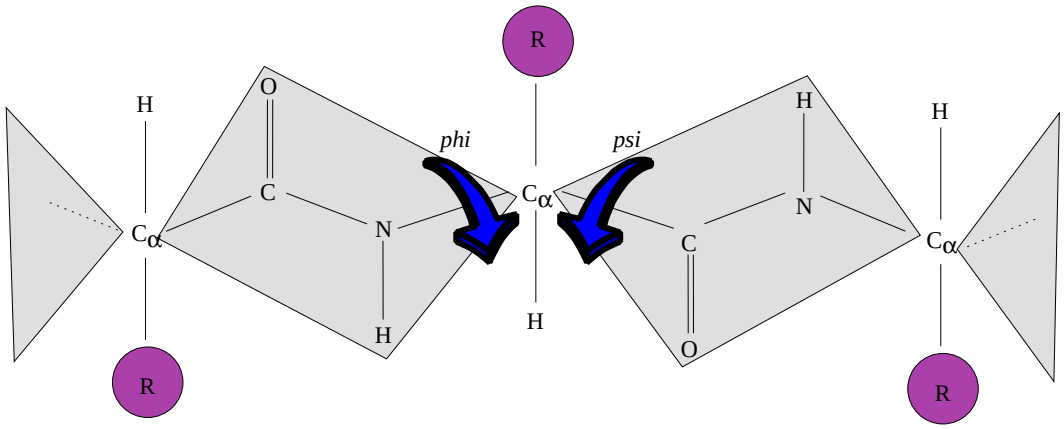


Figure 3.5.: The rotatable bonds of the protein backbone. The two C_α atoms from adjacent residues and the H, N, C, and O atoms "between" the C_α atoms, all lie in a plane.

the central C_α atom, and the carbon (C) atom from the carboxylic acid group — repeated in triplets, one for each residue (see figure 3.4).

The peptide bonds between residues are rigid. Thus, there are only two types of rotatable bonds along the protein backbone: The bond between the C_α atom and its N neighbour, and the bond between the C_α atom and its C neighbour. This means that the overall 3D structure of a protein in principle is determined simply by the rotational states of these two bonds in each residue. The angles of these two bonds a are commonly denoted by ϕ *(phi)* and ψ *(psi)* (see figure 3.5).

Each residue is typically represented by a letter from the Latin alphabet (see figure 3.6). Thus, by the *sequence* of a protein is understood the letter codes for

Alanine	Arginine	Asparagine	Aspartic acid	Cysteine
A	R	N	D	C
Glutamine	Glutamic acid	Glycine	Histidine	Isoleucine
Q	E	G	H	I
Leucine	Lysine	Methionine	Phenylalanine	Proline
L	K	M	F	P
Serine	Threonine	Tryptophan	Tyrosine	Valine
S	T	W	Y	V

Figure 3.6.: The amino acids and corresponding one letter codes.

its amino acids written in the order they appear starting from the amino (NH_2) end of the polypeptide chain. The typical length of a protein varies from 50 to 300; the distribution of lengths of proteins is given in figure 3.7.

3.2.1. Why Proteins?

Proteins are *the* driving forces in the execution of almost all processes taking place in every organism living on earth. This is, indeed, a strikingly central role, and thus understanding proteins, the way they act and fold – according to some researchers – is the key to understanding life itself [148, by Jacques Monod, page 1]. While DNA and other macromolecules such as lipids and carbohydrates are also very important to organic life, proteins differ by being the active agent – proteins chop the other molecules, catalyse reactions, translate DNA, and so forth, while other macro-molecules are passive, rigid, and being acted upon. In fact, proteins are so central to life and have so complicated processes that a protein might very well be regarded as the closest we get to a living organism, without being one [148].

Generally, there are two main classes of proteins. The *globular* proteins that play an active role in catalysing metabolic processes, replication, and expression of genes. These proteins can be thought of as the workhorses of the cell. They tend to be non-repetitive and between 100 and 300 residues in size; they are typically compact and sphere-shaped. The other class, *fibrous* proteins, are more passive, and often serve a structural purpose. For instance, nails and hair are composed of fibrous proteins; the fibrous proteins can be thought of as building blocks. They are repetitive and are composed mainly of α helices (see section 3.4.1). Finally, there is also a class called *membrane* proteins. As the name suggests, they reside around and inside the cell membranes, where they control and regulate transport in and out of the cells of various atoms and molecules. Membrane proteins also serve as message passing devices between cells.

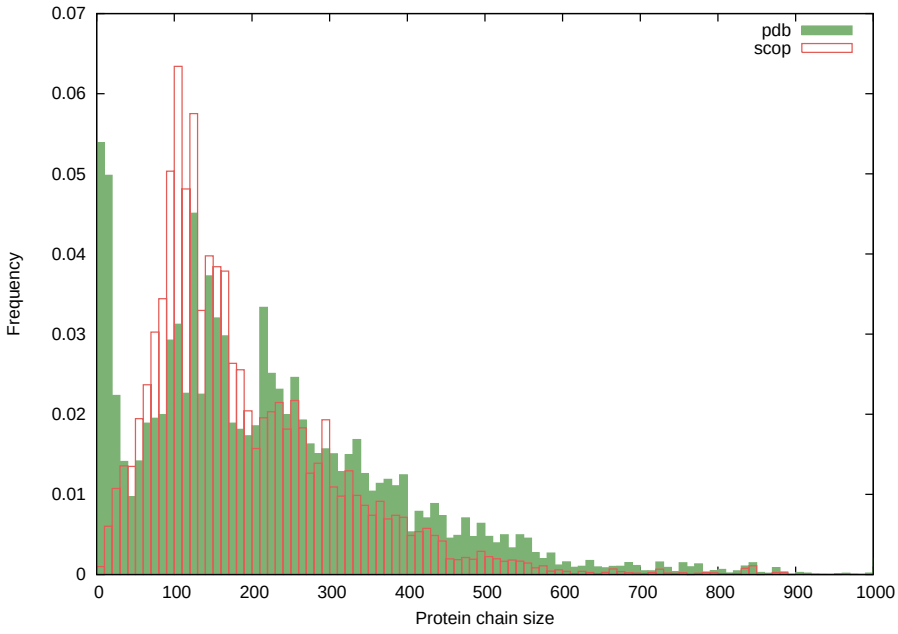


Figure 3.7.: The frequency of the lengths of the 76633 protein chains in the Protein Data Bank (PDB) (see section 3.8.1 on page 48) and the 70859 protein domains in the SCOP database. Many PDB chains longer than 200 residues have been divided into shorter domains by SCOP, as witnessed by the higher frequency of chains longer than 200 residues in PDB. Moreover, many PDB chains shorter than 40-50 residues have been left out in SCOP, as witnessed by the lower frequency of chains shorter than than 50 residues in SCOP.

Protein Folding

There are plenty of reasons and motivation for wanting to understand proteins. One reason is the "holy grail" in structural bioinformatics, namely tertiary *structure prediction*, where the three-dimensional structure of a protein is sought predicted from its amino acid sequence. This is also known as the protein folding problem. Early experiments showed that proteins spontaneously could fold to their native, functional conformation [7]. This finding seemed to imply that it should be possible to determine the structure of a protein through its sequence, and thus theoretically it should be possible to find a deterministic algorithm that translates from the primary level, a letter sequence, to the tertiary level, 3D coordinates (see section 3.5 on page 42). However, the actual biological process by which this – folding – happens is far from understood. Consequently no such algorithms exist, although it is sometimes possible to predict (with some uncertainty) the 3D structure of a sequence given a template structure found by sequence similarity search for the query sequence (homology modelling). When refraining from using known structures in the prediction, and only using the available sequence information – the *ab initio* approach, meaning: *from the beginning* or *from first principles* – the prediction is even more uncertain. The progress of the structure prediction field is measured in the bi-annual competition, Critical Assessment of Structure Prediction (CASP) [87], where proteins with known structures (that are kept secret by the crystallographers until the end of the competition) are sought to be predicted. When folding takes place in cells, there are other factors such as e.g. the cellular environment that do also influence folding to a high degree, which makes it very unlikely that there is a simple algorithm for predicting the 3D structures of proteins. For broader introduction to protein folding, see e.g. [148, section 9.3] or [25, chapter 11].

Other Problems

This above challenge is so difficult that it is often broken down and dealt with on many levels and "attacked" from various perspectives. Thus, another path is to concentrate on more tractable problems that may still add to the knowledge about proteins. E.g. analysing the existing variety of structures, to classify them, and find patterns, similarities, and differences, helps to get an increased – if not understanding then – overview of existing proteins. This still requires development of tools, such as sequence comparison and structure comparison programs, to analyse the vast amount of available data in an efficient and reliable way. Thus, structural comparison — among other disciplines — is important, because it enables the progress for hard problems by solving many of the "simpler" ones.

3.3. Representation

Protein structure can be represented in various ways. Three of the most used are standard 3D Cartesian coordinates, torsion angles, and internal distances also known as distance matrices.

Cartesian coordinates are simply the raw 3D coordinates of the atoms that are included in the description. Torsion angles are the previously described ϕ and ψ angles (section 3.2) – both angles must be saved for each residue. Internal distances are the distances between all pairs of C_α atoms in the protein. The distances are stored in a quadratic matrix, where entry (i, j) is the Euclidean distance between the C_α atoms of residue i and j.

Cartesian coordinates and distance matrices are widely used in protein structure applications, whereas torsion angles are more rarely used. It is possible to convert between the three types of representations. However, converting from torsion angles or internal distances to Cartesian coordinates might produce a mirror image of the true structure (known as a *chirality*) [148]), because the distance matrix representation cannot distinguish mirror images from one another.

In addition to deciding *how* to represent the available information, it must also be decided *what* information to include in the description. Some possible choices in this context are:

- C_α atoms only (which in the most typical choice in structural comparison),
- all backbone atoms (N, C_α, C, N, C_α, C, N, ...),
- one of the above including a description of the type and position of the side-chain,
- all known atoms of the protein.

Although most structural comparison applications only utilise the positions of the C_α atoms, in some cases the position of the side-chain is included for extra accuracy; this is represented by the C_β atom from the side-chain.

3.4. Secondary Structure

When a protein folds, it minimises its energy potential by forming hydrogen-bonds between parts of the chain. Some interactions are between residues near in the sequence, and others are long range. Short range interactions give rise to recurrent local structural motifs, known as *α-helices* and *β-strands*. These are known as *secondary structures*, and it is the description from this perspective that is known as the secondary level. A secondary structure element (SSE) is a sequence of residues with the same secondary structure type.

One SSE is the *alpha helix* (α-helix), which is characterised by hydrogen bonds between the carbon atom from the carboxylic acid group of residue i and the nitrogen atom from the amino group of residue $i + 4$. This formation results in a helical structure of the backbone with 3.6 residues per turn (see figure 3.8).

The other main SSE is the *beta sheet* (β-sheet). A beta sheet is built up from combinations of several beta strands – parallel regions of backbone; thus, while the alpha helix is continuous, the beta sheet is discontinuous. Beta sheets consist either of parallel or anti-parallel beta strands with each variant having its own particular hydrogen bonding pattern (see figure 3.8).

3.4.1. Secondary Structure Assignment

Secondary structure assignment is the task of assigning each residue to either the helix type, strand type, or as none of the two. Note, that this is *not* SSE prediction. In SSE prediction, only the primary sequence is known, while in SSE assignment the tertiary structure is known.

It is not trivial to assign SSEs, and often there is not a unique or "best" assignment. There are differing available techniques for defining SSE boundaries. Some methods – such as DSSP [79] – are based on analysis of hydrogen bonding patterns along the backbone. An α-helix, for instance, is easily detectable by its hydrogen bonding pattern mentioned above — consecutive bonds between residues i and $i + 4$.

Other techniques – such as STICKS by Taylor [147] – is based on the geometry of the C_α atoms from the backbone. It finds SSEs by first gradually smoothing the backbone (e.g., by averaging the C_α atom positions), and then detecting point sub-sequences that lie on a straight line. Such sequences are likely to be SSEs. An advantage of the latter approach is that it is applicable and robust when only C_α atoms are given, and does not require hydrogen bonds to be known.

3.5. Levels of Analysis

Proteins are typically looked upon and analysed on four levels: The primary, secondary, tertiary, and quaternary level.

- The *primary level* is the amino acid letter sequence, and is thus a one dimensional description of the protein.
- The *secondary level* is the amino acid letter sequence, but with annotations for each residue that specify which type of SSE (if any) the residue is part of. A residue need not be in a helix or strand, in which case it is classified as being in a loop region.
- The *tertiary level* is the actual 3D structure of the protein. Typically, this is specified by the (x, y, z) coordinates of the atoms of the protein.

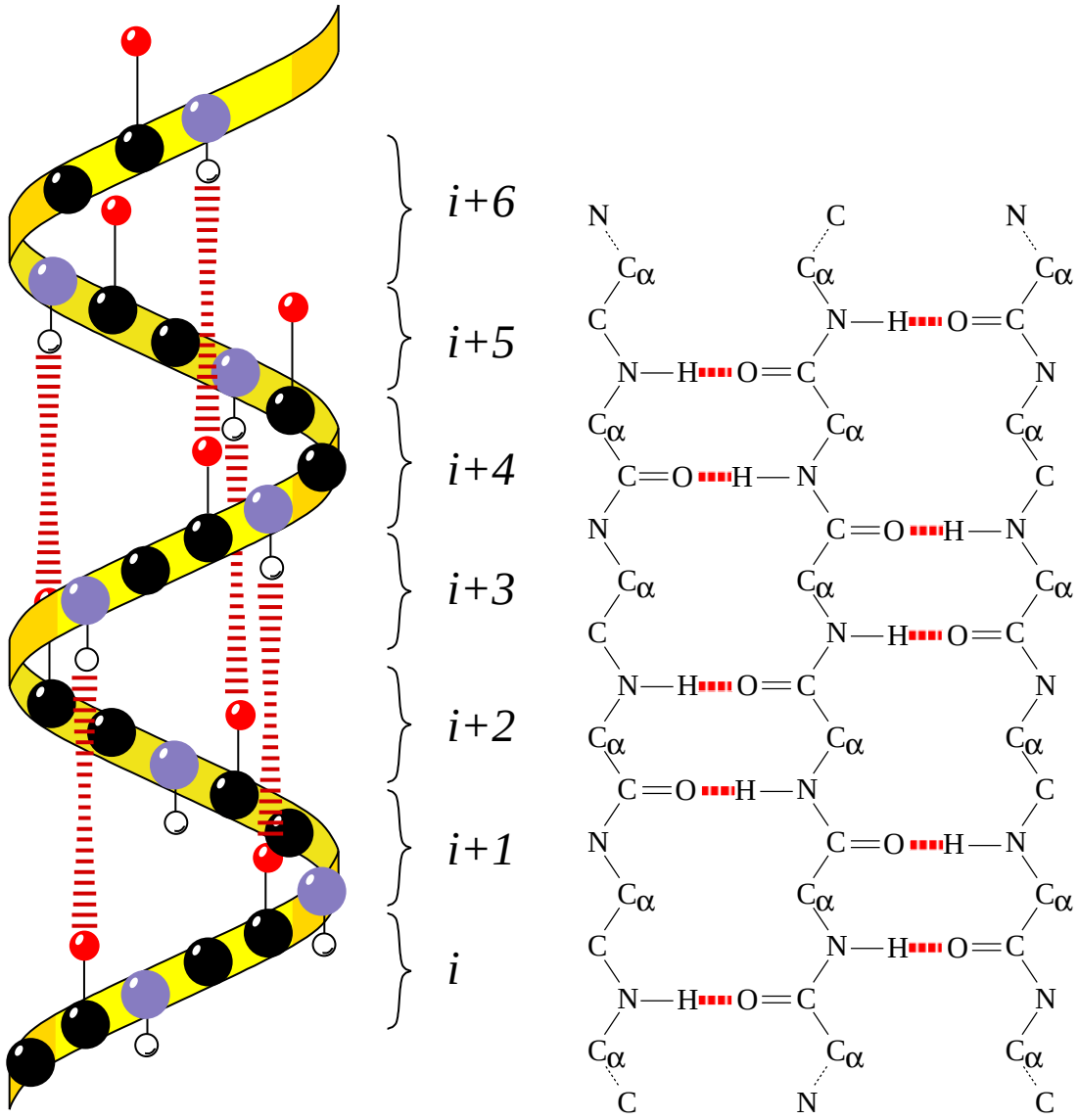


Figure 3.8.: A schematic drawing of the alpha helix (left). The section shown includes seven residues of a helix. One the right hand side, an antiparallel beta sheet is shown using a more schematic drawing style. For the more conventional, high-level cartoon drawings of SSEs, see figure 3.9 or 5.1, where both types of SSEs are present. ***Legend:*** Blue is nitrogen, black is carbon, white is hydrogen, and red stripes indicate hydrogen bonds between atom pairs.

- The *quaternary level* is the interaction of the chains of the protein, which may be specified again by the (x, y, z) coordinates of the atoms. Some proteins have only one chain, in which case the the quaternary level does not add information to the tertiary level information.

3.6. Domain Definition

Most proteins with more than 200 residues consist of two of more seemingly independently folded structural units, also known as *domains*. The exact definition of a domain is, unfortunately, somewhat unclear. Nevertheless, the following properties are all characteristic for a structural domain, and may be employed in defining them:

- The residues within a domain interact more with each other than with residues from other domains.
- A domain should have a hydrophobic core and contain at least 40 residues.
- The domain should stay stable and folded if isolated from the rest of the chain. Although this experiment is hard to carry out in practise and also hard to simulate, it is a useful abstract requirement in domain definition and detection.
- A domain should be associated with a specific function.
- It should not be possible to divide a domain without a significant increase in accessible surface area. This means that two solid 3D shapes that are connected only by a thin connecting region cannot be one domain, because they could be divided at the connecting region without significant increase in accessible surface area.

The formal requirement to a domain is that it is a sequence of substrings from the polypeptide chain, which is fact is any subset of residues from the backbone. For instance, the protein structure with PDB identification code (PDB ID) 1pys (see figure 3.9) consists of two chains, A and B. Chain A is 85 residues long and has only one domain. In contrast, chain B is 785 residues long and consists of 6 domains. Five of these consist of only one continuous substring of residues (b2: 400-474, b3: 39-151, b4: 682-785, b5: 475-681, b6: 191-399), while one domain, b1, consists of two discontinuous substrings (b1: 1-38+152-190).

Thus as we have seen, a domain can be a complete chain, it can consist of one of more potentially discontinuous substrings from a chain, but it cannot span several chains. Although a domain is not a strictly well-defined unit, it is a very practical concept in many respects. In particular in structural comparison, it is common practise to compare domains rather than whole proteins. Considering

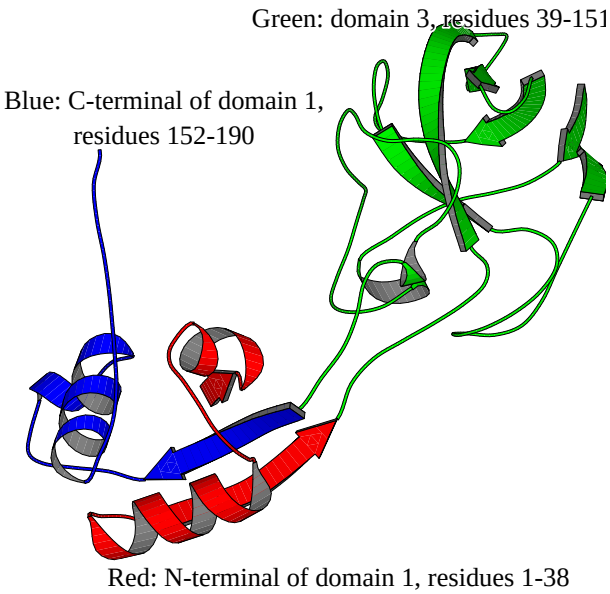


Figure 3.9.: Domains b1 and b3 from the B chain of 1pys. The N-terminal end of b1 is coloured red, while the C-terminal end of the same domain is coloured blue. b3 is coloured green (see text for further details).

the fact that a domain is believed (and indeed appears) to be an independent folded unit, it makes perfect sense *not* to take several domains (or whole proteins) as one unit of comparison, but instead use single domains.

The are many available databases with domain definitions, amongst which the most used are SCOP [6] and CATH [115]. The main purpose of these databases is to classify the known protein structures. However, in doing so they also "chop" the structures into domains; in fact, the classification of proteins is done on the domain level in these databases. In this thesis, the work on structural comparison is based on the SCOP database, and thus implicitly uses the SCOP domain definitions. For instance, the fourth paper of my Ph.D. project (see appendix G) on detection of novel circular permutations using FASE (see section 5.6), was based on comparisons between domains as they were defined by SCOP.

3.7. Simplified Representations

For some problems involving protein structure, it may be possible to use a simplified representation of the data. In structural comparison, for instance, many types of simplified representations are used. The goal is often to increase the speed of the computations by neglecting certain details of the problem.

3.7.1. Line Segments Representing Secondary Structures

An commonly used simplification is the representation of SSEs by line segments. Thus, instead of explicitly storing the (x, y, z) coordinates of the C_α atoms (and perhaps also C_β atoms) for a number of residues, this sequence might instead simply be represented by a line segment, which can be defined by a start- and an endpoint, each given by a (x, y, z) triple.

The standard approach for defining the line segment is to find the line segment with a least squares fit to the C_α atoms of the SSE that the segment is supposed to represent. There are efficient methods for solving this task [148, page 52] (similar to those for superposing two backbones, see section A.2). Alternatively, one may define the line segment by finding its endpoints as the average of the three (or four) end-most points from each end of the SSE.

The line segment representation not only saves space, but also facilitates simpler and thus faster computations: It is easier to determine the similarity between two line segments, than it is to determine the similarity between two SSEs that might consist of 15 atoms each. Some structural comparison programs, such as VAST [49], operate exclusively on these simplified representations, whereas other programs just use the simplification as a temporary representation to achieve a speed up in calculations during the comparison (e.g. LOCK [137], see section 5.5.2).

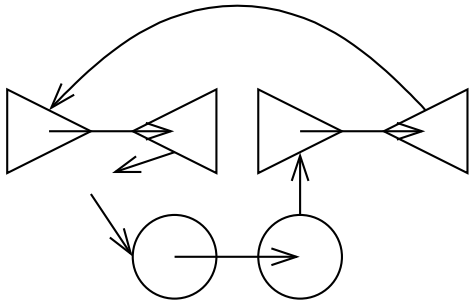


Figure 3.10.: A TOPS cartoon of (scop d1fjgc2). See section 3.8.2 for an explanation of the identifiers used in SCOP.

The line segment representation for SSEs is a good example of the advantages and disadvantages by employing simplifications: Simplifications reduce data complexity and size, thus allowing for faster processing. This is very beneficial in applications that must run fast, and is a crucial "trick" in many structural comparison programs (including FASE). However, if the data is simplified too much, then finer and important details might be lost, thus giving inexact results. In structural comparison, the worst case scenario would be to miss a significant similarity between a pair of structures due to oversimplification. Obviously, the degree of simplification is a difficult trade-off to manage, because — on the other hand — if simplified too little, the desired efficiency may not be gained, and the whole purpose of the simplification is lost.

3.7.2. TOPS Cartoons

TOPS cartoons [43] are very compact and simplified representations of protein structures. Alpha helices are represented by circles, the beta strands by triangles, and loop regions by lines that connect the secondary structures. The 3D structure is projected into 2D by imagining that all the SSEs run perpendicular to the plane — into the paper (see figure 3.10). The cartoons are very useful when comparing the overall similarities of two structures manually; in the fourth paper included in the thesis (see appendix G), where we present and analyse novel circular permutations, we make use of the cartoons to assess the found similarities manually.

The generation of TOPS cartoons can be made automatic, and this has of course created the idea of doing structural comparison not on the raw protein structure data, but on the derived TOPS cartoons. This method has been used to make a large scale classification of protein structures. However, the procedure for automatic TOPS cartoon generation is not always accurate – it fails in app. 18% of the cases [164], which makes the comparison method based upon this technique equally weak. This fact is not completely surprising, since not all structures

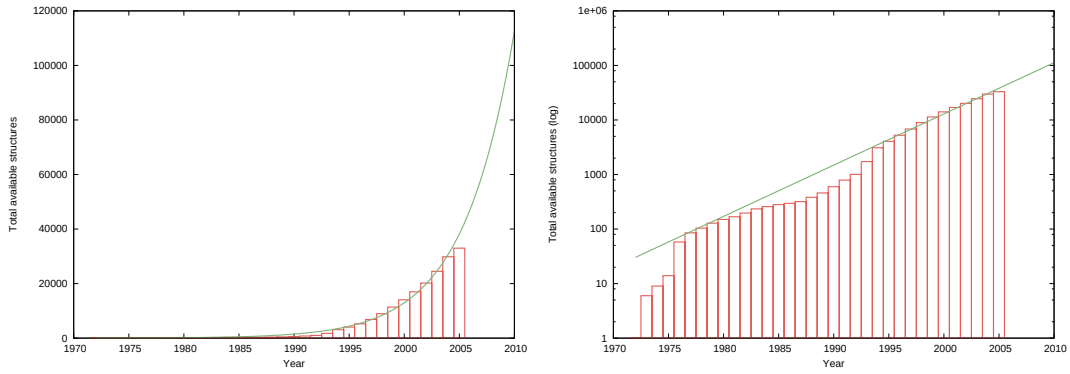


Figure 3.11.: The increase in number of structures in the PDB database (linear scale on the left and logarithmic scale on the right). The actual number of structures is shown with bars. The function $f(x) = 30.2 \cdot 1.24^{x-1972}$ (the line) has been fitted to the development from 1975 to 2004 (the fit was done by gnuplot, which uses the Marquardt-Levenberg algorithm). The fitted function implies a doubling of the number of structures each 3-4 years, and it also shows that more than 100000 structures can be expected in PDB by 2010.

conform to the assumption that all SSEs are parallel and can be orientated run into the paper.

3.8. Database Classifications

3.8.1. The PDB Format

All known protein structures are stored in the Protein Data Bank (PDB) [14] in the PDB format. There has been an approximately exponential growth in the number of structures in the database over the last 20 years (see figure 3.11).

In the PDB format, each atom, its type, and (x, y, z) coordinates are stored. In addition to protein atoms, a PDB file typically also contains atoms from a ligand — a smaller molecule that interacts with the protein — as well as water molecules that interact with the protein and ligand. For the protein atoms, the residue number and type of residue are also stored. Each atom takes up a single line in the PDB file; for instance, the line below is line 1748 from the PDB file with the name 1HBI:

```
ATOM   1594  CA  VAL B  57      22.894 -12.123   6.484
```

The line reveals that there is a carbon atom at the position $(22.894, -12.123, 6.484)$. Moreover, the "`CA`" shows that it is the central C_α atom of a residue, namely

residue 57 of type valine from chain B, which thus is a part of the protein with PDB ID 1HBI. The value 1594 is a unique atom identifier within the file.

3.8.2. SCOP

The database *Structural Classification of Proteins* (SCOP) is a classification of all known protein structures according to structural and evolutionary relationships [6]. More precisely, not whole proteins are classified, but rather protein domains. The unique identifier of a domain in SCOP is formed by concatenation of the letter 'd', the PDB ID of the protein that the domain is a part of, the chain letter, and a digit indicating its domain number within the chain. For instance, the domain from figure 3.10, which is the second domain of chain C from the protein with PDB ID 1FJG, is given the SCOP identifier (scop d1fjgc2). If a protein has only one chain (in which case it is often unnamed), or if a domain is the only one in a chain (in which case numbering of this single domain appears artificial), then the character '_' is used as indicator on both of the last two positions of the SCOP identifier. The actual part of a PDB file corresponding to a SCOP domain can be retrieved from the ASTRAL database [19].

In contrast to other similar databases such as CATH [115] and FSSP [68], the SCOP classification has been constructed largely by manual comparison and inspection of structures. Computational tools have, of course, been used, but only to overcome the huge classification task and to provide some degree of objectivity. In recent years, the SCOP classification has become a "gold" standard for assessing the evolutionary relatedness between protein structures.

The SCOP Levels

The SCOP classification operates with seven levels of similarity. Starting with the levels that indicate the least degree of similarity they are: Class, fold, superfamily, family, protein domain, species, and entry domain. For an illustration of this, see figure 3.12. The "scale" of the tree is 300:1, which means that the app. 900 folds in SCOP are depicted as three nodes in the top level of the tree, and that the app. 1500 superfamilies in SCOP are depicted as five nodes in the second level. Thus, the relative number of nodes in each layer is approximately accurate. The two leftmost folds illustrate the fact that 602 of 945 folds (64%) only have one family; the two leftmost families illustrate the related fact that 602 of 2845 families (21%) belong to a fold with only one family. Counting all entries, 9936 of 67221 (15%), belong to a fold with only one family.

There are 11 different classes in SCOP. Four of these are not true classes; they are e.g. short peptides, low resolution structures, and engineered proteins. The ones where the majority of structures reside — and the most important ones — are: All *alpha*, all *beta*, *alpha/beta*, and *alpha+beta*.

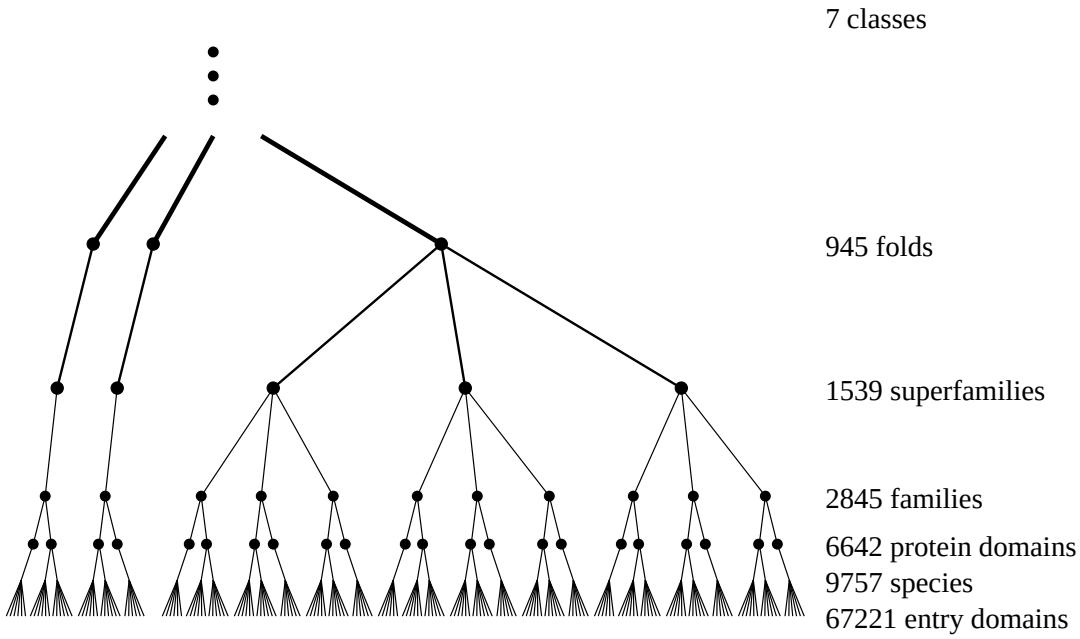


Figure 3.12.: Illustration of the SCOP classification hierarchy. See text for details.

Proteins are classified into either of these four classes according to their SSE content. The all alpha class consists of proteins whose SSEs are almost only alpha helices, and similarly the all beta class consists of proteins whose SSEs are almost only beta strands. The remainder of structures are classified in the mixed alpha beta classes. If the alpha and beta SSEs are alternating sequentially in the structure, it is classified in the alpha/beta class, whereas if the SSEs of same type are clustered together sequentially, the structure is classified in the alpha+beta class. The three remaining actual classes are multi-domain, membrane, and small proteins, respectively. Figure 3.12 is based upon the mentioned seven actual classes in SCOP.

Below the class level comes the *fold* level. The fold level indicates *major structural similarity*. A pair of structures are classified in the same fold, if they have the same *major SSEs in the same arrangement and topological connectivity*. The regions that do not have the same size and conformation within a fold can comprise up to half of the residues in a pair of structures. Such differences are typically located at the end of SSEs and in loop regions. The related CATH classification level, *architecture*, differs from the fold level by not requiring the topological connectivity constraint.

The *superfamily* level (below the fold level) groups together structures with a *probable common evolutionary origin*. Proteins with low (insignificant) sequence similarity, but whose structural and functional features suggest a common evolutionary origin, are grouped in the same superfamily.

The next level is the *family* level. A family in SCOP indicates a *clear evolutionary relationship*. Generally the sequence identities (between the sequences for structures belonging to the same family) are above 30%. However, in some cases structural and functional features can provide the evidence alone, in spite of lower sequence similarity.

In SCOP, the structures are further classified into *protein domains*, *species*, and *entry domains*. In protein domains, all entries are essentially identical in the sense that differences are insignificant with respect to the SCOP classification. The further subdivision of protein domains into species simply splits the protein domains according to the species, from where the structure came. Finally, a species is divided into entries, where each entry simply is a unique sequence and structure. Differences between entries within a species usually only amounts to the type of complexion (that is, the compound with which the protein was bound when determining its structure).

3.8.3. Other DBs

CATH [115] and FSSP [68] are two other databases with classifications of protein structures. With the increase in determined protein structures and the growing activity in structural biology, many new related databases have started to emerge; many of those are automatic, and based on some structural comparison program.

Compared to SCOP, the automatic classifications have some strengths as well as weaknesses: Automatically created databases cannot detect all structural and evolutionary relationships between pairs of structures, because some relationships are so complex that a fully correct classification of all pairs is still not possible. In theory, of course, this *is* possible, but it is a very hard classification problem. And in any case, it can not be solved using structural data alone; functional and environmental data must also be used. On the other hand, by using an automatic method the possibility of subjectivity and human error is eliminated. Also the possibility of overlooking a similarity is not possible – at least not if the comparison method finds all interesting similarities.

Chapter 4

Sequencing and Fragment Assembly

The present chapter on DNA sequencing and fragment assembly did also appear in my progress report [155] in an earlier version. The text has been modified, integrated, and updated with references to other parts of the thesis. In this chapter, the foundation is laid for the research on the Physical Mapping (PM) problem, done in the first half of my Ph.D. project, as well as for the first paper attached to this thesis (presented in chapter D), which deals with PM and analysis of heuristic algorithms for the problem.

4.1. Introduction

A very fundamental and important problem in bioinformatics is to determine the DNA sequences of various organisms. Many organisms have been sequenced, such as mice, yeast, bacteria, and humans. Sequencing is important, because bioinformatics deals with the manipulation and analysis of molecular biological data, and because DNA is the basic genetic information – the origin of most biological data. The importance of sequencing may also be understood by noticing the massive efforts put into sequencing the human genome by the Human Genome Project (HGP) and the private corporation Celera Genomics. Drafts of the 3 billion base pair (bp) long human genome – the complete set of genetic instructions for the development of humans – were published in 2001 by both enterprises. In April 2003, the HGP had sequenced 99% of the gene-containing part of the human sequence with 99.99% accuracy [27].

How is the human genome or any other genome sequenced? Unfortunately, it is impossible to start reading from one end of a DNA molecule to the other, while recording every base pair (bp). If this were possible, there would be no computational problem in sequencing; one would just have to record every nucleotide in a growing string of nucleotides. In contrast, the physical devices that read off the four types of nucleotides from the DNA string can record no more than 500-1000 consecutive bp reliably. Thus, when sequencing a genome (the human genome is 3000 million bp long), one has to assemble short segments into longer sequences

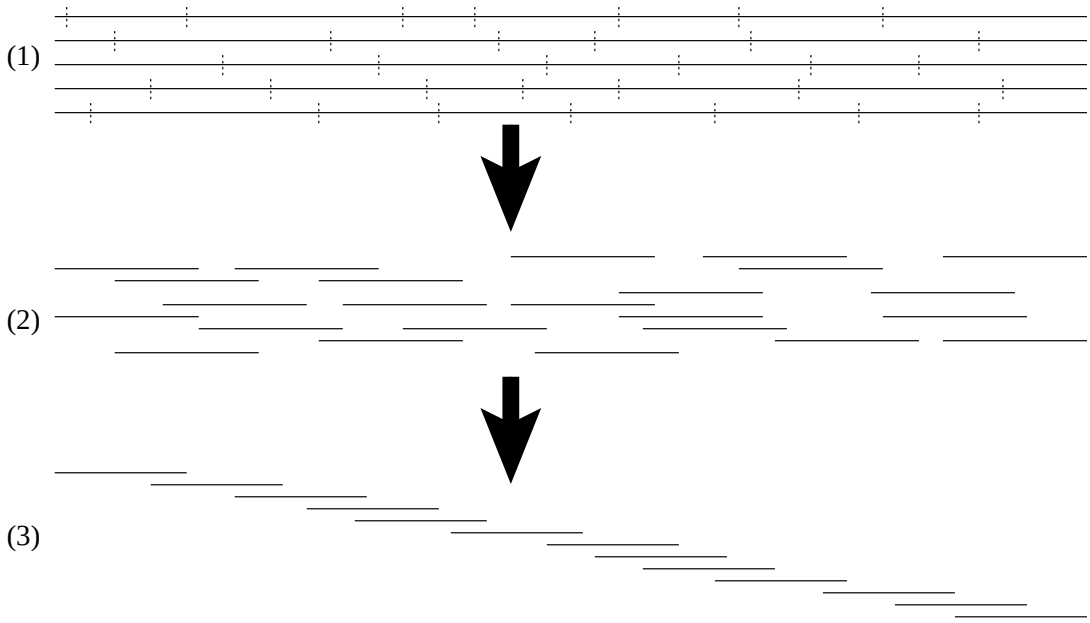


Figure 4.1.: Illustration of the stages in PM: (1) Copies of the source DNA string with future split points indicated. (2) Unordered DNA clones. (3) Ordered clones. Going from (2) to (3) requires solving the the computational clone ordering problem of PM.

that eventually reveal the sequence of the entire genome in question.

Two different approaches exist for genome sequencing: Clone-based shotgun sequencing and whole-genome shotgun sequencing. The former approach was used by HGP [27] and the latter by Celera Genomics [99].

4.2. Clone-Based Shotgun Sequencing

Clone-based shotgun sequencing (or Clone-by-clone sequencing) is a two-level hierarchical approach [72, pages 25–48]. The first level determines the relative ordering of a collection of large segments covering the genome, and on the second level each of the segments is sequenced on the nucleotide level.

Firstly, the whole genome is duplicated in order to obtain a set of copies of the genome. Each of these copies is then fractured into pieces of length 50 to 300 Kbp, each of which is called a *clone*. The copies are broken at different locations, and hence the clones will overlap. When a sufficient number of genome copies have been made and broken down, the clones will cover the genome to a very high degree, and it is therefore with high probability possible to find a list of overlapping clones that spans the genome (see figure 4.1).

The next step is to create a *physical map* (PM) of the genome. A PM is

merely an ordering of a minimal set of clones covering the genome. Thus, to find a PM one must select a set of clones that constitutes a minimal covering, and subsequently this set must be ordered according to the position of the clones on the DNA string. The term physical mapping comes from the fact that a physical map *maps* a given clone to its *physical* location along the DNA string. Typically, the precision of this mapping is not on the nucleotide level (in which case the exact DNA sequence in essence would be known); rather, the PM maps a clone into some interval of the genome. The computation of the physical map is based on overlap information between the clones. We shall return to the issue of computation of the PM in section 4.4.

There are several methods available for experimentally determining overlaps between clones. Common to all of them is that they create a "fingerprint" of each clone based on, for instance, the existence of a certain substring within the clone, or on the way the clone is broken down by a certain enzyme. If sufficiently many unique characteristics are common to the fingerprints of two clones, chances are that the clones will overlap [99].

The mostly used method, Sequence Tagged Site (STS), is based on the observation that two clones must overlap, if they both contain a certain unique DNA sequence. An STS marker is a pair of strings of length 18 that are separated by 200 to 1000 base pairs. An STS marker is unique with a very high probability, and the overlap information given by this method is therefore very reliable [99].

After having found a PM for the genome, each clone in the map is then sequenced using the shotgun sequencing approach. In this process, the clone is further broken down into even smaller fragments. The fragments are kept in a genetically engineered virus called a *vector*. The fragment is now called an *insert*. These inserts can be sequenced using a method invented by Sanger in 1980 that uses DNA polymerase and modified nucleotides called 'terminator nucleotides' (see [34, pages 23–28] for a good description). The output of this process is between 300 and 900 bp of information. After obtaining the experimental data, it is a computational problem, called fragment assembly, to provide the source sequence of the clone by combining the 500 bp of information from each fragment. This is definitely not a trivial task, although it is feasible to solve, and there are several software packages available for that purpose. Some of the inconveniences that complicate fragment assembly are incomplete coverage of the clone, sequencing errors, unknown orientation of reads, and major repeats in the genome [72, pages 25–48]. For example the presence of long repeated DNA sequences (quite disappointingly) implies that the fragment assembly problem cannot be formulated as finding shortest common super string (SCS) of the DNA reads; the SCS formulation might give a shortened clone sequence in the case of repeated sequences that are longer than the longest DNA read.

4.3. Whole-Genome Shotgun Sequencing

The other approach for genome sequencing, taken by Celera Genomics, does not use a PM. Basically, in this approach approximately 60 to 70 million shotgun reads of the genome are made, and it is attempted to assemble them into a genome [99]. The assembly process not only uses the raw sequence data but additional information (see below), without which assembling the whole genome would be very difficult.

The approach relies on sequence reads of very high quality. All reads have been filtered so that only reads with an error rate below are 1% are used. This makes overlap detection very reliable. Furthermore, it is possible to use end-sequence information from the inserts to resolve repeats up to the length of the inserts. Of all the inserts used, 80% of them have a length of approximately 2 Kbp and 20% have a length of length 10 Kbp. To resolve even longer repeats, additional end-information from clones of lengths of 600 Kbp can be used.

Besides getting enough high quality data to perform whole-genome shotgun sequencing, calculating how the 70 million fragments overlap is also a major computational challenge, and not least it is very challenging to determine how these can be put together to give the sequence of the source DNA string. This is essentially the fragment assembly problem, but the source string is no longer in the ballpark of 200 Kbp long – it is now 3 Gbp long. Celera solved this problem with the use of massive computing power: A total of 20,000 CPU hours were used for assembling the data. The final computations required 64 GB RAM [34, page 33].

Recently, a new method for solving the fragment assembly problem has been proposed [116]. It makes the problem of assembling the staggering number of reads more tractable. Traditionally (up through the 1990s), the fragment assembly problem was tackled in a three step process: First overlaps between reads were found. This information can be represented in a graph, where reads are nodes and overlaps between read are edges between nodes. Now, the task is to find a tour that visits all nodes once - that is, finding a Hamiltonian path except that we do not have to return to the starting node. The found path and its resulting *layout* (as it is called), is used to construct a consensus string, which is the final guess for the source string. The problem of finding a Hamiltonian path is unfortunately very hard to solve for large graphs (see section 2.1 for a discussion of the Hamiltonian Path problem and Eulerian Path problem). When doing whole genome sequencing (WGS), there are several million nodes. Instead of trying to find a Hamiltonian path in the overlap graph, one can — by cleverly restating the problem [116] — search for a Euler path, which is a path visiting all edges exactly once. This problem scales much better for many nodes (only polynomial growth) and the problem of finding the source DNA string becomes much more tractable.

4.4. Physical Mapping

At this point we shall return to the PM problem and formulate it more precisely. Firstly, it will be described how the data are constructed from experiments. Secondly, some algorithmic approaches for solving this problem are introduced, and finally their advantages, disadvantages, and limitations will be discussed.

4.4.1. Making Experimental Data

Various steps must be taken in order to get the data for PM. First, a clone library must be made, which may be done in the following manner [99], [72, pages 25–48], and [34, pages 23–28]:

1. Clone a DNA string s, to get N cloned strings: $s_1, s_2, \ldots, s_N$.
2. For each clone (s_i), break it down into fragments ($f_{i1}, f_{i2}, \ldots, f_{in_i}$) using a unique restriction enzyme, e_i.
3. Clone each fragment (f_{ij}) to obtain a sufficient number of clones for further experimentation.

At this point a so-called clone library has been produced. We use the term clone library, because it is a library where each entry in the library constitutes a collection of *clones* of fragment f_{ij}. Each collection of clones is physically stored in a test tube in liquid form.

To perform a PM experiment a collection of probes is needed as well. Probes are short, engineered DNA molecules that (as the strands of DNA molecules do) will bind to their complementary sequence. Naturally, the nucleotide sequence of a probe is known.

The next step is to perform a hybridisation experiment: For each clone and each probe, it is investigated whether they hybridise to each other. A clone and a probe will hybridise, if the clone contains the complementary sequence of the probe. The hybridisation experiment can be carried out using a microarray. This device is essentially a tiny plate with many tiny depressions on it. With modern robot technology, it is possible to place a tiny amount of liquid in each depression quickly and reliably. The idea is that each probe is put into its own depression. Then a clone solvent is "washed" over the plate, where it enters each depression and mixes with the probes. The particular probes that have a complementary sequence in the clone will bind to the clone. Because the probes have been treated in a special way, a hybridising probe and clone pair will cause the liquid to change its colour. This procedure reveals, for a every single probe, whether the probe hybridises to the clone being washed over the microarray. The process of washing a clone over the microarray of probes can be repeated for every clone. The final result is full knowledge of hybridisations between every clone and every probe.

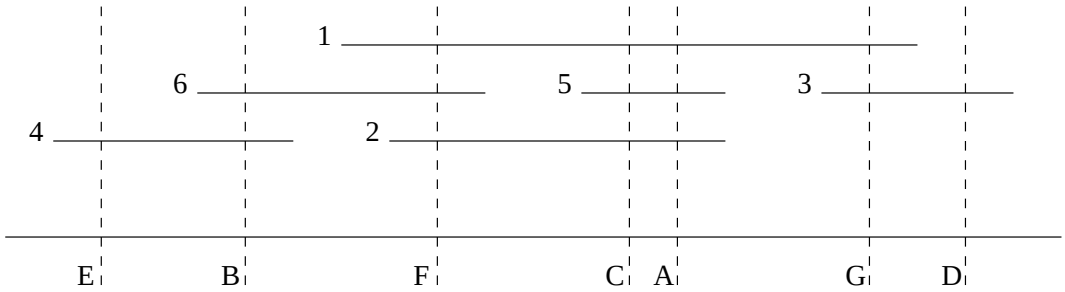


Figure 4.2.: Schematic drawing of a strand of DNA (bottom-most horizontal line), six clones (horizontal lines), and seven probes (indicated by dotted vertical lines).

	A	B	C	D	E	F	G
1	1		1			1	1
2	1		1			1	
3				1			1
4		1			1		
5	1		1				
6		1				1	

	E	B	F	C	A	G	D
1			1	1	1	1	
2			1	1	1		
3						1	1
4	1	1					
5				1	1		
6		1	1				

	E	B	F	C	A	G	D
4	1	1					
6		1	1				
1			1	1	1	1	
2			1	1	1		
5				1	1		
3						1	1

Figure 4.3.: Clone-probe hybridisation matrices. To the left: Unordered. In the middle: Probes have been ordered. To the right: Clones have been ordered as well.

A schematic drawing of a strand of DNA, six clones, and seven probes can be seen in figure 4.2. For example it can be seen that probe G hybridised to clone 1 and 3, and that clone 1 hybridised to probes F, C, A, and G.

The hybridisation data may be represented in a hybridisation matrix. Let the number of clones be m and the number of probes be n. The hybridisation matrix then has m rows and n columns. The matrices shown in figure 4.3 correspond to the DNA strand, the probes, and clones in figure 4.2.

4.4.2. Physical Mapping Problem Definition

The goal of PM is to determine the order of the clones (as they appear on the DNA string). This is equivalent to finding a permutation of the rows in the hybridisation matrix. However, it is standard practise to find the permutation of the columns (i.e. the probes) instead. The number of probes is often smaller than the number of clones, which makes the number of possible solutions much lower when finding probe permutations. A probe permutation can easily be converted into a clone permutation. This is shown in figure 4.3. The probe ordering (in the

middle) immediately leads to the correct clone ordering, simply by sorting the rows after the first occurrence of a 1.

The hybridisation matrix has a certain property when the columns are permuted according to the order in which the probes appear on the DNA string: In each row, all 1s (indexes indicating a hybridisation between the clone and a probe) will appear in one unbroken sequence with no 0s between them (indexes indicating no hybridisation). Such a matrix is said to have the *consecutive ones property* (C1P). The reason for the correspondence between the right ordering and the C1P is as follows: Let c be a clone and p_1, p_2, and p_3 be probes, where p_1 appears before p_2, and p_2 before p_3 on the DNA string. If c hybridises to p_1 and to p_3, then c must hybridise to all other probes between p_1 and p_3, because clone c is a continuous stretch of DNA. Hence, it must hybridise to p_2 as well, since p_2 is located between p_1 and p_3. Figure 4.2 may illustrate this fact by letting c be clone 2, p_1 be probe F, p_2 be probe C, and p_3 be probe A.

The problem of PM boils down to finding a column permutation of the hybridisation matrix that has the C1P. The problem was addressed by Booth and Lueker in 1976 [18]. They constructed an algorithm that can find all orderings that result in a matrix with the C1P. The algorithm runs in linear time of the number of columns. If no orderings result in a matrix with the C1P, the algorithm gives no useful answer.

4.4.3. Solving Real PM Instances

The fact that the method of Booth and Lueker is incapable of dealing with instances that cannot be permuted to obtain the C1P is the core reason for wanting to apply heuristics to PM. Because of experimental errors (and other factors that we shall return to later on), many hybridisation matrices have errors, and consequently they cannot be permuted to obtain the C1P. However, in most cases the property can almost be obtained, with the errors present in one or two positions in each row, for example a 0 dividing a long series of 1s, or a single 1 surrounded but 0s.

For a trained human observer, it is possible to recognise a near optimal permutation, because the positive evidence of the correctness of the given permutation outweighs the uncertainty caused by the errors scattered around in the matrix. The observer weeds out the experimental errors (see figure 4.4 for an example). Summing up, the experimental errors prevent us from using the fast exact method. A method is needed, which uses the information in the hybridisation matrix (even if it contains errors) to find a likely, near-optimal column permutation. Furthermore, what makes the task more complicated is that there might be many optimal solution orderings. In other words, what is needed is a heuristic that can search for a good ordering.

Given the above analysis, one may now realise that two major components are needed to find approximate solutions to the PM problem heuristically:

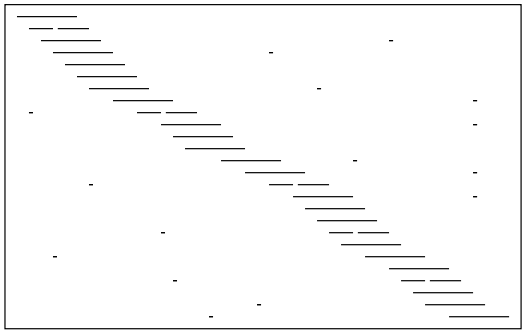


Figure 4.4.: A schematic hybridisation matrix with ordered clones and probes and some false positives and false negatives scattered around in the matrix. In spite of the errors it seems obvious that the ordering is most likely correct.

- A scoring function that scores matrices according to the likelihood that the column ordering reflects the true probe order. Ideally, the true ordering should receive the best score, and orderings increasingly different from the true ordering should receive increasingly worse scores.
- An optimisation algorithm that searches the space of permutations in order to return a column ordering which results in a matrix with a good (optimal) score.

Error Sources

Before going into detail about the two above points, we shall briefly go through some of the reasons why hybridisation matrices are contaminated with errors. These are:

- Non-uniqueness of probes: Although probes are unique sequences in the DNA string in the vast majority of cases, they may occur more than once.
- False positives: The experiments may indicate hybridisation between a probe and clone when, in fact, the clone does not contain the probe.
- False negatives: The experiments may indicate no hybridisation between a probe and clone when, in fact, the clone does contain the probe.
- Chimeric clones: In the construction of the clone library, it may happen that, two unrelated fragments join to create one fragment when the enzymes fragment the DNA string. This cannot be detected, and the result is an artificial and discontinuous fragment that nevertheless appears as one original continuous segment.

In all cases, the consequences will be that the hybridisation matrix for correct ordering of probes does probably not have the C1P. Further, it is also very likely that no other permutation yields a matrix with the C1P. Not all errors are possible to detect, but approaches have been taken to discover those than can be found. Harley [59] devised a topology graph of the data with the property that if data were coherent, the graph was without branches, but if parts of the data were contradicting, the graph contained branches.

4.4.4. Scoring Functions

As mentioned earlier, it would be desirable to have a measure for assessing the quality of a hybridisation matrix in general in those cases when no C1P ordering exists. When such a measure is used with a heuristic optimisation algorithm, it is often referred to as a *fitness function* or a *scoring function* (see section 2.2.3 and definitions therein), and we shall use these term interchangeably throughout the chapter.

Fragment Counting Functions

A number of fitness functions for PM have been proposed. One class of fitness functions is based on counting the number of fragments in the hybridisation matrix. A fragment is a horizontal, unbroken series of 1s. Among the suggested fitness functions in this class are:

- *total number of fragments*
- *number of split clones* (number of rows with two or more fragments)
- *maximal number of fragments in any row*

All three scoring functions have at least two nice properties: (1) They give minimal fitness value for a matrix with the C1P, and (2) the introduction of errors increases the score (except when false positives join two fragments or when false negatives remove a fragment). The former property satisfies the requirement mentioned in section 4.4.2 above (namely that the true ordering should receive the best score), which is necessary if the problem is to be successfully optimised using a fitness function. The latter property indicates that it may be possible to optimise the problem, also in the case of scenarios with many errors.

A further scoring function, which takes the sizes of gaps between fragments into account, exists. This function is called *inner-fragment gap length* and is defined as the sum of the lengths of all gaps, where a gap is a number of consecutive 0s surrounded by 1s. All the fitness functions mentioned above can be found in [53]. In table 4.5 we have listed the fitness values for all fitness functions taken on

Fitness function	Value (4.3 left)	Value (4.3 right)
Total number of fragments	14	6
Number of split clones	6	0
Maximal number of fragments in any row	3	1
Inner-fragment gap length	14	0

Figure 4.5.: Fitness values for the left and right matrices from figure 4.3

the left and right matrices from figure 4.3 (middle and right matrices give same values).

Just as the first three measures might fail to score error-prone matrices ideally, so the inner-fragment gap length measure has a weakness, namely that it punishes large gaps more than small gaps. Thus, a single false positive placed at a strategically bad place, may change the scores in such a way that a search algorithm would never consider any solution in the neighbourhood of the optimal solution.

Maximum Posterior Probability Function

In an attempt to overcome this problem, more sophisticated measures have been designed. One of them is a *maximum posterior probability* based method [3]. This method measures the probability that a hybridisation matrix given by a column permutation is actually the correct one. A fundamental part of this method is a dynamic programming (DP) algorithm that for each row determines a minimal cost of removing experimental errors, thereby turning the row into an error free version. More specifically, a cost for changing a 1 onto a 0 and vice versa can be defined. Furthermore, there is another cost for having a chimeric clone in a row. The DP approach finds a set of indexes to change in each row such that the cost is minimal. This results in an estimation for each row of the position(s) of the interval(s) of 1s, as well as an estimation of whether or not the row is chimeric. These estimates can be collected to get an computationally determined false positives rate, false negative rate, and chimeric rate for the matrix. These values can be compared with rates given to the algorithm (rates that are believed to be the true ones), and a score for the probability of the matrix is computed, based on the degree of agreement of the two sets of error rates. If a hybridisation matrix with an incorrect permutation of the columns is given to this measure, it will result in an estimate of the error rates that is far from the given (believed) rates. Therefore the score will be poor, which correlates with the fact that the permutation was incorrect.

4.4.5. Heuristic Optimisation Algorithms for PM

Given that we have a fitness function, we need an algorithm for optimising it. Every algorithm that is capable of searching the space of permutations is a possible candidate. Specifically, evolutionary algorithms (EAs) and simulated annealing (SA) are candidates, but also local search (LS) and other variants of it are possible (see chapter 2 for an overview). Below some of the suggested general purpose heuristics for PM will briefly be summarised:

General Heuristics for Any Fitness Function

The Random Cost algorithm [161] is essentially a LS algorithm. As usual in LS, an interim solution is maintained – in the present case a permutation. This permutation is iteratively modified by performing a reversal of a randomly chosen interval of the permutation. The criterion for accepting a modification is the following: Initially, 100 random reversals are performed on a randomly initialised permutation. The maximal change of fitness function in the 100 reversals $\max(\Delta D)$ is recorded, and used to define a stochastic and uniform random variable U on the interval $[-\max(\Delta D), \max(\Delta D)]$. In each iteration, a reversal will be accepted if $\Delta D + U \leq 0$, whereafter the range of U is decreased by 10^{-6}. The algorithm terminates when the size of the range of U becomes 0, or when no improvement has been made for 400000 iterations. The random cost algorithm has been applied on the PM problem in several studies (see e.g. [39, 161]).

Microcanonical annealing [16] is based on LS, and uses the same method for finding new solutions, namely reversal of an interval of the permutation. It iteratively tries to improve the solution for $MAXCOUNT$ iterations, but if no improvement has been recorded within the last K iterations, it exits immediately. A matrix E is initialised with each index set to E_{max}. The algorithm accepts a reversal of the interval $[p, q]$, if the change in fitness function ΔV when applying this reversal is below zero (that is, an improvement) or if $\Delta V \leq E_{pq}$. If, either way, the reversal is accepted, E_{pq} is updated by setting $E_{pq} = E_{qp} = E_{pq} - \Delta V$. At the end of each iteration every index in E is multiplied by a variable *factor*. Typical settings for the algorithm are $E_{max} = 0.5; factor = 0.5; K = 3; MAXCOUNT = 100 \times$ length of the permutation. Microcanonical annealing was used in the application ODS2 for creation of PMs [58].

Both the random cost algorithm and microcanonical annealing are LS methods that incorporate a central aspect from SA, name the decreasing probability of accepting a new solution that decreases the fitness. The effect is a LS that starts out by being random and ends up by being a strict hill-climber (HC). Also the standard version of SA [81] has been used for PM, for instance in [32] and [3]. We refer to section 2.4.1 and [81] for a description of the well-known SA algorithm.

Heuristics for Specific Fitness Functions

Instead of applying some general optimisation heuristic to optimise a fitness measure, one may try to exploit the definition of the fitness function to design a specific heuristic. One could for instance imagine building a solution in a stepwise manner. Below some approaches that have been designed for specific fitness functions will briefly be discussed for the sake of broadness in the description of the PM field.

The fitness function that counts the total number of fragments is equivalent to the vector version of the travelling salesman problem (vTSP) [53]. Thus, it is possible to solve the PM problem by converting it into a TSP problem and then apply some TSP heuristic to the converted instance. The conversion from hybridisation matrix to a TSP graph can be carried out as follows: For each column, make a node in the graph. For each pair of columns, add an edge going from the two corresponding nodes and set its weight equal to the hamming distance between the two columns. For instance, the hamming distance between column F and G in the left matrix in figure 4.3 would be 3. Before doing the conversion, a column of 0s should be inserted in the matrix to avoid punishing differences between the first and the last column. The distance of a found tour will be two times the *total number of fragments* fitness function evaluated on the corresponding hybridisation matrix.

Alizadeh et al. [3] made an investigation of the PM problem, where among other approaches, the TSP formulation was tested. A LS based heuristic [174] for TSP was used. The results were good, with the exception of matrices with high false negative error rates. Likewise, Greenberg and Istrail tried to find good permutations using the TSP formulation [53], but did not achieve impressive results using Christofides' maximum matching algorithm [24]. Instead they found that a greedy TSP gave better results.

In the same paper, Greenberg and Istrail also implemented two heuristics for optimising the *inner-fragment gap length* and *maximal number of fragments in any row*. The method which was based on *inner fragment gap length* performed reasonably well in cases with false negatives only, but worse than the TSP based method in other cases. The method based on *maximal number of fragments in any row* (the Cycle basis algorithm; see [53, page 40–41]), performed poorly.

4.5. Research Contribution to PM

Physical mapping (PM) is an interesting problem in many respects: It is not only valuable from the point of view of a biologist as a step in sequencing projects, but it is also interesting as a computational problem for a computer scientist: First, it is difficult in the sense that it is most unlikely that there is an efficient algorithm that finds the optimal solution in the presence of errors – it is an NP-

complete problem [51]. Second, one can make a comparison of different heuristic algorithms on the problem. Observing the efficiency of the tested methods on the PM problem may increase the knowledge of the tested methods. Third, we may investigate the significance of different scoring functions. The relative performance of the measures will indirectly be an indicator of the validity of the different models for PM that the measures represent.

In the paper *Physical Mapping using Simulated Annealing and Evolutionary Algorithms* included in appendix D, three optimisation methods are investigated: EA, SA, and LS. They were all tested on simulated PM data, with varying error rates – including error free data. All experiments were carried out with many different fitness measures (three were reported). In this way it was possible to observe which algorithm performed the best for PM and which measures gave the best results.

It was observed that there was a very poor correlation between neighbouring solutions in the search space. Consequently, it became hard to follow a gradient for the search algorithms and the conditions of the search will be reminiscent of looking for a needle in a haystack. The degree of correlation in the PM problem is determined by two factors: (1) The chosen neighbourhood operator of the optimisation algorithm that defines what a neighbour is, and, of course, (2) the scoring function that defines the measure of correlation. The fitness function *total number of fragments* gave one of the best correlations, whereas others such as *inner-fragment gap length* correlated neighbouring solutions very badly on data with errors. Regarding point (1), it may be noted that the neighbourhood-size is vast, due to of the many possible solutions that can be made from a given permutation. This fact alone implies that it is unlikely that every two neighbours will be well-correlated. However, it was experimentally determined that using smaller neighbourhoods gave poorer results.

As regards the significance of fitness functions, I found that the measure *total number of fragments* seemed to be better than the *maximum posterior measure*. This was also true in the error prone cases, in which the *maximum posterior measure* was designed to be efficient. The *inner fragment gap length* was poor at handling errors, because large gaps were punished too severely, and this fitness function gave far worse results than the two others. In fact, the variation between fitness functions was so outspoken that one could wonder whether the choice of algorithm was less important than the choice of fitness function. Surprisingly, the answer was that picking the right fitness function was more important than using the best heuristic.

There has not been much reflection on the measures used in previous PM studies; especially not in studies dealing with real-world mapping problems. Neither have the used measures been directly compared to other measures. Exceptions to this are the paper by Greenberg and Istrail [53] and Alizadeh et. al. [3]. However, none of the studies compared the measures directly. Instead, different measures were used as starting points to create totally different heuristics, whose efficien-

cies were then compared. Such an approach does not directly tell which measure is the better, since the algorithms are incomparable.

In an effort to explain why the optimal ordering was often overlooked by the algorithms, I carried out experiments to test whether the algorithms were responsible for the suboptimal results, or whether they were due to the error rates. Furthermore, even in the error free case, the optimal ordering of the PM problem is often ambiguous. This clearly implies that it is not possible to find the original ordering in every case. Through a set of experiments, a quantitative measure of this degree of ambiguity was obtained, and the significance it had in relation to the sub-optimal results. Based on these findings, it was possible to conclude that the algorithms performed almost as well as possible given the many errors and the ambiguity of the problem. In many cases the fitness score was optimised fully, but the optimal solution (the true ordering of the probes) simply did not correlate with the fitness function. For more details regarding these observations, see the paper in appendix D, where measures for assessing the quality of an ordering are defined.

The main conclusions were:

- It is very important to use a fitness function whose values correlate with the quality of an ordering. This is more important than trying to improve (or create) an optimisation heuristic.
- The *total number of fragments* measure was better than the *maximum posterior* measure, which was better than the *inner fragment gap length* measure.
- The EA and SA perform equally well. LS is almost a good as SA, but is consistently marginally outperformed by SA. However, when SA outperforms the EA, so does LS. These findings could suggest that a whole range of other problems previously solved successfully by EAs, may be solved just as efficiently by simple LS based methods as with EAs.
- In conclusion, all algorithms generally do well. It is hard to improve much on the results taking the inherent ambiguity of the problem and the effects of errors into account.
- Since the main reasons for the sub-optimal results were inappropriate fitness functions and error prone data, it was suggested that future research on PM should focus on the development of better fitness functions and production of better data through improved experimental procedures and possibly error screening algorithms.

Chapter 5

Structural Comparison

In this chapter, we give an introduction to structural comparison. We first describe and define the problem, whereafter we define measures for similarity. We then describe and analyse various approaches to searching for similarities, followed by reviews of a number of published methods. Finally, we present methods to assess the significance of any found similarities.

Structural comparison is the comparison of the three dimensional structures of proteins. Clearly, we would like such a comparison to reveal the most significant similarities that can be found between the structures. To achieve this, one must consider a strategy in searching for the similarities, and one must also find a measure to quantify the extent of the found similarities. Unfortunately, these are really hard problems, as there is not a best (or most efficient) way to make a comparison (i.e., to search for the similarities) nor to evaluate the answer (i.e., to assess the similarities) [37].

5.1. Problem Definition and Measures for Similarity

How can similarity between protein structures be measured? Recall from chapter 3 that a protein is a sequence of residues. Thus, overall similarity can most naturally be defined as a set of similarities between pairs of residues. The found similarity can be visualised by placing the structures on top of each other (by superposing the structures) with the corresponding residues as near as possible to each other (see figure 5.1).

5.1.1. Alignment Size and RMSD

If the concept of similarity as introduced above is broken down, there are two criteria that determine the overall similarity:

- The number of similar/matching pairs, and
- The proximity of similar/matching pairs when superposed.

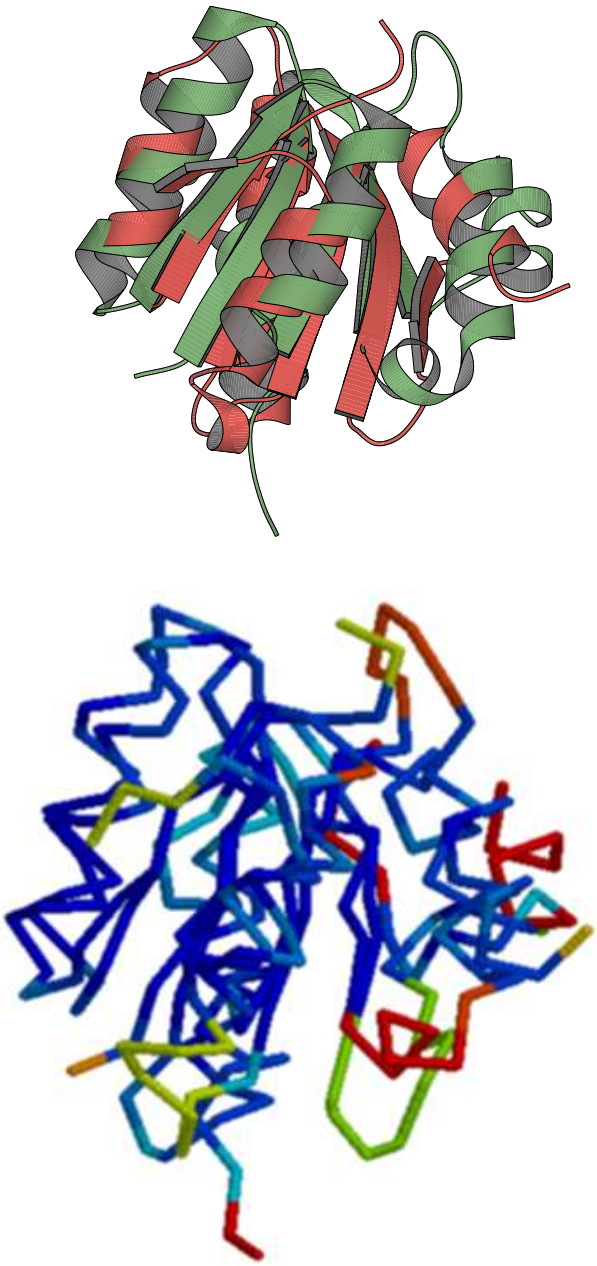


Figure 5.1.: Superposition of the SCOP domains (scop d1jbwa1) and (scop d1mb3a_). Top: The structures are shown using the *cartoon* style. Bottom: The structures are shown using the *backbone* style, where the backbone is drawn with lines between consecutive C_α atoms. Blue indicates residues that fit well, whereas yellow and red are unmatched C_α atoms.

This understanding of protein structure comparison is natural, simple, and widely used as a basic framework for comparison. This approach is called an element-based description of structures, and is in contrast with space-based descriptions, where properties for sub-spaces (typically grid-cells) are stored (such as number of residues in the cell, SSE content, etc.). The individual elements (atom, residues, SSEs, etc.) are hereafter ignored [38, page 184]. The space-based description is also known as the statistical description, but is not widespread in pairwise structural comparison.

Returning to the element-based description, it can be more formally defined. Let two protein structures be given, A and B, and suppose they consist of the elements

$$a_1, a_2, \ldots, a_m$$

and

$$b_1, b_2, \ldots, b_n$$

These elements will most often be residues that moreover simply are represented by their C_α atoms. Structural similarity can then be defined as an *equivalence*, $E(A, B)$, which is *a set of pairs of elements*. Using the notation introduced, an equivalence can be defined as [38, page 187]

$$E(A, B) = (a_{i_1}, b_{j_1}), (a_{i_2}, b_{j_2}), \ldots, (a_{i_N}, b_{j_N}).$$

It is obvious that an equivalence (or alignment) can be judged by the two criteria mentioned in the beginning of this section.

The first criterion, the number of matching pairs, is called the *alignment size* and is throughout this thesis denoted by symbol N. The name, alignment, stems from the fact that the similar pairs implicitly define an alignment of the protein amino acid sequences.

The second criterion, the proximity of the matching pairs, is quantified by a measure called $RMSD$, which is the square root of the mean of the squared distances between similar residues when superposed, thus the name $RMSD$. (The last letter, D, in $RMSD$ stands for deviation.) The distance between two residues is usually defined as the distance between their C_α atoms. Thus, the $RMSD$ can be defined as

$$RMSD = \sqrt{\frac{1}{N} \sum_{k=1}^{N} |a_{i_k} - b_{j_k}|^2}$$

where $|x - y|$ is the Euclidean 3D distance between the points x and y.

5.1.2. Measures Based on Alignment Size and RMSD

Having defined N and $RMSD$ that – as concluded above – are essential in describing overall similarity, one may consider how to use these measures to find the most significant similarities. Using either alone is not particularly attractive, as $RMSD$ alone would favour small alignments with low $RMSD$ (the extreme being $N = 1$ and $RMSD = 0$). Using N would lead to including all residues at the expense of a high $RMSD$. Neither extreme gives much information about possible similarities between structures. Clearly, it is quite unrealistic to use either of those measures alone in an overall evaluation of similarity. Still, the two extremes illustrate the problems that arise when a compound measure is biased towards or away from either of the two measures, N and $RMSD$.

It should be clear that the goals of low $RMSD$ and high N are conflicting, and that there is a trade off between the two. One possible idea for combining the two measures could be to make a composite score, $S = (N, RMSD)$. In this scheme the solution with similarities $S_1 = (N_1, RMSD_1)$ is better than $S_2 = (N_2, RMSD_2)$ if and only if $N_1 > N_2$ and $RMSD_1 < RMSD_2$. However, if $N_1 > N_2$ and $RMSD_1 > RMSD_2$ (or $N_1 < N_2$ and $RMSD_1 < RMSD_2$), neither solutions can be said to be better – or to *dominate* the other. When returning the best similarities from an algorithm using this measure, one could return the non-dominated similarities; that is, similarities for which there is no better similarity as defined above. Although this combined measure nicely illustrates the trade off barrier for N and $RMSD$ (known as the *Pareto front* in multi objective optimisation), this approach can be problematic since there is no complete ordering of similarities; there will be many similarities for which it is impossible to decide which is the better one.

Combined Measures

An alternative to the above approach is to construct a single measure that is simply a function of N and $RMSD$. There are several examples of using this idea in the various studies. It has been noted that the optimal $RMSD$ is proportional to the square root of N [38]. Thus, simple (e.g. linear) functions of N and $RMSD$ are not suitable, as they cannot correctly assess the relative quality of very small and large alignments.

An obvious measure, given observed correlation between N and $RMSD$, is the similarity score

$$\frac{RMSD}{\sqrt{N}}$$

which should be minimised for finding the most significant similarities [38]. In another measure suggested by Lu [93], N is "normalised" by the sizes of the structures being compared:

$$\frac{RMSD}{(\frac{N}{avg(|A|,|B|)})^{1.5}}$$

where $|A|$ and $|B|$ are the size of the structures. As noted by the author, the *minimum* function can be used instead of the *average* function when comparing two proteins with very different sizes; the *average* function is best in classification of structures based on similarity scores.

The $RMSD$ measure severely penalises residue pairs with weak proximity, whereas well aligned pairs do not contribute much to the $RMSD$ (that should be minimised). This can be problematic, because a few residue pairs with weak proximity can obscure the fact that large segments might be very well alignable. Alternatively, one can also design a score to be maximised that rewards well aligned pairs, and where poorly aligned pairs do not contribute much. The measure suggested in relation to the method GaFit [88] does exactly that; it is defined as

$$\sum_{i=1}^{N} c^2 - d_i^2$$

where $d_i = |a_i - b_i|$ (the distance between the C_α atoms of the i'th aligned pair) and c is a threshold value (default $c = 5$). This measure should be maximised for significant similarities, and does not "punish" for badly aligned residues, but "rewards" well aligned residues.

Many similarity scores have employed the even simpler (than $RMSD$) Euclidean distance between residues. A similar measure to the one presented above was presented in [142]. The rewarding of well-aligned residues is implemented slightly differently. In addition, it includes an additional term for for punishing gaps:

$$\Big(\sum_{i=1}^{N} \frac{20}{1 + d_i/5}\Big) - 10 \cdot (gapsA + gapsB)$$

where $gapsA$ and $gapsB$ are the number of gaps in the alignment of the structures.

5.1.3. Distance Matrix Based Measures

In the treatment of measures for structure similarity so far, we have implicitly assumed that two structures with superposed coordinate sets were given or calculated in some way or the other. It is, however, possible to find and assess similarities without dealing with the concept of superposition and without rotating or updating the original coordinate sets as it is necessary to do for $RMSD$ based measures. The use of distance matrices (see section 3.3) enables this.

If two structures have very similar shapes, then it should be intuitively clear

that distances between similar residues are similar as well. For example, suppose that two structures, A and B, are very similar, and that residue 5 from both structures are equivalenced, and likewise for residue 7. Then the distance from residue 5 to 7 in structure A, will be very similar to the distance from residue 5 to 7 in structure B. Therefore, one can search for similarities between structures by comparing their distance matrices rather than superposing the structures. We will return to the details of this approach later in the chapter.

When working on distance matrices, the RMSD measure is therefore slightly differently defined, and is called the *distance RMSD* (as opposed to the *coordinate RMSD* dealt with so far). Assume that $(a_1, a_2, \ldots, a_N)$ and $(b_1, b_2, \ldots, b_N)$ are the matching C_α atoms from the first and second proteins respectively, and further assume that the internal distances have been stored in the $N \times N$ distance matrices A and B. Then the distance $RMSD$ is defined as:

$$RMSD = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}(A_{ij} - B_{ij})^2}$$

where A_{ij} is the Euclidean distance between residues a_i and a_j in protein A and similarly for protein B. It has been shown for sets of residues (with given equivalences) that the minimal coordinate $RMSD$ and the distance $RMSD$ are linearly related [26].

The DALI method [67] by Holm and Sander, which will be described in detail in section 5.5.2, searches for similar sub-matrices in the distance matrices for the two proteins being compared. The similarity of two such matrices is given by the *additive similarity score*:

$$S = \sum_{i=1}^{N}\sum_{j=1}^{N}\phi(i,j)$$

where $\phi(i,j)$ is a similarity measure between residues i and j as they are related in structure A and structure B respectively. The similarity measure, ϕ, was implemented in two variants.

The *rigid similarity score* is defined as:

$$\phi^R(i,j) = \theta^R - |d_{ij}^A - d_{ij}^B|$$

where R stands for rigid, d_{ij}^A is the Euclidean distance between residues i and j in protein A (similarly for d_{ij}^B), and θ^R is a threshold with default value θ^R =1.5 Å. The rigid similarity score uses the principle explained above that similar structures implies similar intra distances. Only intra distances that deviate less than θ^R are counted in the score.

However, the amount of this deviation must compared to absolute distances between residue pairs. (E.g., with an origin at 0, the numbers 1 and 2 might

be said to differ significantly, whereas (using the same origin) 100 and 101 are relatively much more alike.) This aspect plus the scaling down of long distance contributions was incorporated into the *elastic similarity score*, thus taking the form:

$$\phi^E(i,j) = \left(\theta^E - \frac{|d_{ij}^A - d_{ij}^B|}{d_{ij}^*}\right) w(d_{ij}^*) \ , \ i \neq j$$

where $d_{ij}^* = (d_{ij}^A + d_{ij}^B)/2$, θ^E is the similarity threshold and has default value $\theta^E = 0.2$, and $w(t)$ is an envelope function given by $w(t) = exp(-t^2/\alpha^2), \alpha = 20$. For $i = j$, it is defined that $\phi^E(i,j) = \theta^E$.

The distance and coordinate RMSD are similar in the sense that are pretty useless without knowing the size of the alignment, N. Thus, measures that use intra distances (rather than inter distances as GaFit and Structal do) while making a trade off between alignment size and alignment fit, have also been made; for instance the SSAP method assesses the found similarities through alignment size and distance $RMSD$ [150].

5.1.4. Other Formulations

In addition to the $RMSD$ based formulations and the distance matrix based one, structural comparison has also been stated as an optimisation problem over a contact map [20], as the comparison of normalised inter-residue vectors [22], or as the minimisation of the Area-C^α distance [40]. However, no single measure or formulation has proved superior in reflecting structural similarity as defined by accepted classifications such as SCOP and CATH, and no fast, reliable, and convergent method is yet available [82]

5.2. Beneficial Restrictions in Similarity Search

Having defined alignment size N, *coordinate RMSD*, *distance RMSD*, and some methods to combine these quantities into useful measures, it would seem that the problem of structural comparison simply amounts to picking one of the measures and optimising it. Although finding the similarities (at this point and until section 5.5) still remains an open and uninvestigated question, the end-goal would seem clear enough: Just optimise a "good" measure of similarity.

Unfortunately, there are two pitfalls that the above considerations do not address, and we shall discuss each of those below.

5.2.1. Consecutive Residues

The first issue is connected to the fact that when using a simple measure based on $RMSD$ and alignment size, for randomly chosen structures, the most pronounced

similarities will often contain a lot of "random" backbone matches that consist of only one or two consecutive matching residues. Such similarities do clearly not reflect true evolutionary similarity. Of course, from a computational point of view, it is always interesting to optimise a score function fully; however, such alignments are uninteresting from a biological point of view, as the short matching regions with all probability do not represent any common evolutionary history.

To circumvent this artifact, it is common practise in many structural comparison methods to require that any aligned residue must part of a consecutive stretch of aligned residues of a certain length (a value of three is a much used setting, but of course this varies and is dependent on the problem instance and the application). This is an issue for coordinate $RMSD$ as well as distance $RMSD$ based methods. It is obviously possible to construct measures that take this into account and punish single matching residues (e.g. the Structal measure [48] (see page 92) indirectly does this through its gap penalty), but judging from the majority of implemented solutions, the constraint on consecutive residues seems to be the most practical solution. To see the effects of gradually requiring more and more consecutive residues in the alignment, we refer to section C.3.

5.2.2. Maximum Alignment Distance

The second issue, which in particular concerns the $RMSD$ based methods, is the fact the $RMSD$ is measuring dissimilarity; often, however, it is the maximal similarity that is being sought. Consequently, the $RMSD$ can obscure some potential similarities, if one is not careful about which weakly matching residue pairs to include in the alignment. A simple, but efficient, solution is to allow only residue pairs closer than a certain cut-off distance in the alignment. This keeps the $RMSD$ low. Another similar idea is to explicitly allow only a certain maximum RMSD, and to dynamically limit the selection of residues such that the $RMSD$ is kept below this threshold.

5.2.3. Alignment Size Used as Measure

When using e.g. the constraint on maximum distance between aligned residues, the simple measure of alignment size, N, becomes an good measure of similarity (see e.g. [2]). The maximum distance parameter determines which type of similarities that will be found. If set to e.g. 1 Å, then only relatively short similar fragments will be found (perhaps longer fragments for very similar structures, being almost identical in sequence as well). However, if set higher to e.g. 3 Å, it should be possible for a good search method to identify major structurally similar motifs, such as similar architecture (spatial arrangement of SSEs).

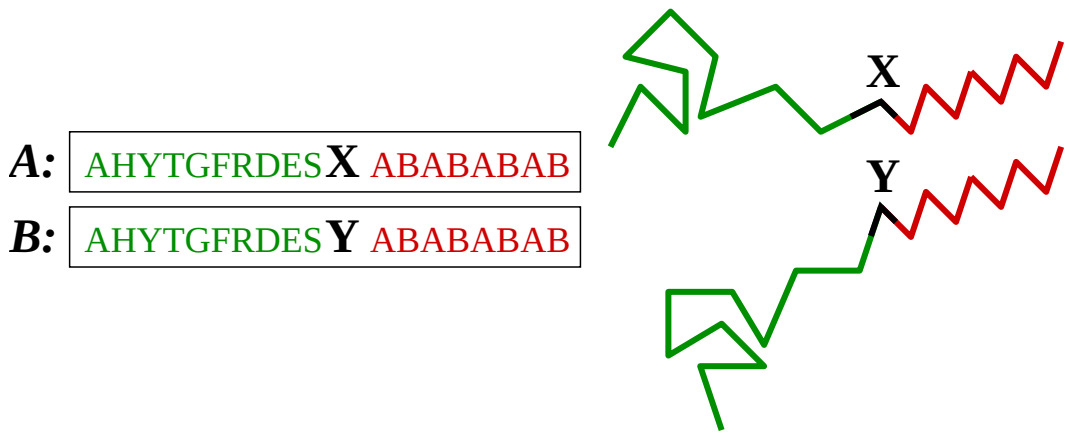


Figure 5.2.: In the alignment of sequences A and B, it is obviously always optimal to align the eight last letters with each other regardless of the contents or alignment of the previous letters. However, when aligning the hypothetical and simplified 2D structures of the sequences, the best structural alignment will not necessarily include a matching of the last eight residues. In this example, there is a longer (and perfect) match at the N-terminal on 11 residues, which excludes matching of the C-terminal, because the backbone is rigid.

5.3. Structural Comparison Versus Sequence Comparison

Structural comparison is different from – and harder to solve than [135] – sequence alignment in several respects.

When making a sequence alignment it is possible to define a similarity matrix, where each entry gives the score for aligning two residues of given types. Furthermore, one can select a scheme for gap-penalties, including gap-opening penalty and gap-extension penalty. Having decided on this scoring scheme, it is straightforward to find the best alignment with dynamic programming (DP) in quadratic time.

Conversely, in structural alignment it is not possible to use DP in the same way. The problem is that residue similarity is not based on residue type, but on residue proximity in 3D space. This proximity is dependent on the superposition of the structures, and this superposition is dependent on the chosen alignment of previous residues. Thus, there is a circularity that prevents the use of DP: Structural residue similarity is dynamic and is not known until the problem has been solved. This is illustrated in figure 5.2.

Methods such as SSAP [150] and SAP [146] (see section 5.5.2) cleverly try to avoid this by calculating a value for the structural similarity of two given residues

that is indeed independent of the superposition. Although elegant, this heuristic does however not guarantee an optimal solution to the structural alignment problem.

5.4. Approaches to Structural Comparison

When the superposition is fixed, it is possible to apply DP to find the best alignment. This is because the structural residue similarities – due to the given superposition – now are known and thus constant. Given an initial alignment or superposition, this leads to a possible approach of cyclic, iterative updating of alignment and superposition. Indeed, this is reflected by the iterative nature of many structural comparison methods: Given a superposition, an alignment can be calculated with DP. Using this alignment, an optimal superposition (which is most likely different from the previous one) can be calculated (see section A.2). Using this superposition, we can return to the beginning and calculate a new alignment. This process can be continued until the user is content with the result, or until the alignments or superpositions converge, or until some maximum number of allowed iterations has been reached.

In terms of complexity and hardness, the structural alignment problem is known to be NP-hard [84] for the Structal similarity score [48] and, therefore, we do not know if it can be solved in polynomial time. If it is possible, no one knows how to do it, and most computer scientists believe that it is impossible to solve NP-hard problems in polynomial time. However, this conjecture remains to be proved. Conversely, as touched upon in the previous section, the pairwise sequence alignment problem is not NP-hard: It can be solved using standard DP in time proportional to the product of the length of the input sequences; i.e. polynomial in the input size.

5.4.1. Means of Modifying the Problem

To solve structural alignment in polynomial time, we have to accept the risk of returning sub-optimal solutions. Kolodny [84] has designed an approximation algorithm, which finds a solution at maximal ϵ units worse than optimum of the given score function and that runs in time $O(n^{10}/\epsilon^6)$. This time-complexity (although polynomial) is really poor, and the algorithm is (as discussed in the paper) more of theoretical interest than of actual practical use.

Instead of finding an approximate solution to the real problem, we might rephrase the problem and subsequently find the optimal solution to the changed problem. This approach was discussed from a more general perspective in section 2.1. The trick is to rephrase the problem to one that we are still interested in solving. This can be done by restricting the possible solutions we search for and thus restrict ourselves to find. For instance, in the case of FASE, we never in-

vestigate solutions that align α-helix residues with β-strand residues. Such (and other) constraints enable a faster search for optimal alignment. Of course, this could mean that we overlook some solutions that have larger alignment size and lower RMSD (or that are better using some other measure). But still, these solution will probably not be of much interest, if the "improved" scores are obtained by alignment of α-helices with β-strands — such alignments do not make sense in an evolutionary context.

5.5. Methods for Finding Similarities

In recent years, structural comparison methods have undergone a development that is driven by the ambition (and necessity) of being able to discover continual more distant similarities [148]. The focus has expanded from simple extensions of a superposition routine that can only identify highly similar fragments, to methods that can find remote similarities [110, chapter 6]. Also, the field has been diversifying and specialising into a number of areas: Fast and approximate scanning of large databases of structures represented by SSEs, search for binding sites by looking at the individual side chain atoms and conformations, methods that emphasise discovery strength rather than speed and can find non-sequential similarities. This variety of focus areas shows that it is necessary specialise the search to a certain degree. In the following section we will illustrate why.

5.5.1. Search Domain – Alignments or Superpositions?

In section 5.1 it was stated, the structural similarity problem can be stated as finding a set of equivalent residue pairs that match well; in other words, similarity is defined by an alignment. Yet, when searching for similarities, there are other options than searching the space of sequence alignments. As mentioned, given a superposition, one can find an optimal alignment. Thus, an alternative approach than searching for alignments is to search the space of combinations of rotations and translations. If desired, a final result can then subsequently be converted to an alignment.

There are some methods that use this strategy. These include the MINAREA method by Falicov [40] and the approximation method by Kolodny [84]. Conversely, there are also methods that only find alignments, and do not calculate one single superposition. These include DALI [67] and SSAP [150]. Finally, there is a large group of methods that use both concepts in the search for similarities; usually this is implemented in the iterative fashion described in section 5.4. Examples are TOP [93], STRUCTAL [48], FASE [158], and others.

For this last group, two operations are common: Deriving an alignment from a superposition, vice versa. Both tasks are quite simple, and there are standard methods for solving them well and fast. However, if a sequential constraint is

not imposed on the alignment, it is not possible to use DP (nor the related Soap Film Measure [40] introduced with the MINAREA method).

In this situation, approaches like nearest neighbour finding can be used to obtain an alignment. A whole range of different heuristics have been presented in many different methods for fine-tuning or improving existing alignments based on the superposition they impose. For instance, FASE uses the DP in submatrices to improve alignments based on a superposition. Similarly, SSM [86] uses a number of heuristics for changing and optimising an alignment derived from a superposition.

5.5.2. Review of Methods

SSAP and SAP

The SSAP method [150] introduced *double dynamic programming* (DDP). The name indicates that SSAP employs DP on two levels, a low level and a high level. The low level DP is done once for each pair of residues (i, j) (a quadratic number of times) to find out how similar the geometric environments of the two residues are. Specifically, the two proteins are superposed with residue i and j on top of each other in 3D space, and such that the coordinate frames, defined by the adjacent residues, $(i-1, i, i+1)$ and $(j-1, j, j+1)$, coincide. Such a superposition gives rise to a matrix, M_{ij}, where entry (m, n) is the distance from residue m in A to n in B (normalised with the function $g(x) = \exp(-x^2/\sigma^2)$). A similarity factor for the local environments of m and n was also incorporated into the entry. The matrix M_{ij} is subjected to DP to find the best path through the matrix. This procedure unveils how similar residues i and j are based on their superposed environments, and moreover which residues that are corresponding in their environments.

The optimal paths found with the above low level DP procedure are summed into a high level matrix, where each summed entry now is a heuristic measure for the structural similarity of any two residues – averaged over the many low level alignments of the two structures. The global best structural alignment is ultimately extracted by use of DP on the high level matrix, hence the name DDP.

For structures of equal size, the time complexity of SSAP is the fourth power of the sizes, because a quadratic number of low level alignments (each of quadratic time complexity) must be made.

This problem was fixed in the successor SAP [146], where an improved scoring scheme was introduced as well. In SAP, the low level calculation is not done for all pairs (i, j), but only for the K most promising pairs as judged from residue exposure, local structure, and sequence similarity. This compound information is stored in the so-called bias matrix, Q. In this way, the time complexity is improved. The global alignment is now found in an iterative manner. The initial alignment is calculated based on the initial K contributions. Then in each itera-

tion K is increased, the best path is incorporated into Q, which is also normalised at each iteration, and Q is incorporated into the low level matrices in particular in early iterations. For full details, the reader is referred to the original papers [146] and [145].

DALI

In the DALI method [67] an alignment of two protein structures is found by comparison of their distance matrices. DALI uses the elastic similarity measure from section 5.1.3 to assess similarity of distance matrices. The overall idea is this: In the first phase, all 6×6 sized sub-matrices, called hexapeptides, from the distance matrix of the first protein are extracted and compared with the 6×6 matrices of the other protein. Similar hexapeptides are stored in a *pair list*.

In the second phase, elements from the pair list are assembled into larger consistent sets of pairs. This is done using a kind of parallel simulated annealing (SA). A simple implementation of SA only has one mode, in which a neighbour is found and possibly accepted as new current solution. In DALI, however, a trimming mode is present as well. In the standard expansion mode, all elements from the pair list that overlaps with the current alignment are tested for inclusion in the alignment; the alignment is enlarged by choosing the element that gives the best overall similarity score (this is the steepest ascent local search strategy known from hill-climbing). In the trimming mode, tetrapeptides that contribute negatively to the overall score are removed.

In addition to this overall outline of the method, a wide range of heuristics are used to make the computation tractable and terminate faster. These includes:

- The protein is divided into segments (basically defined by SSEs), and each segment is represented by one hexapeptide.
- The pair list is built by starting with low intra-distance hexapeptides while restricting the size of the pair list. Essentially, this omits comparison of long range contacts; poorly matching hexapeptides are also disregarded.
- Throughout the SA optimisation, similar alignments (from the parallel optimisations) are merged, and poor alignments are removed to speed up the process.

DALI is only available as a web-service, but a simpler version, called DaliLite [66], can be downloaded for local use. In contrast to DALI, DaliLite is not capable of finding non-sequential alignments (see section 5.6.1).

TOP

The TOP method [93] is perhaps the one with most similarities to FASE (see section 5.6.4 for a detailed description of FASE). As FASE, TOP uses SSEs in

its search and has two phases. There are, however, some important differences: TOP uses a mutually nearest requirement in alignment, which might prevent it from discovering some particular types of alignments. This is discussed in section 5.6.4 and illustrated in figure 5.7 on page 90. In FASE, the motivation to be able to find such alignments — in other words: to be able to find good alignments on a consistent basis — was the reason for introducing the second phase, where the alignments are refined using DP (see section 5.6.4).

Moreover, TOP uses a threshold for the percentage of matched SSEs to decide further searching; this is a rather sensitive measure and is difficult to set (according to the paper describing TOP [93]); if set too high, similarities are missed, and if set too low, the method runtime will dramatically increase. Finally, in the initial phase where SSEs are compared, all pairs from the first structure are compared to all pairs from the second. This gives a time-complexity of $O(n^4)$, where n is the number of SSEs, which is typically proportional to the size of the protein and thus proportional to the overall problem size. In FASE, this is avoided by only aligning each SSE from the first structure in a fixed number of ways with every other SSE from the second structure, giving a time-complexity of $O(n^2)$. For a more elaborate discussion of the time-complexity of FASE, we refer to section C.1.

MASS

MASS [35] and [36] can find non-sequential alignments, and has resemblances with the present method and the TOP approach. As in TOP, the MASS method searches for pairs of matching SSEs from the two structures. This gives the same $O(n^4)$ time-complexity as TOP has. Regarding the residue alignment, MASS uses a linear-time method [10], which only makes use of the simplistic nearest neighbour scheme as TOP does. In conclusion, MASS therefore has the same weaknesses as TOP (time-complexity and alignment method).

LOCK

LOCK [137] uses a three step strategy. First, it performs a local SSE superposition. An initial superposition of the two proteins is found using DP to align the vectors representing the SSEs. Then, a greedy nearest neighbour method is used to minimise the RMSD between the C_α atoms of the two structures. Finally, the best sequential core of aligned C_α atoms is found, and the RMSD is minimised between the C_α atoms, yielding the final alignment. LOCK has been reimplemented and redesigned in a new method called FoldMiner [132].

PRISM

The PrISM system [170, 171, 172], which uses the so called PSD measure, has some resemblances to LOCK [137]. PrISM also uses a two step approach to

alignment: It first makes a SSE alignment using DP, and then it makes a residue alignment also using DP. However, since this alignment is based on DP, the method cannot find non-sequential alignments.

CE

The goal of the combinatorial extension (CE) algorithm [134] is to find good local alignments of the structures of the two proteins. First, an alignment fragment pair (AFP) is selected. An AFP is a continuous segment from one protein aligned with a continuous from the other protein (no gaps are allowed). CE then iteratively tries to extend the AFP with another AFP. It may be appended directly to the old one without any gaps in either protein, or there may be gaps in one of the proteins, but not in both. The algorithm stops when no further extensions are possible.

STRUCTAL

STRUCTAL [47, 48] is based on iterative DP to align inter-molecular distances. The pairwise alignment score in each entry of the matrix is inversely proportional to distance between the two atoms. An iterative approach of alternating DP and superposition is used. Each iterative process is seeded with one out of six relative transformations of the structures to avoid getting trapped in a local optimum. As all standard applications of DP, the method enforces sequentiality in its solutions. Actually, the initial search phase of FASE is a generalisation of this approach; instead of having only six starting points, it takes the full step and aims at finding any interesting SSE arrangements by aligning all SSEs from structure A against all SSEs from structure B.

MINAREA

The MINAREA method [40] spans a triangulated surface between the backbones (represented by the C_α atoms) of the two proteins being compared. This surface can be thought of as a film of soap that is being spanned between the structures. For a given 3D superposition of the proteins, the minimal surface is found using DP. The minimal surface also implies a correspondence between residues. In this way the structural alignment problem can be attacked as a simple optimisation problem, where the search space is the space of all possible combinations of rotations and translations of the proteins, and the fitness function is the area of the soap film, also known as the *soap film measure*. The top-level of the algorithm uses different greedy LS heuristics for searching the space of superpositions.

KENOBI

KENOBI (or K2) [144] uses a genetic algorithm (GA) to search the space of alignments. It first finds an initial alignment of SSEs, and then the alignment is subsequently fine-tuned on the residue level using a GA again. The runtime of the algorithm is somewhat high [144], but K2 was later superseded by a version using simulated annealing, K2SA, that supposedly sped up the search [143].

MSTA

MSTA [109] uses a technique called geometric hashing to find the largest common sub-structure of a set of proteins. The method finds the sub-structures and their alignments simultaneously. MSTA is not limited to align proteins, but is capable of aligning any set of molecules.

URMS Method

Chew [22] describes a method for detecting common substructures based on the Unit vector Root Mean Square (URMS) measure. The method finds contiguous alignments, which are simpler and more restricted than the standard sequence alignment (which further allows for insertions and deletions into a sequence). Thus, no non-sequentiality is allowed.

VAST

VAST [49] only aligns SSEs. Each SSE is represented as a vector (see section 3.7.1). It then finds all possible pairs of vectors from the two structures that are similar. Two pairs of vectors are similar, if they are related in approximately similar ways in 3D (e.g. relative position and orientation). Using a maximal clique detection algorithm, VAST finds a maximal subset of similar SSEs. The overall alignment score is based on the number of similar SSEs between the two structures. In a later version of VAST, the alignment score was changed to reflect the number of aligned residues.

PROTEP

PROTEP [55] creates a graph of each protein. In this graph, a node corresponds to a SSE, and edges correspond to relations between SSEs; edges are labelled with information about spatial and angular relationships between the structural elements represented by the nodes. The algorithm finds a maximal common subgraph for the two proteins under comparison, which corresponds to two sets of SSEs (one from each protein) that are spatially similar.

GRATH

GRATH [60] is an extension of PROTEP. GRATH uses a more advanced scoring scheme; not only does it count the number of SSEs aligned, but it also takes the sizes of the aligned SSEs into account. The algorithm creates a graph where each node n_i corresponds to a possible alignment of two SSEs s_{i1} and s_{i2} – one from each protein. There is an edge between two nodes n_i and n_j if the relative placement of s_{i1} in relation to s_{i2} is the same (within some given margin) as the relative placement of s_{j1} in relation to s_{j2}. That is, there is an edge, if there are no conflicts in aligning s_{i1} with s_{i2} and s_{j1} with s_{j2}. Finding an alignment amounts to finding a maximal set of aligned SSEs with no conflicts. This is equivalent to finding a maximal clique in the described graph.

Other methods

Recently a method called SSM was described which can find non-sequential alignments [86]. It begins with SSE matching followed by several heuristics to make the residue alignment.

The SARF method [2] also finds sets of compatible SSEs using a heuristic for maximal clique discovery. Subsequently these sets are refined and extended on the residue level. SARF is also able to find non-sequential alignments.

PRIDE [21] is a statistical similarity assessment program for protein structures. PRIDE, however, only returns a value that indicates the degree of similarity; it does not make (or return to the user) an alignment or superposition or in any other way describe, where the possible similarities have been found. It is therefore a similarity assessment program, but not an alignment nor superposition program.

MAMMOTH [111] is a method that is specialised in multiple structure alignment. MAMMOTH also requires sequential alignments in pairwise and multiple comparisons.

Other methods than the ones mentioned above have been published, but mentioning them all here is beyond the scope of this review. Instead, I refer to some valuable reviews on the field [136, 82], and chapter 6 on protein structure comparison in [110].

5.6. A Description of FASE

In the following section, I will describe FASE (Flexible structural Alignment from Secondary structure Elements), which I have developed through the latter part of my Ph.D. study. FASE is a method, which compares, aligns, and superposes two protein structures by searching for maximal structural similarities.

The algorithm has been implemented as a C++ program and is available from `http://www.daimi.au.dk/~jve/FASE`. Below, we will describe and discuss the

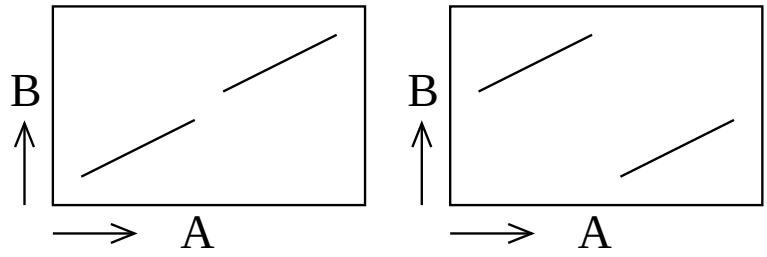


Figure 5.3.: Two schematic alignments are shown. The left alignment possesses the sequentiality property, whereas the right does not, because increasing structure A indices does not correspond to increasing structure B indices.

concepts behind FASE; more technical issues about the practical usage are dealt with in the appendix B and may be found on the above web address as well.

5.6.1. Sequentiality and Non-sequentiality

Methods for structural comparison can be distinguished by their ability to discover non-sequential alignments. For a sequential structural alignment it holds that if any two aligned residue pairs, (r_i^A, r_j^B) and (r_k^A, r_l^B), are chosen from the alignment and $i < k$, then $j < l$ as well. In figure 5.3, two alignments have been plotted in a dot-plot like fashion to illustrate the concept of sequentiality and non-sequentiality. Some proteins are clearly related by a non-sequential alignment, which makes the ability to discover such alignments an attractive feature of any alignment algorithm designed to find similarities on a fold/architecture level.

The ability to discover non-sequential similarities, however, comes at a price. More potential solutions must be searched, and this obviously slows down the method. For instance, it is impossible – even for a fixed pair of structures – to try to find an alignment with DP based algorithms, since DP not only requires later decisions to be independent of previous ones, but *also* requires the order of the elements to be aligned to be known. This is not the case, if non-sequentiality is allowed. This highlights the unavoidable trade-off of speed versus discovery strength that any structural comparison method must deal with. Choices relating to this trade-off are made on many places in the design process, and the ability to find non-sequential similarities is just one of them.

5.6.2. Main Idea and Assumptions

The motivation behind FASE was to be able to detect any significant structural similarity between two proteins. This may sound as a trivial statement, but this

has, in fact, not been the goal of all structural comparison programs. Because of efficiency considerations, many shortcomings in the ability to discovery solutions have typically been accepted by the designers.

Many structural comparison algorithms only find similarities between SSEs, because it makes the algorithm much faster to neglect the residue level altogether. This is the case for VAST [49], PROTEP [55], and GRATH [60].

A popular method to gain speed is the use of a simplified and reduced model of the space of the possible alignments; this speeds up the comparison process, but the result is that important similarities may be missed. The inability to find non-sequential alignments is perhaps the most typical one (see below and section 5.6.1). Of the programs reviewed in section 5.5.2, only TOP, MASS, LOCK, KENOBI (K2), MSTA, SSM, and SARF can find non-sequential similarities as FASE can. The other programs: SSAP, SAP, DALI, CE, STRUCTAL, MINAREA, URMS method, VAST, PROTEP, GRATH, PRIDE, MAMMOTH, and PRISM can only find sequential similarities.

In designing FASE, we have tried to make as few assumptions as possible that could hinder the discovery of similarities. However, some assumptions and restrictions of the search obviously must be made, as structural comparison in its most general and pure form is a very hard problem that would otherwise take very long time to solve reasonably well (see section 5.4). These restrictions can be described through a characteristic of the similarity-space in which we search for similarities. The nature of this space, and the degree to which it facilitates discovery of all interesting similarities, will be discussed below.

The FASE search-space is spanned by superpositions of pairs of structures that have at least one pair of well-aligned SSEs in common. We assume that an optimal superposition must have this property. The sought for optimality is required primarily in relation to the overall alignment of the structures (known as the architecture level in CATH) and not on the residue level. Although FASE puts emphasis on (and uses roughly fifty percent CPU time on) the residue alignment, fine tuning of residue correspondences is an issue that is much more ambiguous and debatable than the overall type of alignment discussed above. The success or failure regarding residue alignment will have no relation to the general principle of the method, but only to the particular heuristics used for fine-tuning the alignment. The general idea of FASE should be understood on a global architecture level scale.

To sum up: It is hopeless to find the "best" alignment (whatever best might mean) on the residue level. There are two reasons for this: 1) There are too many solutions and they cannot be searched efficiently. 2) Even if an optimal solution was found, it could be discussed, if another slightly different one was perhaps better depending on the preferred score function. This is simply because there is no exact and perfect model of protein evolution (see chapter 6).

Therefore, the approach taken, instead, was this: Cleverly search (or enumerate) the space of SSE alignments (which is 2-3 magnitudes smaller than the

residue alignment space). Then, make a quick check of the SSE alignments by including residues in the comparison. In a second phase, perform a more thorough comparison of the best and most promising SSE alignments.

5.6.3. Overall Description

The FASE approach has of two main stages: 1) An initial search phase in which the alignment of SSEs are starting-points for finding tentative alignments using a nearest neighbour alignment scheme. 2) A refinement phase where the solutions are subjected to iterative dynamic programming (DP) in unaligned areas. The method applies DP to several sub-alignments in parallel. One single best solution is returned, but FASE also provides a ranked list of the best non-redundant solutions. The quality of the solutions is determined by a consensus scoring method that includes several measures from known methods (and some new ones) in order to improve consistency and robustness in its similarity assessment (for details, see section 5.6.4 below).

5.6.4. Method

In the following, we will describe the FASE algorithm in greater detail.

The Search Phase

Every SSE pair (one from each structure) of the same type formed the basis for a set of tentative solutions. Let a SSE pair $s = (s_A, s_B)$ be given, let $|s_A|$ and $|s_B|$ be the number of residues in each of the SSEs, and let x_A and x_B be the residue numbers of the first residue in each of them. For each SSE pair, $2 \cdot n_{off} - 1$ alignments were made by aligning $l = 4$ consecutive residues from the structures starting with residues f_A and f_B (see figure 5.4). Residue $f_A = x_A + (|s_A| + 1)/2 - l/2$ (the first aligned residue from structure A) was fixed in all alignments, while f_B was variable, taking on the values $f_B = x_B + (|s_B| + 1)/2 - l/2 + i$, for $i = -n_{off} + 1, \ldots, n_{off} - 1$. In figure 5.4 it is shown how the middle four residues of s_A are aligned with four middle residues of s_B (the case: $i = 0$) together with the four other alignments being made by displacing the four residues from s_B with one and two residues to both sides. In this way we can quickly and efficiently cover all the possible alignments of the structures where s_A and s_B are aligned.

Using this strategy, a maximum of $|SSE_A| \cdot |SSE_B| \cdot (2 \cdot n_{off} - 1)$ initial alignments are made. Some are discarded due to non-matching SSEs or similarity to previous ones. For each of them, the following iterative procedure was performed in order to conservatively extend and stabilise the alignments:

The structures were superposed using a standard least squares method based on the current alignment [70]. Initially, the current alignment consisted just of $l = 4$ consecutive matching residues from each structure. By selecting the pairs of

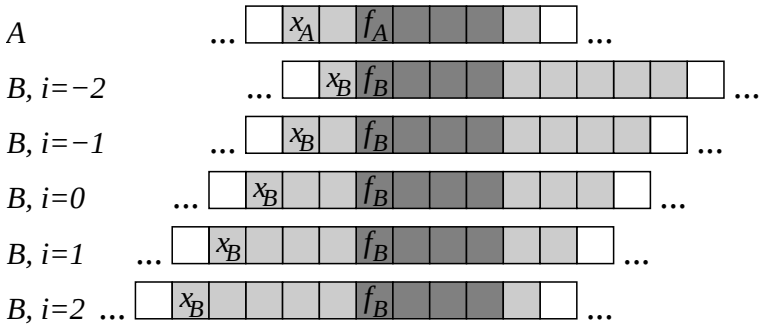


Figure 5.4.: Illustration of alignments tried for structure A and B for two given secondary structures, s_A and s_B. The remaining values are $n_{off} = 3, l = 4, |s_A| = 7, |s_B| = 10$.

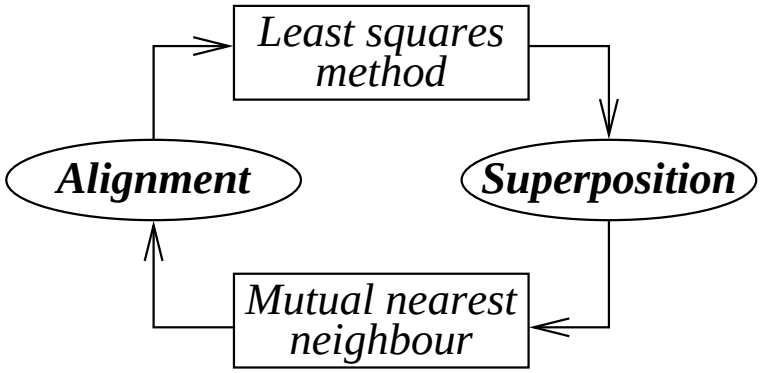


Figure 5.5.: Illustration of the alternation between alignment and superposition. An alignment leads to a superposition by a standard least squares minimisation methods, and a superposition leads to an alignment (in the search phase) by pairing of mutually nearest residues from the two structures.

residues from the two structures that were mutually nearest to each other under the superposition, a new alignment could be derived. Thus, we have gone from alignment to superposition, and from superposition to alignment (figure 5.5). Iteration was terminated when then alignments converged, more precisely when a cyclicity (of maximal length 10) was detected in the alignments. (The iterative process usually terminated naturally after 5-10 iterations, but an upper limit of 50 iterations was enforced.)

To avoid arriving at an alignment influenced by random residue pairings, in the i'th iteration any pairs not part of a continuous stretch of at least $\min(c, i)$ aligned residues were neglected (where the default value of c was 3).

Filtering Solutions for Refinement

A large number of tentative solutions was generated in the search phase, but only some of these were selected for further consideration as many were clearly unacceptable, perhaps matching only a single SSE pair with the remaining parts of the structures occupying different parts of space.

This selection was made by ranking the tentative solutions by the number of aligned residues (N) as first priority and by the root mean square deviation ($RMSD$) of the aligned residues as second priority. The resulting list was inspected starting with the top ranked solution. For a given value of N, only the six best scoring solutions (ranked by $RMSD$) were selected. The selection process terminated when solutions had less than 10 aligned residues, or a length less than 33% of the best solution.

Refinement Phase

An alignment found using the greedy and simple nearest neighbour method rule often includes some biologically unfavourable pairs. It is therefore not suitable for finding similarities between the backbones of protein structures. One explanation of this is that a good structural alignment may, in general, simply not be excepted to adhere fully to the nearest neighbour rule. This has also been illustrated in figure 5.7, where the task of aligning two similar sub-structures (that moreover are almost perfectly superposed in space) can be hard to solve for a mutually nearest neighbour method.

Although a wrong alignment might still give a fine superposition of the structures, the alignment found by a structural comparison algorithm is just as important as the superposition. For instance, the sequence similarity over the aligned residues is often used as a measure to assess homology (see section 6.3.3), and this value could be misleading, if the structures are misaligned. Therefore, a refinement step was applied to the tentative alignments to enable discovery of non mutually nearest residue pairs and, consequently, to get biologically more relevant alignments.

In short, the idea was to look at areas (pairs of sequences) of unaligned residues, and to check if it would make sense to align them. Thus, segments of residues falling between matched SSEs were aligned using DP (see section 5.6.4 for details). This approach, illustrated in figure 5.6, allows discovery of non-sequential alignments because the DP algorithm is only applied locally.

Let a tentative alignment be on the form $(s_1, \ldots, s_n)$, where s_i is a sub-alignment of length $l_i + 1$ of continuously selected residues, $(r^A_{p_i}, \ldots, r^A_{p_i} + l_i)$ from structure A and $(r^B_{q_i}, \ldots, r^B_{q_i} + l_i)$ from structure B. The sub-alignments define $(n+1)^2$ unaligned areas (at most) in the alignment matrix shown in figure 5.6. Each unaligned area a_{ij} was tested to see if the DP algorithm gave a positive (good) score that could indicate local structural similarity. If so, the

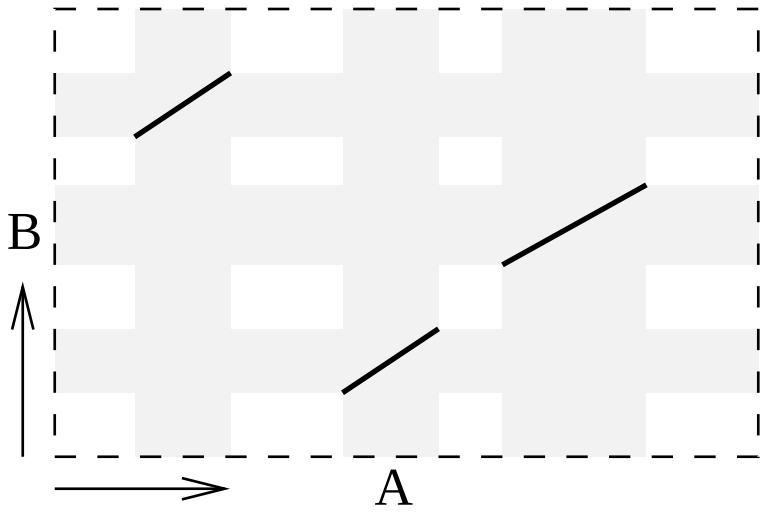


Figure 5.6.: This hypothetic tentative alignment has three sub-alignments. In the shaded regions no new pairs can be aligned because of existing sub-alignments. However, in every white, unaligned region the algorithm attempts to align the residues by DP. In this example, DP will be tried in $4 \cdot 4 = 16$ regions and accepted if the DP-score is good enough.

sub-alignment was added to the overall alignment. After performing DP for all sub-matrices, it was tested if the new pairings caused any alignment conflicts. Any conflicts were resolved by choosing the pair of residues with the lowest distance. Moreover, residue pairs that were further apart than a specified upper limit (d_{max}) were subsequently removed. In addition, pairs where one (or both) residue(s) was not part of a consecutive stretch of at least c_{min} residues, were also removed from the alignment (as in the search phase). Between two iterations, the structures were re-superposed (as in the search phase) in order to update the input to the next phase. Thus, the sub-alignment would change between iterations (and possibly merge or split) influencing the number of sub-regions where DP was applied.

The four step alignment strategy of 1) DP, 2) conflict resolution, 3) weeding by distance, and 4) constraint of consequentially, was repeated iteratively until alignment convergence using the same principles for deciding this as in the search phase.

Dynamic Programming

We used the standard DP algorithm [108] in the refinement phase to align residues in the sub-areas that were defined by the already continuously aligned residue segments. That means that every execution of the DP algorithm was always performed under a fixed relative transformation (or superposition), T, between

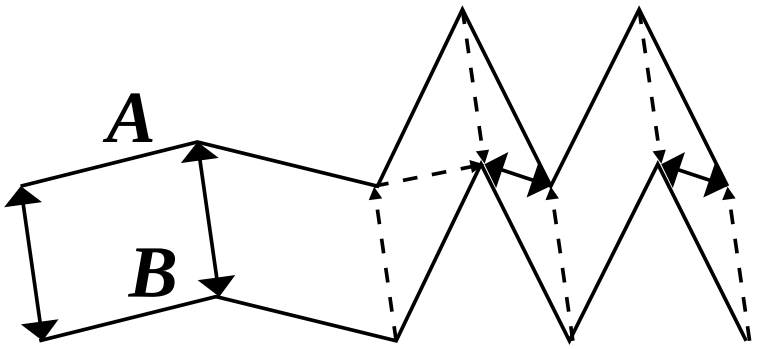


Figure 5.7.: Mutual nearest neighbours. An arrow from A to B indicates that the given residue in A has the given residue in B as its nearest neighbour. There are eight residues in each of the substructures, but only four pairs of mutually nearest neighbours (marked with fully drawn lines).

the structures. Entry (i, j) of the DP matrix M was defined by:

$$M[i,j] = e^{-\frac{d^2}{10^y}}$$

where $y = 1$, and d is a short notation for the distance between residue i in structure A and residue j in structure B under transformation T, $dist_T(r_i^A, r_j^B)$. However, if $d > d_{max}$, then $M[i,j]$ was set to 0.

DP with linear as well as with constant gap-penalty was tested. Linear gap-penalty was eventually preferred. One may note that a constant gap-penalty does not discriminate between small and large insertions. This might work well globally, because an optimal alignment on a global level might need to accommodate large insertions in order to reach a global optimum. However, we used DP locally to align parts of the structures, and our focus was therefore in a greedy way to align as many pairs as possible. Tests showed (as one would expect) that the linear penalty was best to find the type of sub-alignments needed.

Finding Nearest Neighbours

When finding mutually nearest neighbours during the search phase, the following task was performed a large number of times: Under a given superposition (transformations T_A and T_B of the structures A and B), for every residue in structure A, find its nearest neighbour in structure B (and vice versa). See figure 5.7 for an illustration.

Let r_A and r_B be the number of residues in structure A and B. It is straightforward to solve the task in time $O(r_A r_B)$ by simply calculating the distances between all $r_A r_B$ pairs. However, this was a core operation of the algorithm that had to be performed repeatedly, and therefore this gave an unsatisfactory

time-complexity, especially as the structures increased in size.

For simplicity, consider first a simple hash-table and a structure B. For each 1 Å cube in 3D, the hash-table stores all the residues from the structure that can be nearest neighbour to any point inside the given cube. Once calculated and initialised, such a data-structure can be used to find residue i in structure A's nearest neighbour (located in structure B) under the transformations T_A and T_B: Firstly, the transformation $T_B^{-1}T_A$ was applied to the original coordinates of residue i in structure A. This translates residue i from structure A into the original coordinate system for the residues from structure B. The residues associated with the 1 Å cube in the structure B hash-table (where the transformed residue "lands") may now be checked to find the one that is actually nearest to residue i from structure A.

This approach, which "costs" one transformation, a hash-table query, and then calculating c distances (typically less than 10), is considerably faster than calculating all r_A (or r_B) distances for the given residue – in particular when structure sizes increase. Even for an average sized structure, the use of the efficient data-structure gives a factor 5 speed-up compared to the simple approach (data not shown). Assuming momentarily that the two structures are of equal size n, this reduces the time-complexity of nearest neighbourhood finding from $O(n^2)$ to $O(cn)$, where c is the average number of residues in a 1 Å cube.

By decreasing the size of the cubes, the query speed of the hash-table can be improved (because c will decrease). However, this dramatically increases time required for building the hash-table: Every little 3D cube would require processing during creation of the table to find the residues to which it should point. It was realised that the space had to be structured and checked in a more efficient manner than using a standard hash-table. To harvest the advantages of a fine granularity hash-table while avoiding a durable initialisation, a hierarchical space-partitioning hash-table like data-structure — also known as a quad-tree [42] — was used. This is a special case of a normal hash-table and can be used as such in terms of insertions and queries. It is much faster to construct and uses less memory, while having the same query speed.

First, a sufficiently large bounding box was defined in 3D around the structure. It was then divided by a plane at the midpoint of its largest dimension. For instance, the box $C = ((1, 9), (12, 17), (-4, 3))$ would be split by the plane $x = 5$ along its x-dimension resulting in two new boxes: $C_1 = ((1, 5), (12, 17), (-4, 3))$ and $C_2 = ((5, 9), (12, 17), (-4, 3))$ respectively. Now, C_1 and C_2 could then recursively be divided. Since all residues by construction were inside the "root" box, they were all added as a possible nearest residue for a point in the root box and stored in the root box's near-list. Each time a box was divided (as with C above), the near-lists of its "children" were calculated by selecting a subset of the elements from the parent's near-list. The recursive division continued until the largest dimension of a box dropped below a fixed value (default was 2 Å), or when the near-list of the box only contained (a default value of) two points.

A query into this data-structure started at the root and continued to select the child box in which the query point was contained until the current box had no children. When found, the near-list was searched for the nearest neighbour.

Because of the hierarchical structure used to divide 3D-space, the initialisation time and space usage are much lower than a standard hash-table. The 3D-space is not uniformly divided and the granularity is naturally adapted to the residue density of the region. This saves space and time compared to a normal hash-table.

Scoring of Similarity

A consensus score function based on five individual measures was used to rank the final set of solutions internally. As some of these measures have already been used in previous work, one could reasonably suppose they possessed good similarity assessment capabilities. However, a consensus approach was used to add an extra level of robustness. The five measures were the following:

- **S1** First priority: Alignment size, N, in percent of the number of residues in the smallest protein. Second priority[1]: $RMSD$
- **S2** Alignment size and $RMSD$ based score[2]: $\frac{3}{100}N - RMSD$
- **S3** Structal [142] score function: $\left(\sum_{i=1}^{N} \frac{20}{1+d_i/5}\right) - 10 \cdot (gapsA + gapsB)$ (see e.g. page 71)
- **S4** GaFit [90] score function: $\sum_{i=1}^{N} c^2 - d_i^2$
- **S5** TOP [93] score function no. 1: $\frac{RMSD}{\left(\frac{N}{avg(|A|,|B|)}\right)^{1.5}}$

The solutions were ranked in turn by each of the five criteria and on each criterion their ranking was recorded. The final consensus score was the sum of the four best rankings (the worst ranking was discarded). The solutions were ranked by the consensus score and the list was available after completion of FASE. The solution with the best/lowest consensus score was returned as the optimal solution.

A Measure for Alignment Linearity

A measure for alignment linearity (resemblance to the identity alignment, or the "straightness" of the alignment) was defined. Let the residues of structure A be named $r_1^A, \ldots, r_m^A$ and the residues of structure B be named $r_1^B, \ldots, r_n^B$. An

[1] The second criterion was only used when ranking solutions with equal N.

[2] Tests showed that comparisons of randomised structures almost never reached scores where $N > 100/(3 \times RMSD)$, implying that $3/100N - RMSD > 0$.

alignment of A and B may be described as a list of residue-pairs. For instance an alignment of size 3 could be: $(r_1^A, r_2^B), (r_4^A, r_4^B), (r_7^A, r_8^B)$.

We defined a metric for the divergence of an alignment from linearity as the summed difference of the indices of each aligned pair. For example; in the alignment, $(r_1^A, r_1^B), (r_2^A, r_2^B), \ldots, (r_m^A, r_n^B)$, The sum (v) is: $0 + 0 + \ldots + (m - n)$. This difference was then normalised with the theoretical maximum divergence, $0.75 r_A^m r_B^n$. Finally the value v was modified to $1 - v$ such that the identity alignment would get 1, and the most deviant alignment would get 0.

A Measure for Circular Permutation

The result of a circular permutation (CP) on a protein is that the N- and C-terminal parts of the protein become exchanged [74]. If an alignment corresponds to a CP, it can be detected by scanning through the list of aligned residues and for each position, i, calculate the average of the structure B components to the left of i,

$$avg_L(i) = \frac{1}{i} \sum_{j=0}^{i-1} r^B_{alB(j)},$$

and to the right of i,

$$avg_R(i) = \frac{1}{N - i} \sum_{j=i}^{N} r^B_{alB(j)},$$

where $alB(j)$ refers to the residue number of the structure B component of the j'th aligned pair and N is the alignment size. If there is a CP, then there will be an i for which $avg_L - avg_R$ is positive, and the index where $avg_L - avg_R$ is maximal (i_{max}) will match the beginning of structure B (or similarly: (i_{max}) will match the break of structure A).

As we would like to measure the significance of the permutation, we also take into account the size of the two parts before and after the break. If, for example, structure A starts with matching the last five residues from structure B, whereafter A starts to match the beginning of structure B, then the CP cannot be claimed to be very significant. Thus if we define

$$rearrscore_A(i) = \frac{avg_L(i) - avg_R(i)}{N_B} \cdot \frac{i}{N_A} \cdot \frac{N - i}{N_A}$$

where N_A and N_B are the sizes of structures A and B, we can define that $rearrscore_A = max_i(rearrscore_A(i))$. The final score for the degree of CP takes the degree of the permutation into account as seen from both structures, and we define it as:

$$rearrscore = 4 \cdot (rearrscore_A + rearrscore_B)$$

The *rearrscore* has a theoretical maximum value of 1 (a perfect CP), and a

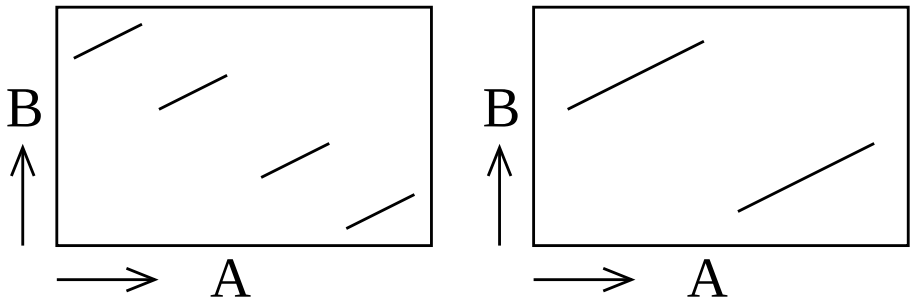


Figure 5.8.: Two schematic alignments are shown. The right alignment might be a real CP, whereas the left is definitely not, but still gets an equal CP score. However, using the measure for straightness on the left and right halves, it is possible to make distinction.

minimum value of -1 (corresponding to a fully sequential alignment). Values less than 0 are truncated to 0. Significant matches for CPs can, typically, be detected at values from 0.25-0.3 and above. The maximum value is 1 because $rearrscore_A$ (and $rearrscore_B$) has a maximum of 0.125, which can be seen, since its maximum value may be calculated as

$$\frac{0.5N_B}{N_B} \cdot \frac{0.5N_A}{N_A} \cdot \frac{0.5N_A}{N_A} = \frac{1}{8}.$$

This measure takes three factors info account: 1) The size of the alignment, 2) the partition of the permutation, and 3) the "pureness" of the permutation. The product $\frac{i}{N_A} \cdot \frac{N-i}{N_A}$ account for 1) and 2), whereas $\frac{avg_L(i)-avg_R(i)}{N_B}$ account for 3).

To further test the nature of the permutation, the "straightness" measure was applied locally to the two parts of the alignment as defined by the circularity. This enabled us to distinguish a real CP alignment (figure 5.8, right) from one that just gets the same CP score, but is not a CP (figure 5.8, left).

5.7. Assessment of Significance

To assess the significance of a general comparison between two objects, its score must be compared to what can be expected by chance. For a sequence alignment this is relatively unproblematic, because the concept of a random amino acid sequence is well-defined, and random sequences are also easy to generate. It is a more complex matter what should be understood by a random structure, since one — in addition to length and distribution of amino acids — must consider many aspects such as clash avoidance, packing density, secondary structure formation, and protein-like patterns [38, page 253].

5.7.1. Generating Random Data

A standard approach to assess the significance of a comparison is to calculate the probability that a comparison of random objects gets an equal or more significant score. In the context of structural comparison, the question then arises how to generate the random structures.

A fully random, but compact and non-clashing, walk might seem as a promising idea. One approach for generating random structures is to perform a self-avoiding, compact random walk. The code for such a procedure is given in [149]. The idea is to sequentially extend the C_α trace by adding a random point in the appropriate distance from the current C_α atom (typically 3.8 Å). Of course, no clashes are allowed between any two atoms; moreover, to ensure compactness, every generated atom must have a suitable number of contacts (neighbours) — that is, atoms in a specific distance range (typically 4.5 to 5.5 Å).

With this simple method some helical structures do emerge, and domain-like formations are also present. However, the chirality and the twist of the helices are not correct. More problematic is the fact that neither beta sheets nor beta strands emerge. Thus, it eventually turns out to be inappropriate; it does not possess SSE like properties, thus giving unrealistically low similarity scores when compared to real structures.

The next logical step in constraining the randomisation, would be to specialise the generation to structures such that they would belong to a certain protein class, fold, superfamily, or family (see section 3.8.2). The extreme is randomised structures based on a single original structure. The more constrained the generation is, obviously the more difficult it becomes to make many randomised structures.

Another method is to let the fold of the reversed backbone guide the generation of the new structure [149]. Moreover, it is possible to alter the connections between the SSEs, such that the fold is altered (but the architecture is per definition still intact).

5.7.2. Using Classified Data

An approach that avoids the generation of random structures is to use existing comparisons of classified structures as a background statistic. This of course requires a "gold standard" such as SCOP, and is therefore not an *ab initio* solution to the real problem; however, in practise this works fine with the large amount of classified data available.

The method allows us to derive an estimate for the significance (false positive rate) when classifying two structures as being homologous (see section 6.1.1). First, one makes a sufficient number of comparisons of non-homologous structures (e.g. from SCOP) to get a reliable frequency distribution. Based on this frequency distribution, one may set a threshold such that only an acceptable low

percentage (e.g. 1%) of non-homologous structure pairs will be classified wrongly as homologous; alternatively one can use this frequency distribution to assess the risk making a false positive when classifying a given pair as homologous. The above outlined method can be extended to making a frequency distribution for the homologous structures, which allows for an estimate of the false negative rate when classifying a given pair as non-homologous.

General Approach for Use of Classified Data

This leads to a general formulation of the above scheme: Given a comparison method, and a classification of objects of the input type, we wish to decide for every input pair, if it is related by some abstract relation. The comparison method outputs a score, but where should the threshold be, and with what certainty can a given score be used to make a classification?

First, make a sufficient number of comparisons of objects that are know to be related (in the classification), and similarly make many comparisons of objects that are known to be unrelated. This gives two frequency distributions. When classifying a new, unclassified pair, the frequency distributions can be used to assess the likelihood that the output of the comparison can correctly be interpreted a if the pair is related and unrelated, respectively. More specifically, the false negative and positive rates can be found using the frequency distributions when classifying a given, new pair.

This approach was used in the paper describing FASE (presented in appendix F). Here, a threshold was derived for classifying structures as belonging to the same fold by using a background statistic for the comparison of structures from different folds (according to SCOP).

Chapter 6

Protein Evolution and Homology Detection

In this chapter, we will introduce protein evolution and some basic ideas and theories relating to this subject. The pool of modern proteins have evolved and continue to evolve as a result of the changes in the underlying DNA that partially determine their structures. Understanding the concepts of protein evolution is therefore necessary when dealing with comparison and homology assessment of protein structures.

Homology assessment, and detection of likely evolutionary steps in protein evolution, is central when evaluating the plausibility of a possible related pair of structures. In particular, homology assessment is central in the journal paper, in which FASE is introduced (appendix F), and also in the paper on detection and evaluation of newly found CPs (appendix G). To make a good judgement in these cases, one must take aspects from protein evolution into account, and the background given in this chapter, is motivated by this fact.

Unfortunately, it is not possible to know with absolute certainty how evolution has progressed for every current protein; thus for certain pairs of structures, we have great difficulties in deciding whether they are evolutionary related or not. Many criteria can be used to reach an answer, when speculating about homology and origin. When sequence (and perhaps even structural similarity) is very weak, other kinds of similarities, such as functional similarities, can be looked upon. In addition, more advanced schemes can be constructed for answering these questions. Nevertheless, one must always remember that when answering questions regarding protein evolution and homology of proteins, there cannot be given any definite answers; this would require a complete knowledge of the evolution of proteins – a knowledge that we do not have [148, pages 293–294].

6.1. Protein Evolution

6.1.1. Some Definitions of Evolutionary Relatedness

Homologous proteins are proteins with the same ancestor — that is, they share the same original protein coding gene. The respective genes of the current proteins might have diverged significantly over time because of random drift after speciation or for other reasons, such as relaxed functional constraints after duplication [114].

Orthologous proteins are proteins whose protein coding genes are very similar and typically have the same function, but where the proteins reside in different organisms. The genes and proteins have diverged after a speciation event, and have accumulated differences as a result of random drift in each organism; still, they probably perform the same function [114].

Paralogous proteins arise as a result of duplicated genes in the same organism. Duplications can lead to extensive drift and to large families of genes with separate but related functions. When a duplication happens, a new backup gene is essentially created, and thus one of the genes can start to diverge, because the other gene still codes for the protein that performs a possibly vital function [96].

Divergent evolution is when two proteins share a common predecessor, but have changed in each their direction during evolution, and are now only partially similar. Thus, homologous proteins (including orthologous and paralogous proteins) are results of divergent evolution. Conversely, *convergent evolution* is when two proteins starting from each their unrelated predecessor evolve to attain a similar function or structural feature. Thus, at first glance they might look related, but they are in fact not [41, pages 26–30].

6.1.2. Theories of Evolution

Monophyletic

It is an accepted assumption that all present day genes and proteins are descendants of one common ancestor — more specifically, a few ancestral genes living in early life forms [1, page 453]. This hypothesis is reflected in most research concerning protein evolution, and also in the classifications of protein structures such as SCOP [106] and CATH [115] that both are hierarchical in their organisation of all known domains into a tree. In other words, there is a broad belief that proteins are monophyletic in their origin. Also the definitions in section 6.1.1 above take the monophyletic assumption for granted.

Polyphyletic

However, analysis of protein structures have lead some researchers to believe that modern proteins may not be monophyletic [96]. Instead, proteins were supposed

to have evolved from antecedent domain segments (ADSs), short polypeptides. The ancient protein domains did thus consist of several chains of ADSs, they were oligomeric (as opposed to modern ones that are single chained). This alternative hypothesis is based upon the observation that many structurally and functionally similar proteins exist in seemingly different folds, which seems to contradict the theory of a monophyletic origin.

It has also been suggested, in a similar train of thought, that modern protein domains evolved from ancient peptides [138]. Supposedly, these peptides were not able to fold, but acted as cofactors in an RNA world. However, in the course of evolution they would assemble to foldable units (similar to the suggestion above), and later they would fuse into the longer chains that modern protein domains consist of. The motivation for this RNA based theory is the observation that a majority of proteins consist of a limited and recurring set of super secondary structure elements (SSEs).

Inconsistencies

The common observation behind both of the above polyphyletic hypotheses is that a monophyletic, tree-based classification of protein structures leads to quite a number of inconsistencies: The are many examples of homologous structures that have diverged widely in sequence and structure – near the limits of detectability [121]. Also, there are examples of proteins with very similar structure, sharing the same fold, that however do not seem to be evolutionary related. To summarise: A common fold does not imply a common ancestor, and a common ancestor does not imply a common fold [56].

Of course, these inconsistencies might simply reflect that evolution have been too long and eventful for us to trace it back to a possible origin. It is beginning to be clear that sequence and structure are subjected to very extensive mechanisms of alteration during evolution (duplications, circular permutations [92], domain insertions, swapping, secondary structure exchange [127], domain stealing and relocation [63]; see below). The conclusion drawn from these observations is that just as proteins consist of homologous and non-homologous domains, so can domains be understood to consist of homologous and non-homologous super SSEs [138].

These uncertainties regarding protein evolution are good to have in mind when dealing with homology or evolutionary history, as they remind of the natural limitations in our abilities to answer these questions.

6.1.3. Major Events in Protein Evolution

Duplication, Insertion, and Swapping

Gene duplication is one of the most prevalent and important mechanisms in the evolution of proteins [114, page 31]. Often, a duplication results in a protein with several near-identical versions of the same gene — and thus production of proteins with the same fold. Also, as mentioned above, duplications can lead to paralogous genes that have been freed from their functional constraints. Different types of domain duplications are described and reviewed in [63]. The most frequently encountered protein fold, the $\beta\alpha_8$-barrel, has been shown to have evolved by tandem duplication and fusion from an ancestral half-barrel [64].

There are also many examples of domain insertions in the evolution of protein domains [127]. Such insertions can be detected by finding sequence or structural similarities to independently existing domains for either the inserted domain or the "parent" domain. Curiously enough, many inserted domains appear to be circularly permuted relative to the independent homologous domains [56].

Another change is domain swapping [63]. A primary contact is a contact between domains in a monomeric form of a protein. A dimer is *domain swapped*, if this primary contact exists, but between the different chains [13]. The prerequisite for this is thus a dimer, where each monomer has at least two domains that can be swapped. A variation of domain swapping is SSE exchange [127], which in principle is the same event, but only involves a swap of a SSE rather than a whole domain.

To complete the picture, we shall also mention the existence of domain stealing and relocation, both of which are also reviewed in [63].

6.1.4. Circular Permutations

Finally, a significant evolutionary event called circular permutation (CP) will be discussed. We give a more detailed description of this change than the others mentioned above, since CPs are a topic in two of of the papers I have written during my Ph.D. project; they are presented in appendices F and G.

Definition

A circular permutation (CP) [92] is a structural difference between two protein molecules that can be conceived as the result of a linkage between the C and N termini, followed by a linearision of the temporary, circular polypeptide by a cleavage at the surface [61] (see figure 6.1). A CP can also be defined on the sequence level, where it may be defined as the effect obtained by dividing a string in two, followed by concatenation of the parts in opposite order than the original. In both cases, the result of the CP is a circularly permuted version of the protein sequence/structure.

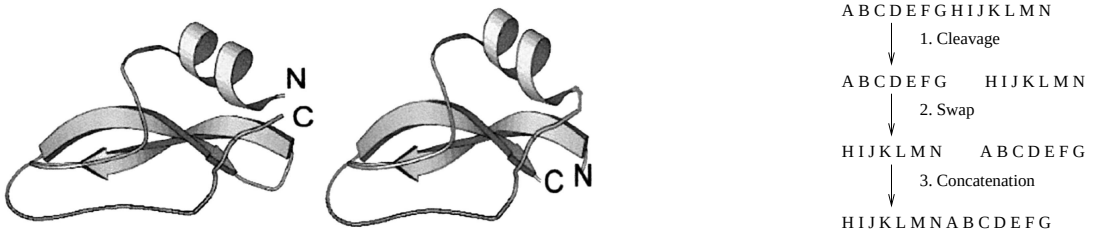


Figure 6.1.: A circular permutation exemplified through two protein structures (left) and the creation of circularly permutation letter sequence (right). Please note that the letters are not amino acid one letter codes, but arbitrary and abstract letters.

Natural CPs

Circularly permuted proteins have been found to occur naturally in many protein families [92]. Among these are the saposins [118], TIM-barrel enzymes like aldolases [75], FMN-binding proteins [105], C2 domains [107], β-glucanases [57, 62, 9], SLH domains [95], and cytosine-C^5 methyltransferases [169]. All of the above are examples of CPs occurring on the gene level. Evidence for the existence of this mechanism has been accumulating during the last decade, starting with the discovery of the "swaposin" domain in 1995 [118]. Until these discoveries, only CPs arising as a result of post-translational modifications had been observed. The most prominent example of this is possibly concanavalin A [31, 123], where the observable CP in the folded protein is not a result of changes on the gene level. Rather, after translation (and during folding) the polypeptide is subjected to the mentioned events of linkage and cleavage, resulting in a CP version.

Artificial CPs

In addition to CPs observed in natural processes, artificial CPs have been created in several studies. One of the first examples is reported by Goldenberg and Creighton [52], who made a CP version of *Bovine Pancreatic Trypsin Inhibitor*; Reeke *et al.* [94] similarly made a CP version of the gene cPRAI in *E. coli*, and finally Hahn *et al.* [57] modified the gene encoding *H(A16-M)*.

Although a CP might seem as a radical change to a protein, in many cases the protein surprisingly continues to fold into a similar architecture as before. This is because the majority of the intramolecular interactions stabilising the native structure are unaffected by the modification and still can exist in the CP version [61]. For instance, in a recent study of myoglobin, it was found that introducing a CP into the protein did not obstruct native folding; however, it did decrease the stability of the structure [125]. This seems to be a general trend: The CP version folds into approximately the same structure and often retains function,

but is less stable and vulnerable to the state of the environment.

Mechanism Behind CPs

The number of naturally occurring CPs found is not vast, and indeed CPs are infrequent in evolution. Still, CPs do take place, and many instances probably remain to be discovered. This logically introduces the question, what the mechanism behind a CP might be? An evolutionary event causing a swap of two parts of a gene in one unit operation – while still preserving a functional gene – seems highly improbable [127, 92, 153].

As suggested be several authors [118, 74], CP might arise by gene duplication of the precursor protein leading to an in-frame fusion of the genes to form a tandem repeat gene variant. The duplicated gene could exist in folded state as a dimer consisting of the two old folds. Next, an excision of a piece of the new gene would have to take place (e.g. by mutations causing stop codons to arise). Provided that the size of this excised part is roughly equal to the size of the original gene, this final product is likely to fold into the original architecture. The CP variant might be longer or shorter than the original due to "imprecision" of the cutoffs; hence the termini of the CP fold might differ significantly from its original counterpart.

An alternative hypothesis for the formation of CPs is that the proteins related by CP were formed by fusion of two smaller units into one larger unit [153]. This process would then have to happen independently twice in evolution with the same subunits – and in different sequential order.

Motivation for Investigating CPs

CPs are interesting for at least two reasons: Firstly, they play an active part in evolution of genes and proteins. It has been suggested that CP may serve as a constructive mechanism in evolution, because the function is usually maintained in proteins subjected to CP [153]. Moreover, the CP mechanism provides a natural form of flexibility as witnessed by the fact that many inserted domains are CPs of known homologous domains [56], which might in some cases ease the insertion of the domain. This drives an area of research to uncover the extent and function of CPs in evolution. One way to conduct this research is to search for new CP relations between protein sequences or structures to determine the extent of the CPs.

Secondly, CPs can be engineered in the laboratory, while the proteins continue to remain functional (although often less stable, as mentioned). This makes CPs interesting from a more practical and chemical point of view: New compounds with altered, but comparable, properties might be created. The ability to create such compounds might be beneficial in for instance drug design and in crystallisation studies. Artificial CP creation of a protein may ease the crystallisation and ultimately the structure determination of a new protein [129]. This moreover im-

plies — as the CP variant often retains the same overall fold and have unaltered binding pockets — that artificial CP creation can be a very useful method in determining the actual structure of drug-binding sites, when only the CP variant can be crystallised.

Finally, a central question in protein folding is whether the sequential order of residues is critical to the stability and folding of protein structures [94]. CPs are ideal for studying this issue [57] and the closely related vectorial folding hypothesis [30], which states that the N-terminus will fold into a fixed core structure, and thus directs folding of the remainder of the polypeptide chain. The hope to answer to the above question is a further motivation for studying CPs.

6.2. Homology Detection

6.2.1. Motivation

It is both important and desirable to be able to make a distinction between the two situations of divergent and convergent evolution (homology or analogy), because it can tell us how (or perhaps how not) the evolution of proteins has progressed. Specifically, if two proteins are shown to be homologous, we can analyse their differences and try to deduce the specific evolutionary events that have caused their divergence, and perhaps generalise these findings as wider applicable rules.

Unfortunately, it can be difficult to separate the two cases – for many pairs we do not know for sure. As with classification of organisms, the results are seldom provably correct, but may be considered more or less plausible. There are many examples that lie in this spectrum that range from very similar proteins — that most likely (almost definitely) are homologous — to proteins with vague, random similarities that probably (almost certainly) are non-homologous [148, page 271].

6.2.2. Ambiguity of the Fold Concept

The above mentioned uncertainties regarding homology assignment and classification of protein structures is illustrated by the fold concept, which is used in SCOP and CATH. The concept of a fold is inherently subjective according to several authors [121, 56], and fold classification has been called an approximative art [56]. It has also been mentioned that there are no independent means of verification (in homology vs. analogy questions), and therefore elaborate debates regarding fold classification should be considered are unscientific and be avoided [12].

To illustrate the difficulty in detecting similarities for remotely related proteins, we may take a look at *human Glutathione synthetase (GS)*, which belongs to the recently identified *ATP-grasp* superfamily despite the fact that it displays no

detectable sequence identity with other family members. *hGS* has a rare gene permutation, which has resulted in a circular shift of the conserved SSEs in it with respect to the other known *ATP-grasp* proteins. Nevertheless, it appears likely that the enzyme shares the same general catalytic mechanism as other ligases [117]. The pairwise sequence identity between *hGS* and *ecGS* after structure-based sequence alignment is only 10% [117].

These uncertainties do, however, not mean that we should not deal with fold classification and homology decision at all. By using a sensible approach to this area, we can get new insights and findings by observing contradictions in existing classifications, by presenting new and surprising similarities, and by finding new indicators for homology. If the tools are used and conclusions interpreted with care, then research on the degree of evolutionary similarity between protein structures can be fruitful and indeed scientific.

A special challenge arises when analysing SCOP with automatic structure similarity methods such as FASE or those mentioned in section 5.5.2. SCOP is based on a range of evolutionary indicators other than structure such as function, binding sites, cellular environment, and other factors [6] in addition to sequence and structural similarity.

Therefore, we present below a summary of the properties that can be investigated to determine the degree of relatedness between protein structures.

6.2.3. A Framework for Decision Making

In the following, we sum up some of the most important properties that should be investigated, when deciding if two proteins are related. Also, we discuss the relative importance of these properties.

The first property to investigate is sequence similarity. A pairwise sequence similarity above 25% implies high probability for homology. For instance, the FSSP database [68] has been constructed to contain no protein pairs with more than 25% sequence similarity [38, page 286]. Of course, for higher percentages the homology becomes even more definite; the CATH classification [115] groups proteins with sequence similarity above 35% in the same sequence family. For lower sequence similarities, the proteins might still be homologous, but other properties must be considered (and be similar) to show it.

Usually, if the sequence similarity was indecisive, one then looks at the structural similarity. Protein structure is more conserved than sequence in evolution [119, 56], so the structures might be very similar and indicate homology in spite of low sequence similarity. The relationship between sequence similarity and structural similarity has been investigated in many papers, for instance [168, 23].

Sequence similarity after structural alignment is also a very efficient indicator of evolutionary relatedness [119, 128, 4, 28]. The significance of a structural alignment can thus be made more or less pronounced by analysing this value. In particular, Murzin has derived a P-value [104, 4] for assessing the probability

that two structures are related, based on their sequence similarity after structural alignment.

In addition to sequence and structural similarity, and sequential similarity after structural alignment, there is a whole range of properties of the protein that one may look at when testing, if they have evolved from a common ancestor [41]:

- Enzyme-substrate interactions,
- catalytic mechanisms,
- residues essential for catalysis are in same sequence, and
- function.

In this context, the above aspects of the structures are less important than sequence and structure similarity. Nevertheless, they can be used to further support a homology (or analogy) hypothesis.

Thus, using the the properties reviewed above, one can start making a decision. The more characteristics in common, the more likely is divergent evolution (that is, homology); and the less common characteristics, the more likely is convergent evolution (analogy). However, as mentioned, in some cases it can be very difficult (almost impossible with current knowledge) to make firm conclusions. An attempt to implement criteria similar to the ones listed above in a decision system was made by Holm and Sander [69]: Structure alignments, sequence alignments, and functional characterisations (such as keywords from PDB files) were used in trying to separate physical convergence from common evolutionary history.

Finally, it is worth mentioning that many of the above characteristics are undetermined for many identified proteins. For the majority of proteins identified from the pool of sequenced DNA regions, the structure is unknown. Moreover, for most proteins with known structure, their precise function and cellular environments are unknown.

6.3. Detection of Circular Permutations

We now confine ourselves to the problem if deciding of two proteins may be considered to be evolutionary related by a circular permutation (CP). Compared to the general problem of homology classification discussed in section 6.2, this problem, of course, is more specialised. Many aspects of the process are similar, such as basic similarity detection, while other aspects require new tools; those are: assessing the "CP-likeness" of the similarities, and distinguishing true relatedness from symmetries arising from structural constraints.

6.3.1. Based on Sequence Similarity

When detecting a CP, the most direct and obvious method — as in the general case — is to look at the pairwise sequence similarity. There have been various examples of algorithms for finding CPs on the sequence level. A common method is to duplicate for instance the second argument and to search for similarities between this doublet and the original first argument [153]. An alternative method is to permute one of the input sequences, A and B. Let A' denote the permuted version of A, according to the hypothesised CP. Now A' and B can be compared using standard sequence comparison methods, and if the similarity score is significant, the pair can believed to be homologous.

Ponting and Russell [118] used an approach similar to the above described, to analyse sequences from the *Saposin* superfamily, and the result was the discovery of a fourth family within it, namely the *Swaposin* family. Similarly, Garcia-Vallve *et al.* [46] found evidence for existence of CPs within the *$(\beta/\alpha)_8$-barrel*-fold using sequence data from 18 *β-glucosidase* sequences (plus the patterns of predicted SSEs). Finally, also in [97], pairwise sequence alignments of glucosyltransferases and the α-amylase family enabled the authors to conclude that the α/β-barrel of the transferases was circularly permuted in relation to the one in the α-amylases. In neither of the three cases were structural data known and thus not used.

6.3.2. Based on Structural Similarity

If the 3D structures are known, the CP can be investigated using structural comparison methods. Because structure is more conserved than sequence during evolution [56], it is a more powerful and reliable method for detecting homology. There has, however, only been one study performing a large scale search for circularly permuted protein pairs using structural comparison, namely a study by Jung and Lee [78]. In that study, the structure comparison method used, SHEBA [77], was restricted to finding sequential alignments, and therefore the structures had to be permuted (or renumbered) in order to find the CP. Such a modification causes the new termini to be located at the cleavage point, and moreover introduces one new backbone binding (of possibly non-standard binding distance) between the old termini to be introduced. More precisely, the idea was the following: Structure B was permuted on the middle, and based on the alignment of A and the modified version of B, B', a second and final permutation between A and a further modification of B', B'', was made.

Thus, the above described approach searches for the appropriate CP in a series of comparisons using a linearly restricted structural alignment algorithm. In contrast, the study presented in the paper in appendix G uses a fully automated and general method, namely FASE (see section 5.6), for large scale search. Moreover, in appendix G, the found CPs are analysed and assessed in great detail, which also contrasts with the approach taken by Jung and Lee [78], who do not analyse

single comparisons.

6.3.3. Based on Sequence Similarity after Structural Alignment

Sequence similarity between a pair of structures after a structural alignment has been shown to be one of the best discriminators of homology and even functional similarity [4]. Murzin [104] derived a measure expressed by a P-value for the significance of a sequence similarity present in a structural alignment, and specifically concluded that a pair of structures were significantly homologous based on a sequence similarity of 23% after structural alignment. Russell [128] analysed 335 pairs of structurally aligned proteins from SCOP. They were divided into analogues, and remote, medium and close homologies, and it was found that there was a higher degree of sequence similarity in remote homologies than for analogues. Thus, this measure, sequence similarity after structural alignment, could help in distinguishing between homology and analogy. In [119] it was stated (by referring to [128]) that *sequence identities in excess of 12% were more likely to be associated with superfamily rather than mere fold similarities.* Also in [119], sequence similarity (and Murzins derived P-value) for structural alignments was used to argue for homology between domains of the β-trefoil structure.

6.3.4. Distinguishing CPs from Identity Alignment

There is one important issue to keep in mind when searching for CPs: As many structures have symmetric structural elements (e.g. β-trefoils, Jelly-rolls, Immunoglobulins, TIM-barrels, and Ferredoxins (see [138])) and are repetitive on the sequence level, one must carefully consider, if a CP match is merely a result of the symmetry or actually represents a CP [83]. More specifically, one is often faced with two alternative superpositions of proteins A and B (when allowing non-linearity): One corresponds to a relative CP alignment of the proteins, and one corresponds to the identity alignment, thus implying simple similarity of the proteins. It is a logical conclusion — and a frequently used approach — to argue that two structures are most likely related by a CP, if the CP alignment is more significant than the identity alignment [104, 128]. When using sequence or linearly restricted structure comparison methods, they question is, if the similarities between A and B' are more significant than the similarities between A and B (as above, B' denotes a circularly permuted variant of B). When allowing non-linearity one must simply compare the significance of the two alternative similarities.

Sergeev [131] used such arguments on *$(\beta/\alpha)_8$-barrel* structures; he found that a special scoring function for structural and sequential similarity was optimised precisely when two structures was compared with one of them being permuted.

Aloy *et al.* [4] compared structural similarities between *transit peptide cleavage system* (scop d1hpca_) and *glucose permase* (scop d1gpr_ _) with the similarities found when *transit peptide cleavage system* was circularly permuted. It was found that it was most likely that the pair of structures was related by a CP, because 10 more residues could be superposed when *transit peptide cleavage system* was permuted; moreover, the $RMSD$ when aligning with respect to the CP was 0.7 lower than using the identity alignment. Finally the sequence similarity was 33% for the CP alignment instead of 4% for the identity alignment.

Jung and Lee [78] compared the similarities between A and B' to those found between A and B by defining a threshold (five standard deviations away from the mean similarity score). If the A-B' match was judged to be significant by this threshold and A-B was not, the pair was defined as being related by a CP; but if A-B was significant as well, the CP similarities were ascribed to symmetries in the structures. Of course, any approach using thresholds in this fashion, might prove very sensitive and instable, when both scores lie near the threshold. This is because a pair, A and B, might be considered to be best related (or not to be) by the identity alignment (or alternatively by a CP alignment) based on two similarity values that might lie arbitrarily close.

Chapter 7

Conclusion

The research made during my Ph.D. has resulted in two conference papers, one journal paper, and one paper currently in submission for a journal. Also, I have made the FASE tool for structure comparison of proteins. As outlined in the introductory chapter 1, the research has been focused towards three goal areas; these were:

- Exploration of Stochastic Search Algorithms,
- Design of Efficient Tools for Solving Relevant Bioinformatics Problems, and
- Use of Bioinformatics Tools to Explore the World from a Bioinformatics Perspective

Looking at the four papers in chronological order, there is a trend in main focus area. In figure 7.1, the focus in each of the four produced research papers is indicated. The initial focus was on the exploration of search heuristics. In the two first papers, heuristics were tested on the PM problem, and on a numerical benchmark test set, respectively. The focus then shifted towards making bioinformatics tools, starting with the FASE paper. Towards the end of my Ph.D. project, the focus shifted towards exploring nature, namely discovery of CPs with FASE. However, the experiences gained on heuristics in the two first studies were also used in the design of FASE.

7.1. Exploration of Search Algorithms

7.1.1. PM Paper

In the PM paper [154], contributions were made to understanding *the PM problem*, *models for the PM problem*, and *heuristics for the PM problem*.

First, regarding the PM problem, the inherent ambiguity of the problem was investigated. This ambiguity is the degree to which the ordering of clones remains uncertain based on the hybridisation data. The investigation showed that

	Explore Heuristics	Design Tools	Explore Nature
PM Investigation	9	1	0
Alg. Comparison	10	0	0
FASE Presentation	1	7	2
CP Evaluation	0	1	9

Figure 7.1.: Illustration of the focus during the Ph.D. Papers are listed in chronological order. On a linear scale, numbers from 1 to 10 indicate the approximate degree of focus on a given topic in a given paper.

the data did not define a unique ordering — there is an upper bound on approximately 90% for the prediction succes in PM (indepentently of fitness function and algorithm used).

The correlation between fitness function and problem describes the quality of the model. With a perfect model, the true ordering would have to have the optimal fitness. Unfortunately, as showed in the paper, due to errors in the data, the true ordering does not have the best value for realistic error rates; thus, errors alone bound the optimal expected prediction succes to 90%.

Heuristics for PM were also investigated, and only showed minor performance differences. In comparison to variations caused by the chosen model and the quality of the data, the differences caused by algorithm choice were negligible. In particular, it proved hard to combine good parts from two orderings into one, and thus the EA did not significantly outperform a simple local search. This was attributed to a very discontinuous search space with weak neighbour correlation.

Overall, the analysis of PM as a problem, and the realisation that LS is as efficient as EAs for such problems, were the main achievements of this paper.

7.1.2. Algorithm Comparison

The second paper [159] — a large comparison study of DE, PSO, EAs, and SA — obviously extended the work on heuristics made in the previous paper. However, it was also a continuation of the work from my masters thesis, where the topic was optimisation with PSO [157].

The paper was the first large scale evaluation of DE in relation to existing heuristic PBSMs. The paper investigated the hypothesis (originating from superior performance in several real-world problems) that DE could outperform its EC counterparts such as EA, ES, and GA on classical, numerical benchmark problems. For the first time, the paper documented — on a diverse suite of 34 well-known, high-dimensional, highly multi-modal test problems — that DE compared to its competitors was more robust in reproducing its results, faster in convergence, possessed better fine-tuning capabilities, and had fewer parameters

to tune in achieving all of the above.

7.2. Design of Bioinformatics Tools

The design of efficient tools for solving relevant bioinformatics problems was the focus and contribution of the FASE paper [158]. In addition to this, the FASE paper also explored and used heuristics as a part of the design of FASE and also laid the foundation for discovery of novel CPs.

7.2.1. FASE Paper

The FASE paper [158] primarily presented FASE — a novel method for structure comparison, superposition, and alignment of proteins. FASE can find non-sequential structure similarities, which many structure comparison programs cannot. FASE uses a new idea to make initial alignments of the protein structures according to their SSEs. This improves the runtime, while still enabling discovery of very subtle similarities.

A novel consensus scoring method was introduced and used. This measure combined classical $RMSD$ and alignment size based measures with known structure similarity measures in order to increase robustness in assessment.

A measure for finding similarity to the identity alignment was derived, and a measure for the detection of CP alignments was presented. These measures is used by the FASE program to detect CPs, and to assess the likelihood of a possible CP alignment of two proteins relative to alternative alignments of them.

It was demonstrated how values for significance testing of similarities could be derived; in the paper, this general approach was made for fold discrimination.

In conclusion, FASE was shown to be robust with defaults parameters, applied DP in a novel way that still allowed discovery of non-sequential alignments, had a better runtime complexity than competetive methods, and used consensus scoring to improve ranking of similarities in difficult cases. FASE has been tested on standard similarity detection, fold discrimination, and demonstrated the ability to consistently find weak similarities and CPs. On all areas, the performance is equal to or better than the best alternative method.

7.3. Use of Bioinformatics Tools

The use of bioinformatics tools to explore nature was the third focus area of the conducted research, and was most clearly carried out in the paper on detection and evaluation of novel CPs [156].

7.3.1. Novel CP Paper

Through its application of FASE and discovery of novel CPs, the fourth paper [156] builds significantly upon the research background on heuristics. The paper included a review of natural and artificial CPs, described possible mechanisms for emergence of CPs and provided the motivation for studying them. Then, methods for assessing homology in general and CPs in particular were reviewed, and used to derive a methodology for the subsequent detection and evaluation of the novel CPs.

The main contribution of this paper was the thorough, systematic, and general approach to the discovery of CPs; moreover, this was combined with a detailed analysis of a dozen of structure pairs that appeared to be related by CP. The analysis consisted of visual analysis of the superpositions found by FASE, inspection of similarity values as found by FASE, and a measure of sequence similarity after structural alignnment (SSSA) found in other studies to be significant in predicting homology. In each case, these properties were used to assess the plausability of the hypothesis that a given pair were related by CP. Also, any CP was compared with a possible identity alignment of the structures.

Finally, in a comparative analysis of the similarities for novel and known CPs (according to the SCOP database), respectively, pairs of subtrees — as indicated by novel and known CP relations — were analysed for similarities. This analysis further supported many of the novel CP findings made, and moreover many existing relationships seemed less plausible compared to some of the novel ones.

Appendix A

Technical Background

A.1. NP-Definitions

Definition A.1 (NP-Problem) *A problem is assigned to the NP (nondeterministic Polynomial time) class if*

- *it is solvable in polynomial time by a nondeterministic Turing Machine, and*
- *if a solution to the problem is known, it can be reduced to a single polynomial time verification.*[1]

Definition A.2 (NP-Hard Problem) *A problem is NP-hard if an algorithm for solving it can be translated into one for solving any other NP-Problem. NP-hard therefore means "at least as hard as any NP-Problem," although it might, in fact, be harder.*

Definition A.3 (NP-Complete Problem) *A problem which is both*

- *an NP-problem, and*
- *NP-Hard*

is called an NP-Complete Problem.

The above definitions are in accordance with: E. W. Weisstein, "The CRC concise encyclopedia of mathematics" [163].

[1] In some definitions of an NP-problem it is *only* required that a solution to the problem can be verified in polynomial time for the problem to be in NP. I.e. the first point of definition A.1 is not required.

A.2. Least Squares Minimisation of Two Point-Sets

Superposition of two point-sets is a very central operation in FASE and in most other structural alignment algorithms. Although distance matrix based methods can avoid the use of this operation, because they use the distance matrices of the structures to obtain an alignment, the vast majority of structural comparison methods must implement a least squares minimisation routine. Even distance matrix based methods such as DALI [67] and SSAP [150] must also implement a least squares minimisation routine similar to the one described below, if a final superposition is needed. The method presented — and used in FASE — is by Horn [70].

Data: We are given two point-sets, a and b each containing n points in 3D-space. Formally, $a = (a_1, a_2, a_3, \ldots, a_n)$ and $b = (b_1, b_2, b_3, \ldots, b_n)$, where $a_i, b_i \in \mathbb{R}^3$ for $1 \leq i \leq n$. Thus, a_i represents the i'th point in point-set a, and a_{ij} represents the j'th coordinate (x, y, or z coordinate) of the i'th point in point-set a. Similar definitions apply to b_i and b_{ij}.
Problem: Determine rigid transformations T_a and T_b of a and b such that

$$RMSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(T_a(a_i) - T_b(b_i))^2}$$

is minimised, where $T_a(a_i) - T_b(b_i)$ is the Euclidean 3D distance between $T_a(a_i)$ and $T_b(b_i)$.
Solution: Calculate the centroids, C_a and C_b, where $C_a = \frac{1}{n}\sum_{i=1}^{n} a_i$ (similar for C_b). Denote with a' and b' the point-sets a and b with their respective centroids subtracted, thus: $a'_i = a_i - C_a$ and $b'_i = b_i - C_b$.

Find the 4×4 matrix N:

$$N = \begin{pmatrix} S_{11}+S_{22}+S_{33} & S_{23}-S_{32} & S_{31}-S_{13} & S_{12}-S_{21} \\ S_{23}-S_{32} & S_{11}-S_{22}-S_{33} & S_{12}+S_{21} & S_{31}+S_{13} \\ S_{31}-S_{13} & S_{12}+S_{21} & -S_{11}+S_{22}-S_{33} & S_{23}+S_{32} \\ S_{12}-S_{21} & S_{31}+S_{13} & S_{23}+S_{32} & -S_{11}-S_{22}+S_{33} \end{pmatrix}$$

where the value $S_{j_1 j_2}$ is defined as

$$S_{j_1 j_2} = \sum_{i=1}^{n} a'_{ij_1} a'_{ij_2}$$

.

Find the eigenvector q (with dimension 4) corresponding to the largest eigenvalue of N. Interpret q as a quaternion that directly defines a rotation. Convert the quaternion q to a corresponding rotation matrix M:

$$M = \begin{pmatrix} sq_0 + sq_1 - sq_2 - sq_3 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & sq_0 - sq_1 + sq_2 - sq_3 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & sq_0 - sq_1 - sq_2 + sq_3 \end{pmatrix}$$

where $sq_i = \sqrt{q_i}, 1 \leq i \leq 4$.

The optimal superposition of the original point-sets a and b is given by the two transformation functions:

$$T_a(x) = M(x - C_a) \text{ and } T_b(x) = x - C_b.$$

The operation of T_b on a point from b is simply given by a centering of the point-set b, while T_a operates by first centering a followed by application of the found rotation matrix M.

Appendix B

Overview and Practical Usage of FASE

B.1. Overview and Availability

FASE has been written in C++ and implemented as a program that runs on Windows and Linux platforms, but it should be possible to compile a version of FASE on any platform with a decent C complier. FASE is available from

`http://www.daimi.au.dk/~jve/FASE`

A distinctive feature of the Windows version of FASE is that if input files have not been specified through the command line, a dialog box appears for choosing the PDB files to be compared. Moreover, on Windows a dialog box displays the progress of FASE during the comparison in addition to the console output present on any platform.

For each comparison, all data is placed in a new, unique directory (created in the current directory) named after the two structures that are being compared. Output data can easily be inspected and analysed in a browser through a set of html files produced by FASE.

B.2. Parameter Usage

In most cases, FASE should be executed with default parameter settings, but of course it *is* possible to change them via the command line. In some situations it might be beneficial to set certain parameters such that the sought for similarities between protein structures are found. In addition, the parameters can control the execution time of FASE — and hence, potentially, the quality of the found solutions.

In the following we will describe the parameters of FASE.

- *Minimum number of consecutive residues*
  Option: -c x.

This parameter ensures that each aligned residue will be part of at least x consecutively aligned residues. The exact details of how the requirement is fulfilled by FASE is described in section 5.6.4. The default value is 4.

- *Maximum alignment distance*
  Option: -d x.

  This parameter ensures that each aligned residue will not be further away than x Å from its aligned and corresponding residue in the other protein structure. The default value is 5.0.

- *Minimum number of residues in a solution*
  Option: -r x.

  This parameter ensures that each solution will contain at least x aligned residue pairs. This requirement is enforced after phase 1 as well as for the outputted solutions. The default value is 10.

- *Maximum number of iterations of superposition and alignment*
  Option: -i x.

  This parameter ensures that not more than x iterations of superposition and alignment will be made in the search and refinement phases. As described in section 5.6.4, if a cyclicity is detected in the alignments, the iteration is immediately terminated; in the vast majority of all cases this happens after less than 10 iterations, so this parameter is not crucial. However, by setting the parameter very low (to e.g. 3) it is possible to get an even lower running time – of course on the expense of more unstable results. The default value is 50.

- *Percent of SSEs that must match for further search*
  Option: -f x.

  This parameter dictates that x percent of the SSEs must match (using mutually nearest neighbourhood matching) for the search to proceed further after the initial alignment of a pair of SSEs. This parameter is not used in the default setting. It can, however, be used to achieve lower running times in particular for comparison of very large structures, for which FASE becomes perceptibly slower (see e.g. appendix C). A setting of 0.2 typically halves the running time without missing the best solutions (except for extremely rare cases). As mentioned, the default value is that the SSE matching requirement is disabled, i.e. 0%.

- *Maximal angle between CaCb vectors for residues to be equal*
  Option: -b x.

  This parameter dictates that the maximal angle between the CaCb vectors of two residues must be less than x for the residues to be alignable. This

parameter is not used in the default setting. It may, however, be used to test the found similarity on a more demanding level, since that it indirectly can ensure that the orientations of the side-chains are similar for an aligned pair of residues. As mentioned, maximal angle checking is disabled, i.e. the default value is ∞.

B.3. Server Setup

Few structural comparison programs are available as executables for a standard operating system, whereas most are available as web-services. In many of these services it is possible, given a query structure, to find the most similar structure among "all" known structures.

Here, we will outline how this can be achieved with FASE. Firstly, we will not search all approximately 30000 pdb structures (or approximately 70000 SCOP domains). Rather, we will search only 1000 representative structures. This corresponds to one structure per fold in SCOP. These representative are divided into alpha, beta, and mixed depending on their SSE content, which may be matched with the SSE content of the query. Thus, only 300-600 comparisons must be made for a query. Using 10 CPUs, this can be done in $600/10 = 60$ times the search-time for a single comparison, which is approximately 4 seconds on a standard PC for a typical pair of SCOP domains (see figure C.3). Thus, the total answer time for such a query would amount to approximately $60 \cdot 4 = 240$ seconds $= 4$ minutes.

For further accuracy or granularity, this process can be repeated for the ten most similar fold representatives found using the above query. The query structure could then be compared to e.g. 6 diverse structures from each fold, where each of these 6 structures would represent a protein family. This second accurate phase would require $10 \cdot 6 = 60$ comparisons, which (still using 10 CPUs) could be done in less than half a minute.

Appendix C

Experiments on FASE

C.1. Analytical Analysis of Running Time

The running time of FASE is dominated by the initial search phase. As described in section 5.6.4, every SSE from structure A is aligned with every SSE of structure B in five different ways (see figure 5.4) assuming the SSEs have same type. The time it takes to perform each of these alignments is proportional to the size of the largest protein, because the time it takes to align each residue essentially is constant (that is, independent of the size of the other protein). The dominant term in the running time of is thus proportional to:

$$|SSE_A| \cdot |SSE_B| \cdot \max(|A|, |B|).$$

Assuming that the number of SSEs in a structure is proportional to the size of the structure, and that structure A is the larger structure, then the running time is actually bounded by an expression that is proportional to $|A|^3$. Thus, using n to denote the "problem size", the time-complexity of FASE is

$$O(n^3).$$

Note, that for methods like TOP [93] and MASS [35], a similar careful analysis of their running times would yield $O(n^5)$, because they also must make residue alignment for each initial alignment that is generated by aligning two SSE pairs. However, they operate with n^4 of those initial alignments, instead of the n^2 alignments that FASE generates, so the total time-complexity becomes $O(n^5)$.

Looking at the $O(n^3)$ running time for FASE again, we notice that there are some natural speed ups, or optimisations, when structures grow in size. One is that large proteins have large fractions of α and β SSEs. In other words, large proteins tend to consist not only of α helices or β strands. Thus, the full potential of the $|SSE_A| \cdot |SSE_B|$ factor is to a lesser extent fulfilled for large structures, because they have more even distribution of the SSE types, as more SSEs are incompatible, and thus FASE does not procced to a more lengthy comparison

for these pairs. The actual running time of FASE (see section C.2) for practical purposes is therefore

$$O(n^k)$$

where $2 \leq k \leq 3$.

C.1.1. Comparison to TOP and MASS

In this section, we wil make a direct comparison of, on the one hand, TOP and MASS and, on the other hand, FASE.

TOP and MASS are examples of what might be called the SSE pairs alignment approach, because all pairs of SSEs from structure A are aligned with all pairs of SSEs from structure B. Conversely, FASE may be called a fixed number of SSE alignment approach, because each SSE from structure A is aligned with each SSE from structure B in a limited and fixed number of ways.

The running times can be expressed, respectively, as

$$k_1 \cdot |SSE_A|^2 \cdot |SSE_B|^2$$

for TOP and MASS, and

$$k_2 \cdot 5 \cdot |SSE_A| \cdot |SSE_B|$$

for FASE. The constant 5 comes from the fixed number of ways each SSE from A is aligned with each SSE from B in FASE.

There are no reasons to believe that k_1 and k_2 are significantly different in the two approaches, because the constants express the time used to refine an alignment of two proteins, which is done independently of the differences between TOP/MASS and FASE.

Thus, by assuming that $k_1 \sim k_2$ and equating the two above expressions, we can conclude that the FASE approach is faster for

$$|SSE_A| \cdot |SSE_B| > 5$$

which is true for almost all pairs of protein structures. This analysis thus reveals that the FASE approach has an inherent advantage in time complexity compared to the ones of TOP and MASS.

C.1.2. Best Solution Effect

A circumstance that affects the relative running times of phase 1 and phase 2 of FASE is the quality of the best solution (or similarity) found between the structures after phase 1. A very strong similarity will cause many phase 1 solutions to be weeded out before phase 2, because they must have less than 33% of the

number of aligned residues than the best solution has. Conversely, if the best found similarity is just a random similarity, no single solution will "stand out" from the rest and weed out other inferior solutions. To sum up, for almost similar structures, phase 2 will take very little time compared to phase 1. For structures with weak similarities, the two phases will be of approximately same length.

This relationship is controlled by a number of parameters that determine the duration of them. However, the weeding between them is the most important element, and three parameters controls the weeding: 1) a lower bound for the number of residues that must be in an alignment, 2) the percentage of aligned residues a solution must have relative to the best solution found so far, and 3) after sorting phase 1 solutions by alignment size and *RMSD*, only the n (default 6) best solutions for each alignment size are transferred to phase 2. For further details, see section 5.6.4.

C.2. Testing of Running Time

In section C.1 the running time of FASE was determined by an analytical approach. Moreover, the usage of FASE has been described in appendix B and in particular the significance of its parameters has been described in section B.2.

However, since FASE is a heuristic, it is still very important to obtain actual empirical indications of the overall running time for standard inputs. It is also important and useful to get an impression of the change in running time for certain variations of parameters. Finally, it is central to empirically get some impression of the robustness of the FASE heuristic; for instance, the change of quality and reliability when varying the consecutive number of residues parameter.

These questions will be investigated below in this appendix through a number of experiments.

C.2.1. Input Size

From a set of 253 structures (see figure C.1) a list of 500 randomly selected comparisons were made. Firstly, we wished to verify the first formula of section C.1, which states that the running time of FASE is bounded by the product of the number of SSEs in the two structures and the number of residues in the largest structure.

In figure figure C.2, we therefore plotted this upper bound on the x-axis and the running time on the y-axis. From figure C.2 it is clear that there is a good proportional correlation between the theoretical upper bound (indicated by the top-most line in the figure) and the actual running times for the "slowest" comparisons. Moreover, many comparisons are faster than this upper bound due to the reasons mentioned in section C.1 above. By manual judgement, it seems that the average expected running time for a given problem size is approximately half

```
137L  1AYA  1CBF  1CZY  1ETB  1HBI  1JUD  1MJC  1PFC  1RIN  1TME  2AZA  2HMZ  2TGI  4BCL
155C  1AZC  1CC5  1DC7  1EXG  1HBS  1K61  1MOL  1PLC  1RIS  1TMY  2BNH  2HSP  2TMV  4BNH
1A1E  1AZU  1CCR  1DCK  1EZ3  1HDS  1KIT  1MUC  1PMB  1RNL  1TNF  2BPA  2LH2  2TSS  4CLN
1A1V  1B6G  1CDE  1DHR  1F5S  1HHO  1KJQ  1MUP  1PNE  1ROP  1TUL  2CCY  2LHB  2UDP  4ENL
1A5D  1BBH  1CDK  1DJI  1FBA  1HKB  1L2H  1NFP  1PNT  1RSY  1UBQ  2CHB  2LTN  2WRP  4FGF
1AAJ  1BD7  1CEW  1DLW  1FDH  1HKC  1LGR  1NKL  1PSI  1SHF  1UN2  2CLN  2LZ2  351C  4FX2
1ACX  1BEO  1CHC  1DLY  1FHA  1HLB  1LPE  1NLS  1PYS  1SLC  1VKK  2CNA  2LZM  3BNH  4HHB
1ADL  1BGE  1CHN  1DPG  1FJ7  1HMY  1LUC  1NNL  1QAS  1SRI  1VLA  2CRO  2MHB  3C2C  5BNH
1AH7  1BGU  1CID  1DQG  1FNA  1HNF  1LYZ  1NSB  1QDM  1SRR  1VQB  2DHB  2MHR  3CHY  5CPV
1AII  1BKS  1COB  1DSB  1FRD  1HOE  1LZ1  1OOU  1QMP  1STF  1WLK  2DOR  2OMF  3CLN  5P21
1AIZ  1BMV  1COL  1DSX  1FXA  1HSB  1LZT  1O97  1RO9  1STP  1XNB  2FOX  2PAB  3CYT  5PAL
1AJK  1BN6  1CPC  1DYN  1FXI  1HYP  1MOE  1ONC  1R69  1SU4  1YVE  2GCH  2PTN  3GRS  5TNC
1ALC  1BOO  1CRL  1ECA  1GBG  1IFC  1MAI  1ONE  1RBL  1SUG  256B  2GDM  2RHE  3HHR  7API
1AQI  1BOV  1CRN  1ECO  1GD1  1IZ8  1MBA  1ONR  1RCB  1SUW  2ABL  2GMF  2RSL  3HLA  8ATC
1ASH  1BRN  1CUK  1EDE  1GMP  1J97  1MBN  1OVA  1RCF  1SWG  2ADM  2HBG  2SIM  3ICB  9BNH
1AVD  1BTM  1CUS  1EG2  1GPB  1JS8  1MBS  1PAZ  1REC  1TEN  2ASR  2HDD  2STV  3SGB
1AXN  1BWW  1CYC  1EPU  1GTQ  1JT6  1MI1  1PDG  1RFB  1TIE  2AYH  2HMQ  2TCT  4APE
```

Figure C.1.: The 253 pdb identifiers of the structures used in the empirical testing of the running time of FASE.

the time for worst case demarcation-line as indicated by the middle line. The bottom line indicates a manually found lower bound.

The same comparisons can also be visualised in a standard "size versus time" plot — with the size of the largest structure on the x-axis and the running time of FASE on the y-axis. This is shown in figure C.3.

For comparisons with the largest structures smaller than 150 residues, all running times are done below 6 seconds in a standard PC laptop (Intel Pentium 4, 2 GHz) and for those comparisons the median comparison time is below 2 seconds. For comparisons below 300 residues, 90% are done in less than 10 seconds. Overall (including structures with up to 994 residues), 89% of all comparisons are performed in less than 40 seconds.

C.2.2. Consecutive Residues

The parameter settings of FASE play a significant role in determining the actual running time of the program. For instance, the parameter for minimum number of consecutive residues makes the program run faster if more consecutive residues are required. This is because many solutions are discarded when the requirements are higher (and the program will in total have to process less data), and because similar solutions tend to be merged or tend to converge into one solution when strong constraints are present. This naturally also makes FASE run faster.

We have investigated this effect — that we could call the *consecutive residues effect* — through an experiment. We compared a given pair of structures a number of time while only varying this particular parameter; this procedure will illustrate the speed improvement. The results are shown in figure C.4.

The average running time of a given comparison is reduced to below 60% of the original, when using 3 consecutive residues; it drops to almost 40%, when

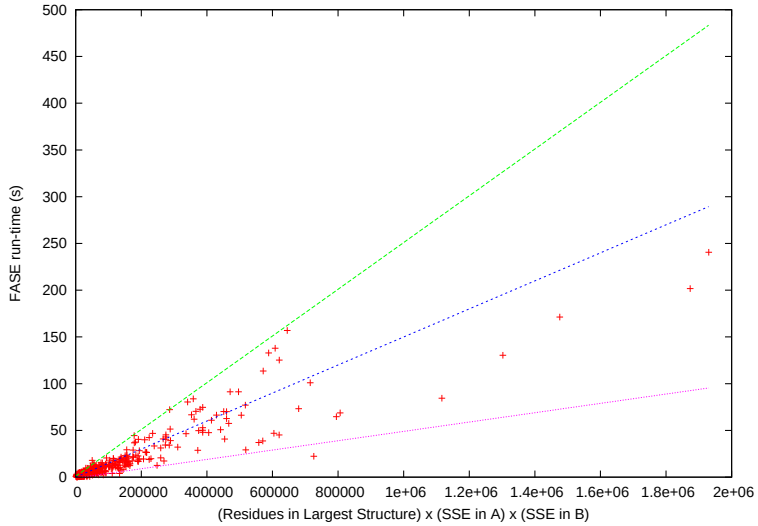


Figure C.2.: The running time of FASE and worst case time-complexity. The three fitted lines indicate manually found average, maximum, and minumum bounds for the running time of FASE.

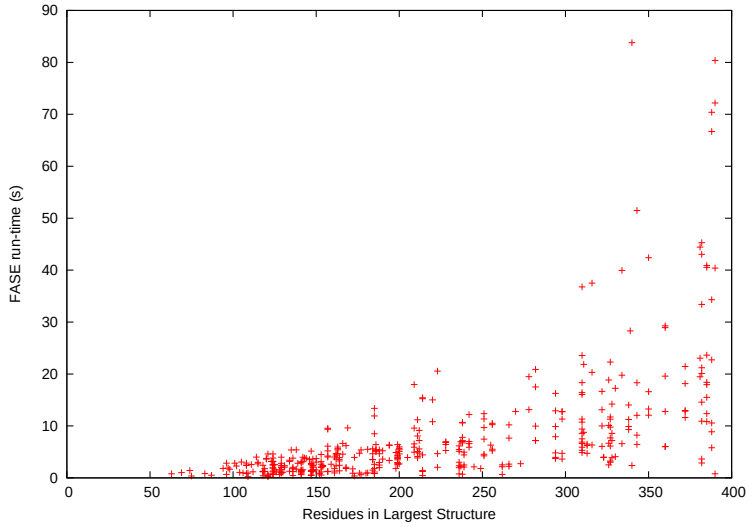


Figure C.3.: The running time of FASE for different input sizes. Each point corresponds to a comparison of two structures.

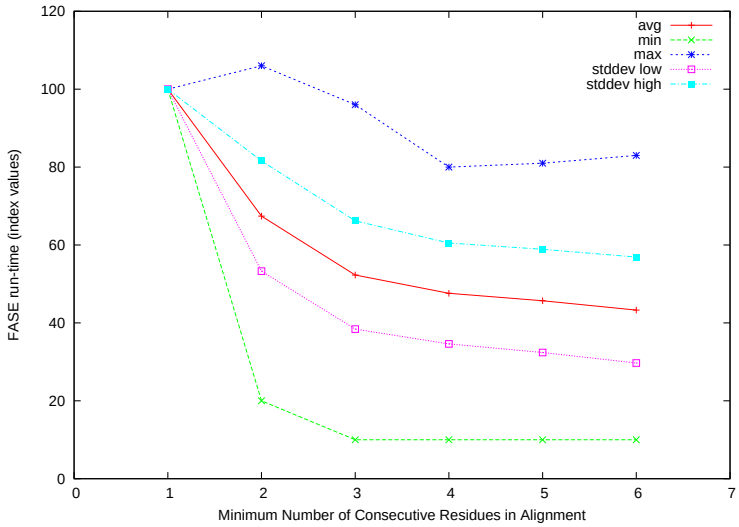


Figure C.4.: The running time of FASE for different values of the minimum number of consecutive residues parameter.

requiring 6 consecutive residues in the alignments.

C.3. Test of Solution Quality

C.3.1. Varying Consecutively Aligned Residues

In order to investigate the effects of requiring more consecutive residues in the alignment, an experiment was carried out. On a set of 143 pairs, each pair of structures was aligned with the requirements of 1, 2, 3, 4, 5, and 6 consecutive residues.

In figure C.5 it is illustrated how the average number of aligned residues decreases when tightening the consecutive constraint, and how the $RMSD$ also tends to decrease. Thus, based on these two measures, neither the use of one nor four consecutive residues is the most favourable choice — rather, they each express the natural trade-off between low $RMSD$ and high N. Using the other scores, Structal, GaFit and TOP, it is for two of them possible to obtain an improvement of scores by relaxing the consecutive constraint. The Structal measure seems to assign relatively constant scores to the found solutions when varying the parameter, while — according to GaFit and TOP — it possible to obtain better alignments when only requiring one consecutive residue.

However, using few consecutive residues tends to produce solutions with more irrelevant and random matches, and the "quality" of the solutions decreases. It is precisely this fact that Structal uses in assessing a solution: Gaps are punished,

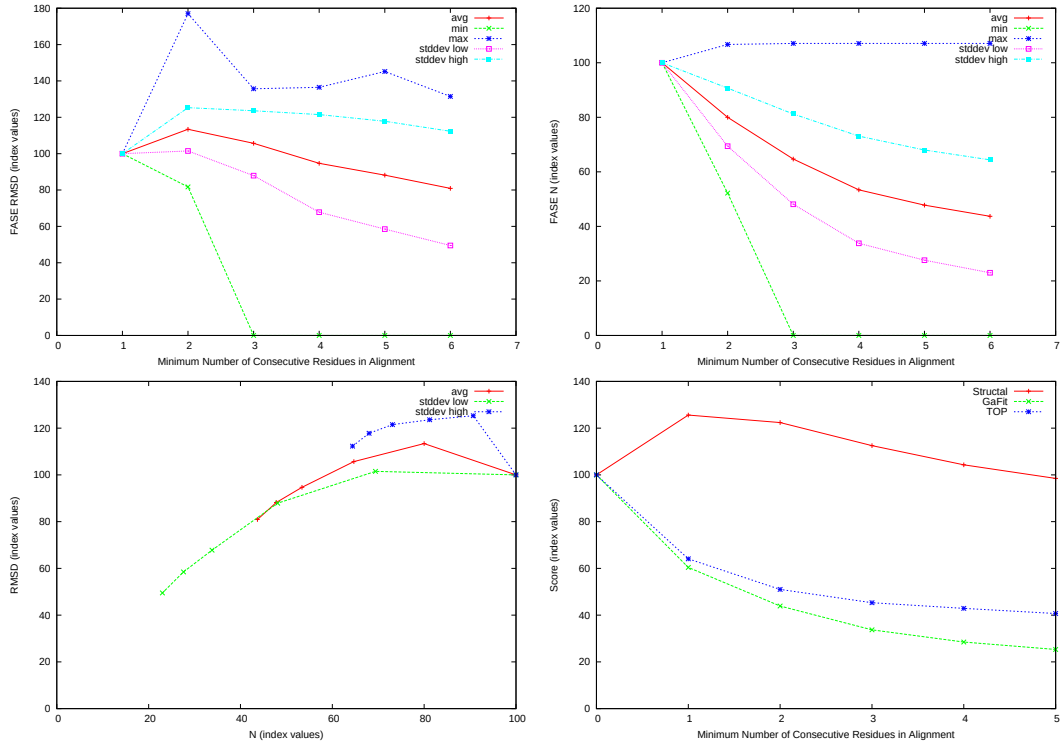


Figure C.5.: The RMSD (left) and N (right) of FASE for different values of the minimum number of consecutive residues parameter. The combined plot is shown below to the left. Finally, (bottom right) the development of the Structal, GaFit, and TOP score functions is depicted for increasing consecutive values.

and using few consecutive residues results in many gaps. Especially for divergent pairs, the added similarity when only few consecutive residues are required, is an artifact of the optimisation progress, and the found similarities are more prone to be "random" similarities rather than interesting structural similarities.

This fact is somewhat difficult to express with a number (measure), but can be shown through single examples. The similarity between 1CRL and 1EDE is in the twilight zone with respect to homology. The TOP score has a Z-value of 1.59, and this means that many structural similarities can be picked up by FASE. However, many random matches are also found if only one or two consecutive residues are required. This is illustrated in figure C.6, where random single matches slowly are weeded out, and where the alignment slowly converges towards the "true" and "noiseless" alignment of the two structures.

C.3.2. Varying SSE Percentage Match

For the 500 total comparisons from section C.2.1 (see e.g. figure C.1 for the PDB IDs), the SSE fraction matchings were recorded. Only two comparisons were found with a significant similarity (Z-score above 1) *and* with the best solution having a SSE fraction matching value less than 0.2 — i.e. significant similarities that would have been discarded, if the SSE fraction matching had actually been enforced, using the setting of 0.2.

In the first case (PDB IDs 2DHB and 1CPC), the best comparison (SSE fraction value 0.12) was "lost" in filtering between phases 1 and 2 (see page 88); however, almost the same solution was still obtained, just having a different trade-off regarding $RMSD$ and N, namely lower N and lower $RMSD$.

In the second case (PDB IDs 2WRP and 1BBH), the "lost" best solution was a very insignificant match: Only one long helix was matched, which — seen in relation to three long unmatched helices in structure A and many unmatched helices in structure B — must be considered an insignificant match. The Z-value above 1 reflects the inability of the scoring function to take the abundance of long alpha helices — that often fit with very low $RMSD$ without any homology — in protein structures into account. The overall match is only 25%, and thus this second "miss" is concluded actually not to be a miss.

In conclusion, based on this experiment, it seems reasonable that the SSE matching fraction threshold can be set to 0.2 almost certainly without discarding out any significant matches. Although more rigorous testing must be done before engaging in large scale usage with this setting (0.2), it seems that this setting speeds up the comparison without any loss of ability to discover interesting similarities.

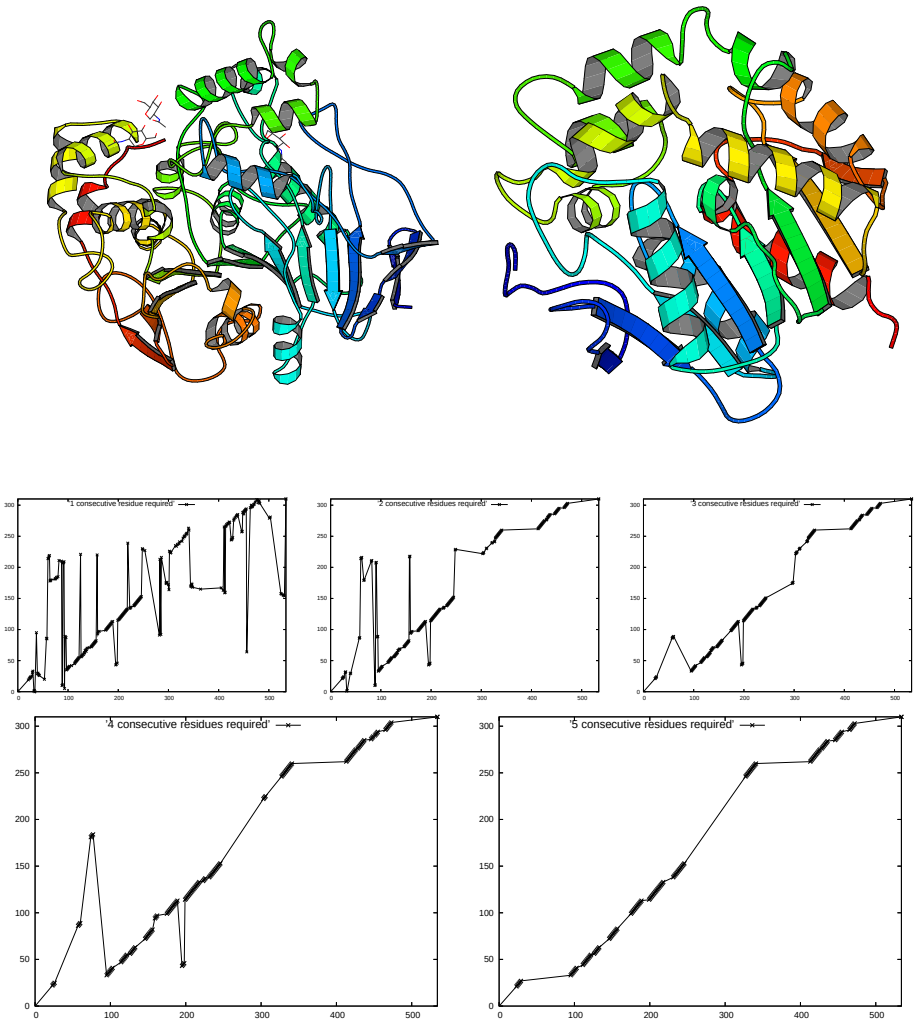


Figure C.6.: ***Top row:*** The compared structures, 1CRL and 1EDE. ***Middle and bottom rows:*** Dot-plots of the alignments of the structures requiring 1, 2, 3, 4, and 5 consecutive residues in the alignments.

Part II.

Papers

Appendix D

Physical Mapping using Simulated Annealing and Evolutionary Algorithms

The paper *Physical Mapping using Simulated Annealing and Evolutionary Algorithms* has been presented on the *2003 Congress on Evolutionary Computation* (CEC2003) and published as a conference paper in the *Proceedings of the 2003 Congress on Evolutionary Computation* (CEC2003) [154].

Physical Mapping using Simulated Annealing and Evolutionary Algorithms

Jakob Vesterstrøm
EVALife Research Group, Dept. of Computer Science, University of Aarhus, Denmark
Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, jve@daimi.au.dk

Abstract- Physical Mapping (PM) is a method of bioinformatics that assists in DNA sequencing. The goal is to determine the order of a collection of fragments taken from a DNA strand, given knowledge of certain unique DNA markers contained in the fragments. Simulated Annealing (SA) is the most widely used optimization method when searching for an ordering of the fragments in PM. In this work, we applied an Evolutionary Algorithm to the problem, and compared its performance to that of SA and Local Search on simulated PM data. Furthermore, we investigated the simulated data, in order to determine the important factors in finding a good ordering of the segments. The analysis highlights the importance of a good PM model, a well-correlated fitness function, and high quality hybridization data. We suggest that future work in PM should focus on design of more reliable fitness functions and on developing error-screening algorithms.

1 Introduction

The goal in Physical Mapping (PM) is to determine the order of a set DNA fragments (clones) extracted from a given strand of DNA [15]. Determining this order is of great importance in sequencing projects. In a PM experiment, a set of short DNA subsequences (probes) are tested for inclusion in each clone. The probes are unique with high probability on the DNA strand; thus, if two clones contain the same probe, they must overlap. The probe-occurrence information can be put into a hybridization matrix, where each row corresponds to a clone and each column corresponds to a probe. If clone i contains probe j, the clone is said to hybridizes to the probe, and the corresponding index in the matrix is set to 1. If they do not hybridize, the index is set to 0.

In an idealized experiment, if a clone hybridizes to two different probes, it must also hybridize to any probe located in between these two probes on the DNA strand. Consider a matrix created by ordering the columns of the hybridization matrix such that the probes appear in their true order. In such a matrix, all 1s in a row must be consecutive. Not all matrices can be reordered by column swaps to make all 1s in a row be consecutive. However, if it is possible, we say that the matrix has the "consecutive ones property" (C1P).

If the experimental procedures (from which the hybridization matrices originate) were always error-free, PM would be uninteresting from a computational point of view. The problem of efficiently deciding if an error-free matrix has the C1P, and subsequently finding a column permutation that brings the matrix to a form where all 1s in a given row are consecutive has already been solved in [3]. However in reality, errors are (practically always) present in hybridization matrices. Three types of errors often occur: False negatives, false positives, and chimerisms. A false positive is when a clone does not contain a given probe, but the experiment indicates that the clone does contain the probe. A false negative is when a probe is contained in a clone, but this fact is not reflected by experiments. Chimerism happens during the "production" of clones. In this process, when the original DNA strand is broken into fragments, two of these might join into one artificial fragment.

Finding the true probe ordering under these conditions can be very difficult. The hybridization matrix is not bound to have the C1P, and finding the permutation that comes closest (e.g., with f_5 defined in section 2.2 as a distance measure), is an NP hard problem.[1] Furthermore, the notion of a "good ordering" is no longer well-defined, because each proposed ordering much be evaluated with respect to a model of the experimental errors. In contrast to the error-free case, an optimal ordering is no longer easy to define. Finding a good measure for an ordering of the probes is both important and difficult.

Based on these facts, the most tractable approach is to define a fitness measure for evaluating the quality of a proposed ordering. The space of orderings can then be searched for high quality orderings. Simulated Annealing (SA) has been used widely as optimization heuristic for PM. In fact, it is perhaps the most popular optimization method used for PM [1, 5, 6, 17]. Other heuristics have been applied to the problem as well, such as the random cost algorithm [7, 20] and microcanonical annealing [10, 22]. These techniques are (just as SA) local search techniques that have been enhanced with the ability to "escape" local optima, by using heuristics for accepting worse solutions than the current one. However, there has not yet been a single application of Evolutionary- or Population-based search techniques to the probe ordering problem of PM. This is in sharp contrast to other areas of bioinformatics (e.g. Protein-Structure Prediction [4] and Multiple Sequence Alignment [18]) where EAs frequently appear in the literature. "Sequencing by Hybridization" is one of the problems that is most related to PM and has been addressed by EAs as well [2].

In this paper, we investigate reasons for the absence of population based methods in PM and determine if the EA is a strong competitor to the previously used optimization techniques in PM. Might the PM problem-domain be of such a nature that EAs and related techniques do not show themselves superior to SA?

Before we proceed, we shall summarize some interesting previous research. Hsu [13] developed an algorithm that solved the PM problem in the error-free case and in the error-prone case within some given error-rate limitations. However, these limitations were not well-defined and

[1] The problem is equivalent to the Traveling Salesman Problem; see [8].

no tests of the algorithm's quality were conducted. Wilson et. al [21] analyzed how the distribution of probes along the DNA strand influenced the quality of the probe orderings that can be inferred from the data. However, this was analyzed only in the error-free case.

Because the hybridization data is often corrupted by errors, it is an important issue to identify the errors or at least determine suspicious areas. Harley [11] created a graph of the data, whose topology indicated the characteristics of the data: If the topology and data were coherent, the graph was without branches, but if parts of the data were anomalous, there were branches. In [20], a "confidence graph" was constructed based on bootstrap sampling. The probes were the nodes of the graph and they were linked by weighted edges. A weight close to 1 indicated that the probes were likely to be adjacent, whereas a value close to 0 indicated that they were not adjacent. This work was extended by Heber [12], to include the definition of a "confidence neighborhood" of probe orderings based on the graph.

2 Data generation and Measurement of Results

2.1 Simulator

To test our algorithms, we implemented a simulator based on [9]. By using a simulator we could parameterize the problem, and in a controlled way investigate different aspects of the problem domain as well as the performance of different algorithms on it. Furthermore, the "true" solution was known, making it possible to compare possible orderings to the optimal one, which is not always possible for experimental data. Many other simulations of the experimental process of PM have been made, and many variations regarding the details of their construction exist. It is difficult to assess the exact impact this might have on the results in the various investigations; we cannot rule out the possibility that the use of other simulators might change some of our conclusions to some extent, but this can be said for any simulation of the PM process, because there is no common agreement on how details of the data generation are defined.

The simulator had the following parameters: Number of probes (pr), coverage (cov), false positive rate (fp), false negative rate (fn), and chimerism rate ($chim$). The number of clones (cl) is given by $pr \cdot cov$, and the hybridization matrix size is $cl \times pr$. Note that in this context, the coverage is given by $cov = \frac{cl}{pr}$. It may also be defined as the sum of the clone lengths divided by the length of the DNA strand.

2.2 Fitness measures

We tested six different fitness measures. They were: f_0: Total number of fragments, f_1: Maximal number of fragments in a clone, f_2: Number of split clones, f_3: Inner fragment gap length, f_4: Maximum posterior probability, and f_5: Summed column distance. Functions f_0 to f_3 were taken from [9], f_4 was taken from [1], and f_5 is a very commonly used measure (also denoted the "Hamming Distance" between probes [10]). All fitness functions have the nice property in the error-free case that the true permutation yields a minimal fitness value. However, in the presence on errors, they can easily give a bad indication of the quality of a permutation – some more than others. In figure 1 the fitness functions have been calculated for an example hybridization matrix with the probes (columns) listed in their "true" order.

```
    0  2  4  6  8  10 12 14 16 18
0   X n . . . . . . . . . . n X X . . . .
1   X X . . . . . . . . . . . . . . . . .
2   . . . . . . . X X . . . . . . . . . .
3   . . . . . . . X X . . . . . . . . . .
4   . . . . . . X . . . . . . . . . . . .
5   . . . . . . . . . . . . . . . n . . .
6   . . . . . X . . . . . . . . . . . . p
7   . . X n X X . . . . . p . . . . . . .
8   . . . . . . . . . X X X X X X . . . . .
9   X X . . . . . . . . . . . . . . . . .
10  . . . . . . . . . . . . . . . . X . . .
11  X X X X X X . . . . . . . . . . . . .
12  . . . . . . X X . X X n X X X X . . . .
13  . . . . . . . . . X . . . . . . p . . .
14  X X . . . . . . . . . . . . . . . . .
15  . . . . . . X . . . . . . . . . . . .
16  . . . . . . . . . n . . . . . . . . .
17  . . p . . . . . X X . . . . . . . . .
18  . . . . . . . . . . X X X X X . . . . .
19  . . . . . . X . . . . . . . . . . . .
```

Figure 1: Example hybridization matrix generated with: $pr = 20,\ cov = 1,\ fp = 1\%,\ fn = 10\%,\ chim = 10\%$. An 'X' means hybridization, a 'p' means false positive, and an 'n' means false negative. Clones 0 and 12 are chimeric. Fitness function values: $f_0 = 26,\ f_1 = 3,\ f_2 = 6,\ f_3 = 44,\ f_4 = 36.08,\ f_5 = 46$

2.3 Quality Measures

Using simulated data, we could compare the discovered solutions with the *true* solution, and thus define an absolute measure for the quality of a solution (the degree of correctness). This measure is independent of the fitness function used. Two probes were considered *adjacent*, if they were placed next to each other in an ordering. The *adjacency correctness* was defined as the fraction of correctly predicted probe-adjacencies. This can be written as:

$$\textit{adjacency correctness} = \frac{adj_{cor}}{adj_{tot}}$$

where adj_{cor} is the number of true adjacencies found by the algorithm and $adj_{tot} (= pr - 1)$ is the total number of adjacencies in the problem. Having this absolute quality measure enabled us to compare the performance of different algorithms and fitness measures on a given PM problem type.

3 Methods

We implemented three optimization algorithms: Simulated Annealing (SA), an Evolutionary Algorithm (EA), and a simple Local Search strategy (LS).

3.1 Representation

A solution to the problem (i.e. an ordering) was naturally represented as a vector v of length pr, which contained all

the numbers from 0 to $pr-1$ (both included) once. That is, an ordering is a permutation of the integers in the interval $[0, pr[$. The i'th column of the solution matrix is the $v[i]$'th column of the problem matrix.

3.2 Algorithms

3.2.1 Local Search

The term LS can be defined as the class of algorithms in which a single solution, s, is maintained. The specific type of LS-algorithm that was used in this paper can be described as follows: The initial value of s was picked uniformly and at random from the given search space. A new solution s_{new} was generated by applying a neighborhood operator (see section 3.3.1) to s. s was replaced by s_{new}, if and only if s_{new} had a better fitness than s. These steps were iterated until a stop-criterion was met. In this study, the stop-criterion was simply a fixed number of iterations. Thus, the described method could be called an "improvement-only hillclimbing" method. However, in the following we shall for convenience just refer to it as "LS".

3.2.2 Simulated Annealing

SA [14] is (except for one important detail) the same algorithm as Local Search: If s_{new} has a worse (or equal) fitness than s, s_{new} might replace s anyway. The probability for such a replacement is given by $e^{\frac{a-b}{T}}$, where b is the fitness of s_{new}, a is the fitness of s, and T is the temperature, which is a parameter of the algorithm that is decreased over time.[2] Hence, the probability for accepting less fit versions of s_{new} decreases over time. In this study, the initial temperature T was set to 100, and it was reduced by 5% each 20 iterations, changing the search towards pure LS over time.

3.2.3 Evolutionary Algorithm

The EA [16] had a population of μ solutions. Firstly, by applying the neighborhood operator to each solution in the population in a cyclically manner and starting with the first solution, λ new solutions were generated. Secondly, δ new solutions were created by using a binary operator (crossover). Its first argument was a solution picked from the population in a cyclically manner and starting with the first solution, and its second argument was a random solution from the population. These $\lambda+\delta$ new solutions were put in a pool with the α best solutions from the old population, and from this pool the μ best solutions were selected to make up the new population.

By setting $\alpha = \mu$ we have a $(\mu+\lambda)$-strategy and by setting $\alpha = 0$ we have a (μ, λ)-strategy.

3.3 Operators

We have implemented a mutation operator, which is used by all three algorithms. In addition, we have implemented a crossover operator for the EA, taking two orderings as input and returning an ordering.

[2] We present the minimization version of the replacement probability.

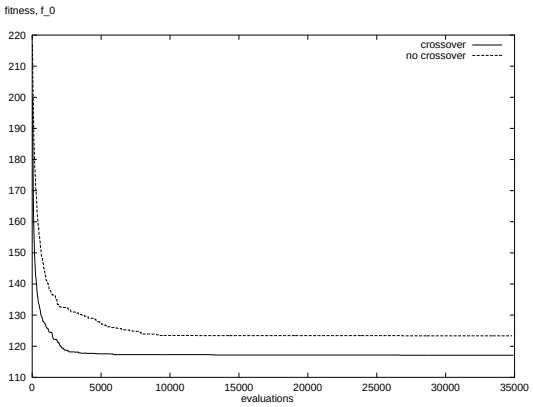


Figure 2: Performance of the EA with and without crossover (f_0, 30 runs, 20 probes, coverage 5, false positive-rate 0.01, false negative and chimerism rates 0.10).

3.3.1 Mutation/Neighborhood operator

The neighborhood operator used by all three algorithms performed with a probability of 0.25 a simple swap of two indices, and with a probability of 0.25 a reversal of the interval $[a, b]$, where $0 \leq a < b < pr$. Finally, with a probability of 0.50 it displaced the interval $[a, b]$ ($0 \leq a < b < n$) to some random new position (maximally a to the left and $n-1-b$ to the right, thus avoiding wrap arounds).

3.3.2 Crossover

The crossover operator searched through the first argument, vector $v1$, to find the adjacency pair $(v1[w], v1[w+1])$ that had hybridized to the least number of common clones (the weakest adjacency). The intension was to find two probes that were wrongly placed next to each other. First, the subvector $v1[0..w]$ was copied to the offspring vector o. Then, the remaining probes were copied to the offspring vector o in the order they appeared in $v2$. The first probe copied was the one that had most clones in common with $v1[w]$. Thus, the likely effect of crossover was that the weak adjacency from $v1$ would be replaced with a better one, while the offspring reflected sub-orders from both parents. The EA generally performed slightly better with crossover than without (see figure 2). However, this difference was not statistically significant because of a large spread of the results.

4 Problem Investigation

When an optimization algorithm does not find the "true" probe ordering in PM with respect to adjacency correctness, there can be different reasons for this:

1. The *algorithm* simply did not find the minimum of the fitness function.
2. The *fitness function* did not correlate with the problem (i.e. the "true ordering" did not give a minimal fitness value).

	$pr = 10$	$pr = 20$	$pr = 30$
coverage 1	64.3(14.1)	60.4(11.6)	55.9(7.8)
coverage 2	78.0(15.8)	74.7(11.3)	74.9(9.1)
coverage 3	88.2(10.2)	80.8(9.8)	79.7(8.2)
coverage 4	85.6(13.4)	82.4(10.6)	83.8(6.9)
coverage 5	87.8(12.4)	87.4(8.6)	84.7(8.5)
coverage 6	90.0(10.9)	88.2(8.4)	86.6(6.8)
coverage 7	87.4(12.3)	90.3(6.8)	89.9(6.9)
coverage 8	92.8(12.0)	91.5(8.3)	89.0(6.5)
coverage 9	93.7(9.0)	92.7(6.9)	90.6(5.6)
coverage 10	93.4(10.0)	93.5(7.1)	92.3(5.1)
coverage 15	95.1(8.4)	95.1(6.2)	93.4(6.7)
coverage 20	96.4(7.2)	96.4(4.9)	96.6(5.5)

Table 1: Average adjacency correctness for orderings with optimal fitness in the error-free case. Results are averages over 30 trials. Numbers in parentheses are standard deviations.

3. The *problem* was ambigous, meaning that many orderings (in addition to the "true" ordering) were optimal.

To shed light on these issues, we conducted 2 experiments. The first experiment investigated the ambiguity of the problem. The second experiment looked at the (lack of) correlation between a fitness function and the adjacency measure.

4.1 Ambiguity of Optimal Orderings on Error-free data

The procedure in this experiment was as follows: We started the algorithm, and ran it until it found an ordering with optimal fitness, and recorded the adjacency correctness. If the discovered ordering was not the true one, we knew the reason had to be found under point 3, problem ambiguity. This wass beacuse the algorithm found fitness minimum, and because the true ordering always gets minimal fitness in the error-free case. We used SA as optimization algorithm in this experiment, but have no reason to believe that results would be different, if we had used another algorithm to find optimal orderings. The results appear in table 1 for different values of cov and pr. As indicated by the table, the adjacency correctness is not dependent on pr.[3] However, table 1 shows that the measure strongly depends on the coverage. The discovered solutions are quite bad for low coverages. However, as the coverage increases, they tend to approximate the true orderings.

Experiments related to this have been conducted. Greenberg and Istrail [9] looked at the expected number of so-called weak adjacencies, and many studies have looked at the expected number of contigs (sets of clones that are "connected" by common probe-hybridizations) for given problems. By measuring the number of weak-adjacencies and contigs, we get an indication of the ambiguity of a class of PM instances. However, this experiment investigated and determined the expected best-case adjacency correctness — a number which is not possible to infer from earlier investigations.

[3] Higher values of pr were tested as well, but but did not deviate.

4.2 Fitness Function failure on Error-prone data

The second experiment sought to uncover to what extent the fitness function was responsible for non-optimal orderings on error-prone data. Thus, the experiment addressed point 2 from section 4. The procedure was: The algorithm was seeded with the true ordering and given 4000 evaluations. Then the adjacency correctness was recorded. Of course, in the error-free case, it would not be possible to improve the fitness of the true ordering, but when errors were present, it was nearly always possible to "optimize" on the true solution, because of lack of correlation between fitness and adjacency measures.

We used the maximum posterior measure [1] as the fitness function. Although this measure was designed to take errors into account, the algorithms easily found better solutions. The parameters took on these values: $pr = \{10, 20, 30, 40\}$, $cov = \{1, 5, 10, 15\}$, and fp, fn, and $chim = \{0, 0.01, 0.06, 0.10, 0.25\}$, and for each of the settings 30 trials were performed. In figures 3 to 6 we summarize the results for matrices generated using different settings of fp, fn, and $chim$. The graphs show the distribution of the results for the 16 settings of cov and pr for a given error-setting of $fpos$, $fneg$, and $chim$. For instance in figure 3, approximately 25% of the 16 settings with false positives rate 0.01 and no other errors gave an average adjacency correctness over 30 trials of less than 0.85 (marked with an arrow). We conclude that as the rate of false positives reaches 0.05 or more, then the expected best case adjacency correctness drops drastically (see figure 3). High rates of false negatives and chimeric clones do not disrupt the reliability of the results in the same manner (see figures 4 and 5). When the error types occour simultaneously they merely add-up (i.e. there is no extra synergy in the decrease of solution quality when errors are combined; see figure 6). Finally, we observed that high coverage and high dimensionality tend to increase the quality of the orderings, although this fact cannot be inferred from the figures.

These results are important to keep in mind, when evaluating the optimality of an ordering with respect to a given error-rate setting. Average adjacency correctnesses significantly higher that those found in this experiment are not likely.

5 Experiments

5.1 Experimental Settings

SA and LS were run for 30000 evaluations. The EA was run with these settings: $\mu = 6, \lambda = 12, \delta = 12, \alpha = 1$. This gives 24 evaluations/generation, thus 1250 generations were used to make 30000 evaluations.

The error-free case was tested with 20 and 50 probes and with a coverage of 5 and 15. The results can be found in table 2. The error-prone case was tested with 20, 40, and 60 probes, with a coverage of 5 and 10, and error-settings of

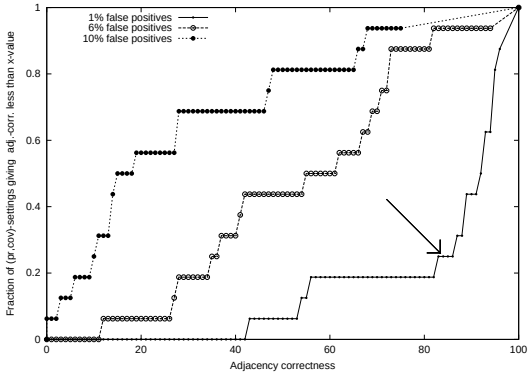


Figure 3: Distribution of average adjacency correctness obtained with the 16 (cov, pr) settings for varying false positive error rates.

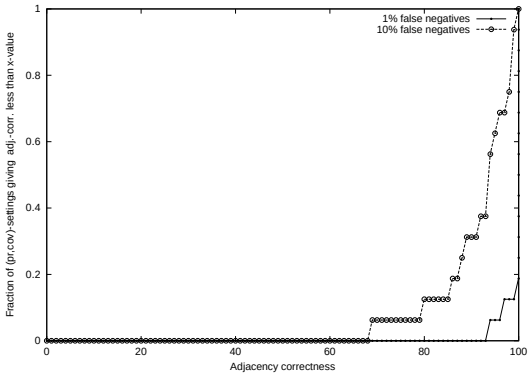


Figure 4: Distribution of average adjacency correctness obtained with the 16 (cov, pr) settings for varying false negative error rates.

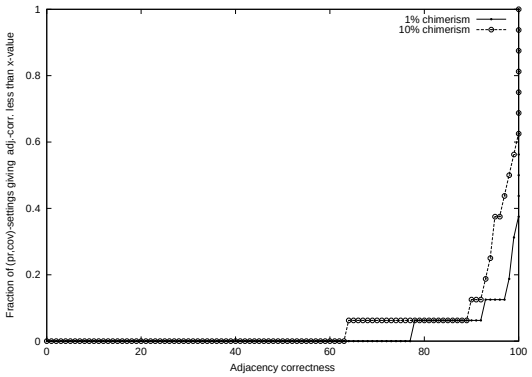


Figure 5: Distribution of average adjacency correctness obtained with the 16 (cov, pr) settings for varying chimerism error rates.

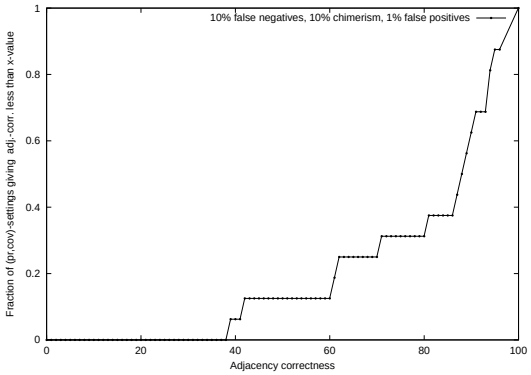


Figure 6: Distribution of average adjacency correctness obtained with the 16 (cov, pr) settings for the error rate $fp = 1\%,\ fn = 10\%,\ chim = 10\%$.

$fp = 1\%,\ fn = 10\%,\ chim = 10\%$. The results can be found in tables 3 and 4.

5.2 Experimental Results

5.2.1 Error-free case

There were 12 error-free problems. The SA had the best average fitness on six of these, the EA on five of them, and the LS on two of them (SA an LS shared best fitness on one problem). The EA and SA performed approximately equally well, with the LS being less efficient. However, LS outperformed SA on 4 of the 12 problems, and was not far behind on the other problems. Taking the relatively large standard deviations into account, the differences between the heuristics becomes even less apparent.

5.2.2 Error-prone case

In total there were 18 error-prone problems. Regarding best fitness, the SA got best fitness in 6 of them the EA in 9 of them, and LS in 4 of them (the EA and LS shared best fitness on one problem). Regarding best adjacency, SA found best ordering in 12 cases, and the EA in 6 cases. The SA and the EA seemed to perform equally well averaged over all the problems, with the EA being best for 20 probes and the SA being best for 60 probes. This is probably because the settings for the EA were determined based on tests with 20 probes.

Even though LS did not find the best ordering for any problem, it is striking how well this simple strategy performed. In almost all the problems it was worse than the SA – but only by a few percent! Actually, it was so close to the SA that on the problems where the SA beat the EA, the LS sometimes also beat the EA; this happened 6 times with respect to fitness and 9 times with respect to adjacency correctness. In this light, LS seems almost as strong as the EA. In conclusion, the results indicate that the three algorithms are roughly speaking equally strong. Especially when taking the standard deviations into consideration, the scores lie very close to each other.

	$pr = 20$	$pr = 50$
SA	83.9(11.1)	**84.4**(4.8)
cov 5, f_0, EA	**84.7** (10.6)	83.9 (6.4)
LS	83.3 (10.7)	82.2 (5.6)
SA	84.0(8.8)	**86.3**(4.6)
cov 5, f_3, EA	**88.2** (7.5)	80.9 (6.5)
LS	82.5 (8.7)	81.3 (5.6)
SA	**85.4** (8.4)	83.9 (4.7)
cov 5, f_4, EA	85.1 (11.3)	82.3 (5.7)
LS	84.7 (7.2)	**84.4** (5.3)
SA	**92.5**(6.9)	92.4(3.6)
cov 15, f_0, EA	91.4 (7.1)	**93.1** (5.0)
LS	91.8 (5.1)	92.5 (4.2)
SA	91.9(6.6)	**90.3**(4.1)
cov 15, f_3, EA	**93.7** (6.5)	88.4 (6.3)
LS	89.6 (9.0)	**90.3** (5.0)
SA	**93.2** (6.2)	91.2(4.7)
cov 15, f_4, EA	92.8 (7.9)	**93.1** (4.5)
LS	93.0 (6.2)	92.0 (4.2)

Table 2: Average best solutions found measured in adjacency correctness (averaged over 30 trials) by SA, EA, and LS using 3 different scoring functions on error-free hybridization matrices, varying pr and cov. Best results in bold.

Comparison of the fitness functions provides us another but very clear picture. Fitness function f_0 consistently provided the best adjacencies, f_4 was approximately 10 percentiles behind, and finally f_3 gave the worst results. The only exception in 18 problems was the problem with coverage 10 and 20 probes.

Finally, it is worth noticing that the correlation between the algorithm that got the best fitness for a given problem and the algorithm that got the best adjacency for the same problem, actually is very low! Only in 9 of 18 problems did the algorithm that found the best fitness, also find the best adjacency.

6 Discussion

The results demonstrate that the EA and SA perform equally well for the problem of PM. Furthermore, the simple LS strategy does a surprisingly good job. There could be several explanations for these findings.

First, the fitness landscape is very difficult to sample. The correlation between fitnesses of neighboring points is very low. Almost any kind of small modification for a given solution, can potentially drastically change its fitness. Thus, searching for good solutions is a bit like looking for a needle in a haystack; evaluation of two closely related solutions does not give a good picture of which "direction" to follow.

Another factor is the size of the neighborhood given by the operator, which contains a large number of orderings; namely all orderings that can be reached by swaps, translocations and reversals. As pr increases the neighborhood size increases drastically. Thus, it is not unlikely that the fitness landscape with respect to the operators used is without local

	$pr = 20$	$pr = 40$	$pr = 60$
SA	**118**(9)	285(15)	**475**(16)
cov 5, f_0, EA	121(10)	**281**(16)	477(17)
LS	121(10)	283(11)	484(20)
SA	112(31)	784(93.6)	2594(236)
cov 5, f_3, EA	**98**(29)	**777**(92)	2593(174)
LS	105(26)	783(107.6)	**2488**(247)
SA	143(32)	**504**(72)	**1168**(97)
cov 5, f_4, EA	**117**(27)	531(69.6)	1294(142)
LS	124(28)	524(64)	1181(98)
SA	246(15)	567(19)	980(27)
cov 10, f_0, EA	**239**(12)	**563**(21.5)	973(30)
LS	245(16)	**563**(26)	**968**(32)
SA	**226**(46)	1694(185)	5290(504)
cov 10, f_3, EA	227(32)	**1676**(183)	5418(459)
LS	232(43)	1722(182)	**5270**(389)
SA	292(67)	1092(111)	**2521**(154)
cov 10, f_4, EA	**266**(49)	**1074**(103)	2645(215)
LS	283(62)	1112(104)	2526(180)

Table 3: Average best solutions found measured by fitness (averaged over 30 trials) by SA, EA, and LS using 3 different scoring functions on error-prone hybridization matrices, varying pr and cov. Error-settings: false positives: 1%, false negatives: 10%, chimerism: 10%. Best results in bold.

	$pr = 20$	$pr = 40$	$pr = 60$
SA	78.4(8.2)	**77.1**(8.5)	**74.0**(6.4)
cov 5, f_0, EA	**81.4**(11.5)	74.5(8.5)	70.3(7.5)
LS	75.4(11.7)	75.7(7.9)	71.6(7.1)
SA	71.6(9.7)	**65.9**(7.6)	**57.9**(5.7)
cov 5, f_3, EA	**72.8**(12.7)	62.2(6.8)	48.8(6.2)
LS	71.2(11.5)	65.0(9.2)	56.8(5.0)
SA	77.9(11.8)	**71.1**(7.4)	**65.3**(6.2)
cov 5, f_4, EA	**78.2**(11.0)	69.1(7.6)	57.3(6.3)
LS	70.7(8.3)	67.7(5.5)	65.0(6.7)
SA	**84.7**(8.2)	**84.3**(7.3)	**81.8**(5.1)
cov 10, f_0, EA	81.8(10.1)	81.7(8.9)	75.0(5.0)
LS	82.5(8.9)	81.7(7.7)	80.7(6.8)
SA	**83.2**(10.8)	71.9(7.5)	**62.8**(5.4)
cov 10, f_3, EA	82.6(12.3)	**72.9**(7.3)	55.8(4.8)
LS	79.3(12.2)	71.6(7.0)	62.0(8.5)
SA	83.9(8.6)	78.4(8.9)	**71.5**(4.7)
cov 10, f_4, EA	**85.4**(9.9)	**78.9**(6.8)	64.7(7.5)
LS	79.1(10.0)	76.8(7.8)	67.4(6.7)

Table 4: Average best solutions found measured in adjacency correctness (averaged over 30 trials) by SA, EA, and LS using 3 different scoring functions on error-prone hybridization matrices, varying pr and cov. Error-settings: false positives: 1%, false negatives: 10%, chimerism: 10%. Best results in bold.

optima – the optimum can most likely be reached from any solution through a series of neighborhood operations with ever improving fitness for the temporary solutions. EAs are known to outperform SA on continous domains with many local optimal (multimodal problems) [19], where the search easily gets trapped and needs to escape local optima, but because of the huge neighborhood and the non-correlated neighbor points, it is very difficult to get trapped. However, this also makes it difficult to follow a good path.

We could have chosen a more restrictive neighborhood. However, the operators provided more efficient results than simpler operators such as swapping only, translocation only, or reversal only (data not shown). This is true for all the algorithms. Had we used a simple operator, the EA would have probably been the stronger technique (especially compared to LS), because of its ability to escape the local optima that would arise because of the smaller neighborhoods.

Analyzing the results actually pinpoints a central issue, which is much more decisive for the solution quality than choice of search heuristic: The fitness function. With the settings used in our error-prone experiments, we observed that f_0 was better than f_4, which again was far better than f_3. The differences between algorithms were negligible compared to the differences with different fitness functions. This might easily be changed if we had used other error-settings, or another simulator, or a real-life experiment. However, our conclusion is still clear and valid: The choice of fitness function can be much more important than the choice of algorithm for a class of problems. Our experiments make it very clear how important it is to have a model that fits the problem; if the model is inaccurate, the fitness function and consequently the search will suffer from this fact, and we risk solving the wrong problem.

In a real "problem solving" situation, much can be gained from adding various tricks to the search technique: Seeding with initial solutions based on a quick, greedy approach, followed by optimization of the outputted solution by some sort of exhaustive local search or fragment rearrangements [1]. We have deliberately made our investigation clear cut, to restrict the attention to questions regarding search methods, search space topology, and fitness functions. If we had applied various heuristics to improve the final solution, we would have "hidden" the results, such that the above insights could not have been gained.

Finally, we shall compare the results obtained by the three algorithms in this section to the expected average best-case adjacencies, which we calculated experimentally in section 4. In the error-free case we can e.g. look at the results with coverage 15, $pr = 20$, and using f_4. The calculated average best-case expected adjacency is 95.1% for these settings (table 1). The results obtained by the algorithms were 93.2% (SA), 93.0% (LS), and 92.8% (EA), so at least for the SA there is only 1.9% to improve on. Putting this number in perspective, this means that on average the SA in two of five runs found one adjacency less than what can be considered optimally possible (alone based on the ambiuity of the PM problem). In the three other runs it finds the average expected optimal adjacency correctness. In the error-prone case, we look at $pr = 20$, a coverage of 5, and using f_4. Here, the EA found an average adjacency correctness of 78.2%. In the experiment from section 4.2, the average adjacency correctness found when optimizing for 4000 evaluations using f_4 was 86.8%. Thus, in the error-prone case the theoretical upper bound for improvement is about 8%. These findings confirm the hypothesis that it is difficult to make huge improvements on the algorithms' performance. Further, we observe that a simple LS strategy seems to find solutions which are not far from the optimally achievable.

7 Conclusion

We have compared SA, EA, and LS on the PM probe-ordering problem. EA and SA performed equally well. LS performed marginally worse, but almost as well. The choice of optimization algorithm has little significance. A simple local strategy performs as well as more advanced search heuristics. This was attributed to the search-space that contains uncorrelated neighborpoints, making it hard to sample efficiently. Other matters complicate the search as well: Erroneous hybridization data and fitness measures having low correlation with the real quality of the orderings. We conclude that a better understanding of the PM problem structure is needed. This may allow for more efficient algorithms for error pruning, and design of better fitness functions, making the search more tractable and less random for optimization algorithms.

8 Future work

It would be interesting to look into the design of better fitness functions. Even though fitness function f_0 clearly gives the best results, the found orderings are far from perfect, and there is therefore still room for improvement.

One idea is to make a combination of normalized fitness functions. This would have the potential advantage that if one fitness measure suffers from an artifact in its measurement, this bias will hopefully be averaged out by the other measures that take part in the combined value, and hence the function will be more robust. Of course, one can fear that a combination of the measures only complicate the search and that the combined measure will be too noisy.

One could also imagine using consensus scoring, where the majority from a set of fitness functions is used to decide if a solution is of sufficient worth. This gives a more conservative type of measure. Again, this may make the search more robust, only accepting really good solutions, but we also risk "stopping" the search, as not all solutions can live up to the requirement that a majority of measures agree on its worth. However, the idea should be clear: Even the best of the fitness measures may be improved by combining it with elements from other measures.

Finally, more use of instance-specific knowledge can improve the results as well. Operator design is one area that could benefit from this. In particular, it could be interesting to look into more efficient crossover operators for the EA

that increase the rate of the convergence and improve the results the last few possible percent.

9 Acknowledgements

The author would like to thank the reviewers for useful remarks and Rene Thomsen for valuable discussions, suggestions, and comments.

Bibliography

[1] F. Alizadeh, R.M. Karp, D.K. Weisser, and G. Zweig. Physical mapping of chromosomes using unique probes. In *Symposium on Discrete Algorithms*, pages 489–500, 1994.

[2] J. Blazewicz, P. Formanowicz, M. Kasprzak, W.T. Markiewicz, and J. Weglarz. Dna sequencing with positive and negative errors. *Journal of Comp. Biol.*, 6:113 – 123, 1999.

[3] K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13:335– 379, 1976.

[4] D.E. Clark. *Evolutionary Algorithms in Molecular Design*. Wiley-VCH, Weinheim, 2000.

[5] A.J. Cuticchia. The use of simulated annealing in chromosome reconstruction experiments based on binary scoring. *Genetics*, 132(2):591–601, 1992.

[6] A.J. Cuticchia, J. Arnold, and W.E. Timberlake. ODS: ordering DNA sequences–a physical mapping algorithm based on simulated annealing. *Comput. Appl. Biosci.*, 9(2):215–219, 1993.

[7] J. Enkerli, H. Reed, A. Briley, G. Bhatt, and S.F. Covert. Physical Map of a Conditionally Dispensable Chromosome in Nectria haematococca Mating Population VI and Location of Chromosome Breakpoints. *Genetics*, 155(3):1083–1094, 2000.

[8] M.C. Golumbic, H. Kaplan, and R. Shamir. On the complexity of physical mapping. *Advances in Applied Mathematics*, 15:251 – 261, 1994.

[9] D.S. Greenberg and S. Istrail. Physical mapping by STS hybridization: algorithmic stratgies and the challenge of software evaluation. *J. Comp. Biol.*, 2(2):219–273, 1995.

[10] D. Hall, S. Bhandarkar, and J. Wang. ODS2: A Multiplatform Software Application for Creating Integrated Physical and Genetic Maps. *Genetics*, 157(3):1045–1056, 2001.

[11] E. Harley, A. Bonner, and N. Goodman. Revealing hidden interval graph structure in STS-content data. *Bioinformatics*, 15(4):278–285, 1999.

[12] S. Heber, J. Hoheisel, and M. Vingron. Application of bootstrap techniques to physical mapping. *Genomics*, 69(2):235 – 241, 2000.

[13] W.L. Hsu. On physical mapping algorithms - an error-tolerant test for the consecutive ones property. In *Computing and Combinatorics*, pages 242–250, 1997.

[14] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[15] J. Meidanis and J.C. Setubal. *Introduction to Computational Molecular Biology*. Springer, 1997.

[16] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1992.

[17] R.A. Prade, J. Griffith, K. Kochut, J. Arnold, and W.E. Timberlake. In vitro reconstruction of the Aspergillus (= Emericella) nidulans genome. *PNAS*, 94(26):14564–14569, 1997.

[18] R. Thomsen, G.B. Fogel, and T. Krink. A clustal alignment improver using evolutionary algorithms. In *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, volume 1, pages 121–126, 2002.

[19] J.S. Vesterstrøm and J. Riget. Particle swarms: Extensions for improved local, multi-modal, and dynamic search in numerical optimization. Master's thesis, EVALife, Dept. of Computer Science, Aarhus Universitet, 2002.

[20] Y. Wang, R.A. Prade, J. Griffith, W.E. Timberlake, and J. Arnold. A Fast Random Cost Algorithm for Physical Mapping. *PNAS*, 91(23):11094–11098, 1994.

[21] D.B. Wilson, D.S. Greenberg, and C.A. Phillips. Beyond islands (extended abstract): runs in clone-probe matrices. In *Proceedings of the first annual international conference on Computational molecular biology*, pages 320–329. ACM Press, 1997.

[22] Z. Xu, B. Lance, and C. Vargas. Mapping by Sequencing the Pneumocystis Genome Using the Ordering DNA Sequences V3 Tool. *Genetics*, 163(4):1299–1313, 2003.

Appendix E

A Comparative Study on Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms

The paper *A Comparative Study on Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms* has been presented on the *2004 Congress on Evolutionary Computation* (CEC2004) and published as a conference paper in the *Proceedings of the 2004 Congress on Evolutionary Computation* (CEC2004) [159].

A Comparative Study on Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms

Jakob Vesterstrøm
BiRC - Bioinformatics Research Center
University of Aarhus, Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark
Email: jve@birc.dk

René Thomsen
BiRC - Bioinformatics Research Center
University of Aarhus, Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark
Email: thomsen@birc.dk

***Abstract*— Population-based search methods such as evolutionary algorithms (EAs) and particle swarm optimization (PSO) can be used to sample large search spaces effectively and are among the most commonly used techniques for numerical optimization within the Evolutionary Computation community. During the last decade, several extensions to these algorithms have been suggested, offering improved performance on selected benchmark problems. Another search heuristic called differential evolution (DE) was introduced at the same time as PSO. Surprisingly, DE has remained partly 'unknown' although it has shown superior performance compared with other population-based methods in several real-world applications. In this paper we evaluate the performance of DE, PSO, and EAs regarding their general applicability as numerical optimization techniques. The comparison is performed on a suite of 34 commonly used benchmark problems. The results from our study show that DE generally outperforms the other algorithms, suggesting that it should be used as a first choice when encountering new optimization problems. However, when dealing with noisy functions, both DE and PSO are outperformed by the EA.**

I. Introduction

The Evolutionary Computation (EC) research community has shown a significant interest in optimization for many years. In particular there has been a focus on global optimization of numerical, real-valued 'black-box' problems for which exact and analytical methods do not apply. Since the mid-fifties many general-purpose optimization algorithms have been proposed for finding near-optimal solutions to this class of problems; most notably: Evolution Strategies (ES) [8], Evolutionary Programming (EP) [3], and Genetic Algorithms (GA) [6].

During the same period of time, many efforts have also been devoted to comparing these algorithms to each other. Typically, such comparisons have been based on artificial numerical benchmark problems. The goal of many studies was that of verifying that one algorithm outperformed another on a given set of problems. In general, it is possible to improve a given standard method within a restricted set of benchmark problems by making minor modifications to it.

Recently, Particle Swarm Optimization (PSO) [7] and Differential Evolution (DE) [11] have been introduced and particularly PSO has received an increased interest from the EC community. Both techniques have shown great promise in several studies based on real-world applications [4], [5], [12], [14]. However, to our knowledge, a comparative study of DE, PSO, and EAs on a large and diverse set of problems has never been made.

In this study, we investigated the performance of DE, PSO, and an EA on a selection of 34 benchmarks. The main objective was to examine whether one of the tested algorithms would outperform all others on a majority of the problems. Additionally, since we used a rather large number of benchmark problems, the experiments would also reveal whether the algorithms would have any particular difficulties or preferences.

Overall, the experimental results show that DE is far more efficient and robust (with respect to reproducing the results in several runs) compared to PSO and the EA. This suggests that more emphasis should be put on DE when solving numerical problems with real-valued parameters. However, on the noisy test problem, DE was outperformed by the other algorithms.

The paper is organized as follows. In Section II we introduce the methods used in the study: DE, PSO, and the EA. Further, Section III outline the experimental setup, parameter settings, and benchmark problems used. The experimental results are presented in Section IV. Finally, Section V contains a discussion on the results.

II. Methods

A. Differential Evolution

The DE algorithm was introduced by Storn and Price in 1995 [11]. Basically, it resembles the structure of an EA, but differs from traditional EAs in its generation of new candidate solutions and by its use of a 'greedy' selection scheme. DE works as follows: First, all individuals are randomly initialized and evaluated using the fitness function provided. Afterwards, the following process will be executed as long as the termination condition is not fulfilled (e.g. the current number of fitness evaluations performed is below the maximum number of evaluations allowed): For each individual in the population,

an offspring is created using the weighted difference of parent solutions. In this study we used the $DE/rand/1/exp$ scheme shown in Figure 1. The offspring replaces the parent if it is fitter. Otherwise, the parent survives and is passed on to the next iteration of the algorithm.

procedure create offspring $O[i]$ from parent $P[i]$ {
  $O[i] = P[i]$ // copy parent genotype to offspring
  randomly select parents $P[i_1]$, $P[i_2]$, $P[i_3]$,
    where $i_1 \neq i_2 \neq i_3 \neq i$
  $n = U(0, dim)$
  for ($j = 0$; $j < dim \wedge\ U(0,1) < CR$; j++) {
    $O[i][n] = P[i_1][n] + F \cdot (P[i_2][n] - P[i_3][n])$
    $n = (n + 1)$ mod dim
  }
}

Fig. 1. Pseudo-code for creating an offspring in DE. $U(0, x)$ is a uniformly distributed number between 0 and x. CR is the probability of crossover, F is the scaling factor, and dim is the number of problem parameters (problem dimensionality).

B. Particle Swarm Optimization

PSO was introduced by Kennedy and Eberhart in 1995. It was inspired by the swarming behavior as it is displayed by e.g. a flock of birds, a school of fish, or even human social behavior being influenced by other individuals [7].

Basically, PSO consists of a swarm of particles moving in an n-dimensional, real-valued search space of possible problem solutions. Every particle has a position vector $\vec{x}$ encoding a candidate solution to the problem (similar to the genotype in EAs) and a velocity vector $\vec{v}$. Moreover, each particle contains a small memory that stores its own best position seen so far $\vec{p}$ and a global best position $\vec{g}$ obtained through communication with its fellow neighbor particles. In this study we used the fully connected network topology for passing on information (see [15] for more details).

Intuitively, in PSO information about good solutions spreads through the swarm, and thus the particles tend to move to good areas in the search space. At each time step t, the velocity is updated and the particle is moved to a new position. This new position is calculated as the sum of the previous position and the new velocity:

$$\vec{x}(t+1) = \vec{x}(t) + \vec{v}(t+1)$$

The update of the velocity from the previous velocity to the new velocity is determined by the following equation:

$$\vec{v}(t{+}1) = \omega{\cdot}\vec{v}(t){+}U(0,\phi_1)(\vec{p}(t){-}\vec{x}(t)){+}U(0,\phi_2)(\vec{g}(t){-}\vec{x}(t)),$$

where $U(a, b)$ is a uniformly distributed number between a and b. The parameter ω is called the inertia weight [10] and controls the magnitude of the old velocity $\vec{v}(t)$ in the calculation of the new velocity, whereas ϕ_1 and ϕ_2 determine the significance of $\vec{p}(t)$ and $\vec{g}(t)$, respectively. Furthermore, at any time step of the algorithm v_i is constrained by the parameter v_{max}.

The swarm in PSO is initialized by assigning each particle to a uniformly and randomly chosen position in the search space. Velocities are initialized randomly in the range $[-v_{max}, v_{max}]$. When a particle exceeds the search space boundary, it is repositioned at the boundary and its velocity is set to zero. It will travel into the search space already in the next time step because of the attraction to $\vec{p}$ and $\vec{g}$.

C. Attractive and Repulsive PSO

The attractive and repulsive PSO (arPSO) [9], [15] was introduced by Vesterstrøm and Riget to overcome the problems of premature convergence [1] for PSO. The modification to the basic PSO scheme is that when the swarm diversity drops below a certain value d_{low}, the velocity update formula is modified to express repulsion of particles instead of the normal attraction. Thus, the velocity is updated according to:

$$\vec{v}(t{+}1) = \omega{\cdot}\vec{v}(t){-}U(0,\phi_1)(\vec{p}(t){-}\vec{x}(t)){-}U(0,\phi_2)(\vec{g}(t){-}\vec{x}(t))$$

This will increase the diversity over some iterations, and eventually when another value d_{high} is reached, the commonly used velocity update formula will be used again. Thus, arPSO is able to zoom out when an optimum has been reached, followed by zooming in on another hot spot, possibly discovering a new optimum in the vicinity of the old one. Previously, arPSO was shown to be more robust than basic PSO on problems with many optima [9].

The arPSO algorithm was included in this study as a representative for the large number of algorithmic extensions to PSO that try to avoid the problem of premature convergence. Other PSO extensions could have been chosen but we selected this particular one, since the performance of arPSO was as good (or better) as many other extensions.

D. Evolutionary Algorithm

In this study we used a simple EA (SEA) that was previously found to work well on real-world problems [13]. The SEA works as follows: First, all individuals are randomly initialized and evaluated according to a given fitness function. Afterwards, the following process will be executed as long as the termination condition is not fulfilled (e.g. the current number of fitness evaluations performed is below the maximum number of evaluations allowed). Each individual has a probability of being exposed to either mutation or recombination (or both). Mutation and recombination operators are applied with probability p_m and p_c, respectively. The mutation and recombination operators used are Cauchy mutation using an annealing scheme and arithmetic crossover, respectively. Further, individuals that were altered due to mutation or recombination are re-evaluated using the fitness function. Finally, tournament selection [2, section 2.3] comparing pairs of individuals is applied to weed out the least fit individuals.

The *Cauchy mutation* operator is similar to the well-known Gaussian mutation operator, but the Cauchy distribution has thick tails that enable it to generate considerable changes more frequently than the Gaussian distribution. The Cauchy distribution has the form:

$$C(x, \alpha, \beta) = \frac{\beta}{\pi\beta^2 + (x - \alpha)^2}$$

where $\alpha \geq 0, \beta > 0, -\infty < x < \infty$ (α and β are parameters that affect the mean and spread of the distribution). All of the solution parameters are subject to mutation and the variance is scaled with 0.1 $\times$ the range of the specific parameter in question.

Moreover, an annealing scheme was applied to decrease the value of β as a function of the elapsed number of generations t. α was fixed to 0. In this study we used the following annealing function:

$$\beta(t) = \frac{1}{1 + t}$$

III. Experiments

A. Experimental Setup and Data Sampling

The algorithms used for comparison were DE, PSO, arPSO, and the SEA. For all algorithms the parameter settings were manually tuned, based on a few preliminary experiments. The specific settings for each of the algorithms are described below. Each algorithm was tested with all of the numerical benchmarks shown in table I. In addition, we tested the algorithms on f_1–f_{13} in 100 dimensions, yielding a total of 34 numerical benchmarks. The termination criterion for each algorithm was set to 500,000 fitness evaluations for the 30-dimensional (or less) benchmarks and to 5,000,000 fitness evaluations for the 100-dimensional benchmarks. Each of the experiments was repeated 30 times with different random seeds, and the average fitness of the best solutions (e.g. individuals or particles) throughout the optimization run was recorded.

B. DE Settings

DE has three parameters: The size of the population ($popsize$), the crossover constant (CR), and the weighting factor (F). In all experiments they were set to the following values: $popsize = 100, CR = 0.9, F = 0.5$

C. PSO Settings

PSO has several parameters: The number of particles in the swarm ($swarmsize$), maximum velocity (v_{max}), the parameters for attraction towards personal best and the neighborhoods best (ϕ_1 and ϕ_2), and the inertia weight (ω). For PSO we used these settings: $swarmsize = 25,\ v_{max} = 15\%$ of the longest axis-parallel interval in the search space, $\phi_1 = 1.8,\ \phi_2 = 1.8,$ and $\omega = 0.6$.

Often the inertia weight is decreased linearly over time. The setting for PSO in this study is a bit unusual, because the inertia weight was held constant during the run. However, it was found that for the easier problems, the chosen settings outperformed the setting where ω was annealed. The setting with constant ω, on the other hand, performed badly on multi modal problems. To be fair to PSO, we therefore included the arPSO algorithm, which was known to outperform PSO (even with annealed ω) on multi modal problems [9].

D. arPSO Settings

In addition to the settings used in basic PSO, arPSO used the following parameter settings. d_{low} and d_{high} were set to $d_{low} = 0.000005$ and $d_{high} = 0.25$ in all experiments. The two parameters were only marginally dependent of the problem. These settings were also consistent with settings found and used in previous studies [9].

E. SEA Settings

The SEA used a population size of 100. Furthermore, the SEA used Cauchy mutation with an annealing scheme to control the variance and arithmetic crossover to create offspring from randomly chosen parents. The probability of mutating and crossing over individuals was fixed at p_m = 0.9 and p_c = 0.7, respectively. Tournament selection with a tournament size of two was used to select the individuals for the next generation. Further, we used elitism with an elite size of one to keep the overall best solution found in the population.

F. Numerical Benchmark Functions

For evaluating the four algorithms, we used a test suite of benchmark functions previously introduced by Yao and Liu [16]. The suite contains a diverse set of problems, including unimodal as well as multi modal functions, and functions with correlated and uncorrelated variables, respectively. Additionally, a few noisy problems and a single problem with plateaus has been included. The dimensionality of the problems originally varied from 2 to 30, but we extended the set with 100-dimensional variants to allow for comparison on more difficult problem instances. Table I shows all the benchmark problems used. Moreover, the ranges of the search space, problem dimensionality, and the respective global minimum fitness values and the corresponding solution in the search space are presented. We omitted f_{19} and f_{20} from Yao and Liu's study [16] because of difficulties in obtaining the definitions of the constants used in these functions (they were not given in [16]).

IV. Results

A. Problems of Dimensionality 30 or Less

The results for the benchmark problems f_1–f_{23} are shown in Table II and III. Moreover, Figure 2 shows the convergence graphs for a selection of the problems.

For functions f_1–f_4 there is a consistent performance pattern across all algorithms: PSO is the best, and DE is almost as good. They both converge exponentially fast toward the fitness optimum (resulting in a straight line when plotted with logarithmic y-axis). The SEA is magnitudes slower than the two other methods, and even though it eventually might converge toward the optimum, it takes hours for it to fine tune its solutions to a level that PSO and DE can reach in a few seconds. This analysis is illustrated in figure 2 (a).

On function f_5, DE is superior to both PSO and arPSO and the SEA. Only DE converges toward the optimum. After 300,000 evaluations, it commences to fine-tune around the optimum at an exponentially progressing rate. We may note that PSO performs moderately better than the SEA.

TABLE I

NUMERICAL BENCHMARK FUNCTIONS WITH A VARYING NUMBER OF DIMENSIONS (DIM). REMARKS: 1) FUNCTIONS SINE AND COSINE TAKE ARGUMENTS IN RADIANS. 2) THE NOTATION $(a)^T(b)$ DENOTES THE DOT PRODUCT BETWEEN VECTORS a AND b. 3) THE FUNCTION u AND THE VALUES y_i REFERRED TO IN f_{12} AND f_{13} ARE GIVEN BY $u(x,a,b,c) = b(x-a)^c$ IF $x > a$ AND $u(x,a,b,c) = b(-x-a)^c$ IF $x < a$ AND $u(x,a,b,c) = 0$ IF $-a \leq x \leq a$ AND FINALLY $y_i = 1 + \frac{1}{4}(x_i + 1)$. THE MATRIX a USED IN f_{14}, THE VECTORS a AND b USED IN f_{15}, AND THE MATRIX a AND THE VECTOR c USED IN f_{21}–f_{23}, ARE ALL DEFINED IN THE APPENDIX.

Function		Dim	Ranges	Minimum value
$f_1(\vec{x}) =$	$\sum_{i=0}^{n-1} x_i^2$	30/100	$-5.12 \leq x_i \leq 5.12$	$f_1(\vec{0}) = 0$
$f_2(\vec{x}) =$	$\sum_{i=0}^{n-1} \lvert x_i \rvert + \prod_{i=0}^{n-1} x_i$	30/100	$-10 \leq x_i \leq 10$	$f_2(\vec{0}) = 0$
$f_3(\vec{x}) =$	$\sum_{i=0}^{n-1} \left(\sum_{j=0}^{i} x_i \right)^2$	30/100	$-100 \leq x_i \leq 100$	$f_3(\vec{0}) = 0$
$f_4(\vec{x}) =$	$\max \lvert x_i \rvert, 0 \leq i < n$	30/100	$-100 \leq x_i \leq 100$	$f_4(\vec{0}) = 0$
$f_5(\vec{x}) =$	$\sum_{i=0}^{n-1} \left(100 \cdot (x_{i+1} - (x_i)^2)^2 + (x_i - 1)^2 \right)$	30/100	$-30 \leq x_i \leq 30$	$f_5(\vec{1}) = 0$
$f_6(\vec{x}) =$	$\sum_{i=0}^{n-1} \left(\lfloor x_i + \frac{1}{2} \rfloor \right)^2$	30/100	$-100 \leq x_i \leq 100$	$f_6(\vec{p}) = 0,$ $-\frac{1}{2} \leq p_i < \frac{1}{2}$
$f_7(\vec{x}) =$	$(\sum_{i=0}^{n-1} (i+1) \cdot x_i^4) + rand[0,1[$	30/100	$-1.28 \leq x_i \leq 1.28$	$f_7(\vec{0}) = 0$
$f_8(\vec{x}) =$	$\sum_{i=0}^{n-1} -x_i \cdot \sin(\sqrt{\lvert x_i \rvert})$	30/100	$-500 \leq x_i \leq 500$	$f_8(42\vec{0}.97) =$ 12569.5/41898.3
$f_9(\vec{x}) =$	$\sum_{i=0}^{n-1} (x_i^2 - 10\cos(2\pi x_i) + 10)$	30/100	$-5.12 \leq x_i \leq 5.12$	$f_9(\vec{0}) = 0$
$f_{10}(\vec{x}) =$	$-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2}\right) -$ $\exp\left(\frac{1}{n}\sum_{i=0}^{n-1}\cos(2\pi x_i)\right) + 20 + e$	30/100	$-32 \leq x_i \leq 32$	$f_{10}(\vec{0}) = 0$
$f_{11}(\vec{x}) =$	$\frac{1}{4000}\left(\sum_{i=0}^{n-1} x_i^2\right) + \left(\prod_{i=0}^{n-1} \cos(\frac{x_i}{\sqrt{i+1}})\right) + 1$	30/100	$-600 \leq x_i \leq 600$	$f_{11}(\vec{0}) = 0$
$f_{12}(\vec{x}) =$	$\frac{\pi}{n}\{10(\sin(\pi y_1))^2 +$ $\sum_{i=0}^{n-2}\left((y_i-1)^2(1+10(sin(\pi y_{i+1}))^2)\right) +$ $(y_n-1)^2\} + \sum_{i=0}^{n-1} u(x_i,10,100,4)$	30/100	$-50 \leq x_i \leq 50$	$f_{12}(\vec{-1}) = 0$
$f_{13}(\vec{x}) =$	$0.1\{(\sin(3\pi x_1))^2 +$ $\sum_{i=0}^{n-2}\left((x_i-1)^2(1+(sin(3\pi x_{i+1}))^2)\right) + (x_n-1)$ $(1+(\sin(2\pi x_n))^2)\} + \sum_{i=0}^{n-1} u(x_i,5,100,4)$	30/100	$-50 \leq x_i \leq 50$	$f_{13}(1,\ldots,1,-4.76)$ $= -1.1428$
$f_{14}(\vec{x}) =$	$\left(\frac{1}{500} + \sum_{j=0}^{24}(j+1+\sum_{i=0}^{1}(x_i - a_{ij})^6)^{-1}\right)^{-1}$	2	$-65.54 \leq x_i \leq 65.54$	$f_{14}(-3\vec{1}.95)$ $= 0.998$
$f_{15}(\vec{x}) =$	$\sum_{i=0}^{10}\left(a_i - \frac{x_0(b_i^2+b_i x_1)}{b_i^2+b_i x_2+x_3}\right)^2$	4	$-5 \leq x_i \leq 5$	$f_{15}(0.19,0.19,0.12,0.14)$ $= 0.0003075$
$f_{16}(\vec{x}) =$	$4x_0^2 - 2.1x_0^4 + \frac{1}{3}x_0^6 + x_0x_1 - 4x_1^2 + 4x_1^4$	2	$-5 \leq x_i \leq 5$	$f_{16}(-0.09,0.71)$ $= -1.0316$
$f_{17}(\vec{x}) =$	$(x_1 - \frac{5.1}{4\pi^2}x_0^2 + \frac{5}{\pi}x_0 - 6)^2 +$ $10(1-\frac{1}{8\pi})\cos(x_0) + 10$	2	$-5 \leq x_i \leq 15$	$f_{17}(9.42,2.47) = 0.398$
$f_{18}(\vec{x}) =$	$\{1+(x_0+x_1+1)^2(19-14x_0+3x_0^2-14x_1+$ $6x_0x_1+3x_1^2)\}\{30+(2x_0-3x_1)^2$ $(18-32x_0+12x_0^2+48x_1-36x_0x_1+27x_1^2)\}$	2	$-2 \leq x_i \leq 2$	$f_{18}(1.49\text{e-}05,1.00) = 3$
$f_{21}(\vec{x}) =$	$-\sum_{i=0}^{4}\left((x-a_i)^T(x-a_i)+c_i\right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{21}(\approx \vec{4}) = -10.2$
$f_{22}(\vec{x}) =$	$-\sum_{i=0}^{6}\left((x-a_i)^T(x-a_i)+c_i\right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{22}(\approx \vec{4}) = -10.4$
$f_{23}(\vec{x}) =$	$-\sum_{i=0}^{9}\left((x-a_i)^T(x-a_i)+c_i\right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{23}(\approx \vec{4}) = -10.5$

DE and the SEA easily find the optimum for the f_6 function, whereas both PSOs fail. This test function consists of plateaus, and apparently PSO has difficulties with functions of this kind. The average best fitness value of 0.04 for PSO comes from failing twice in 50 runs.

Function f_7 is a noisy problem. All algorithms seem to converge in a similar pattern, see Figure 2 (d). The SEA had the best convergence speed, followed by arPSO, PSO, and finally DE.

Functions f_8–f_{13} are highly multi modal functions. On all of them, DE clearly performs best and it finds the global optimum in all cases. Neither the SEA nor the PSOs find the global optimum for these functions in any of the runs. Further, the SEA consistently outperforms both PSOs, with f_{10} being an exception.

Both DE, the SEA, and arPSO come very close to the global optimum on f_{14} in all runs, but only DE hits the exact optimum every time, making it the best algorithm on this problem. PSO occasionally stagnates at a local optimum, which is the reason for its poor average best fitness.

On f_{15} both PSOs perform worse than DE and SEA. DE converges very fast to good values near the optimum, but seems to stagnate suboptimally. The SEA converges slowly, but outperforms DE after $\frac{1}{2}$ million fitness evaluations. To find out if the SEA would continue its convergence and ultimately reach the optimum, we tried to let it run for 2 million evaluations. In the majority of runs, it actually found the optimum after approximately 1 million evaluations (data not shown).

Functions f_{16}–f_{18} are all easy problems, and all algorithms are able to find near optimum solutions quickly. PSO converges especially fast. For some reason, the SEA seems to be able to fine-tune its results slightly better than the other algorithms.

The last three problems are f_{21}–f_{23}. Again, DE is superior to the other algorithms. It finds optimum in all cases. DE, the SEA, and PSO all converge quickly, but the SEA stagnates before optimum, and PSO even earlier. arPSO performs better than PSO, and is almost as good as the SEA.

B. Problems of Dimensionality 100

On f_1, PSO and DE have good exponential convergence to optimum (as in 30 dimensions) and the SEA is much slower. However, on f_2 the picture has changed. DE still has exponential convergence to optimum, but both PSOs fail to find the optimum – they are now performing worse than the SEA. The same pattern occurs for f_3 and f_4. This contrasts with the 30 dimensional case, where PSO performed exceptionally well for f_1–f_4.

On the difficult f_5 problem, DE is superior and finds optimum after 3.5 million evaluations. The other algorithms fail, the SEA being slightly better than basic PSO.

Both DE and the SEA find optimum on f_6 in all runs; moreover they find it very quickly. PSO only finds optimum in 9 of 30 runs. This is the reason for the average of 2.1 for PSO on this problem.

Apparently, the results for the 100 dimensional version of f_7 are similar to the results in the 30 dimensional case. The SEA has the best convergence, arPSO is slightly slower, followed by PSO, and finally DE.

For problems f_8–f_{13} the results also resemble those from 30 dimensions. One exception is that arPSO is now marginally worse than the SEA. In 100 dimensions the SEA is consistently better than both PSOs on these six problems.

V. Discussion

Overall, DE is clearly the best performing algorithm in this study. It finds the lowest fitness value for most of the problems. The only exceptions are: 1) The noisy f_7 problem, where the nature of the convergence of DE is similar, but still slower than that of all the other algorithms. Apparently, DE faces difficulties on noisy problems. 2) On f_{15} DE stagnates on a suboptimal value (and both PSOs even earlier). Only the SEA is able to find the optimum on this problem, which only has four problem parameters, but still appears to be very difficult.

We have not tried to tune DE to these problems. Most likely, one could improve the performance of DE by altering the crossover scheme, varying the parameters CR and F, or using a more 'greedy' offspring generation strategy (e.g. $DE/best/1/exp$).

DE is robust; it is consistently able to reproduce the same results over many trials, whereas the performance of PSO and arPSO is far more dependent on the randomized initialization of the individuals. This difference is particularly observable on the 100 dimensional benchmark problems. As a result, both versions of PSO must be executed several times to ensure good results, whereas one run of DE and the SEA usually suffices.

PSO is more sensitive to parameter changes than the other algorithms. When changing the problem, one probably need to change parameters as well to sustain optimal performance. This is not the case for the SEA and DE. The 100 dimensional problems illustrate this: The settings for both PSOs do not generalize to 100 dimensions, whereas DE and the SEA can be used with the same settings and still give the same type of convergence.

In general, DE shows great fine-tuning abilities, but on f_{16} and f_{17} it fails in comparison to the SEA. We have not determined why DE fails to fine tune these particular problems, but it would be interesting to analyze why.

Regarding convergence speed, PSO is always the fastest, whereas the SEA or arPSO are always the slowest. However, the SEA could be further improved with a more 'greedy' selection scheme similar to DE. Especially on the very easy functions f_{16}–f_{18}, PSO has a very fast convergence (3-4 times faster than DE). This may be of practical relevance for some real-world problems where the evaluation is computationally expensive and the search space is relatively simple and of low dimensionality.

To conclude, the performance of DE is outstanding in comparison to the other algorithms tested. It is simple, robust, converges fast, and finds the optimum in almost every run. In addition, it has few parameters, and the same settings can

TABLE II

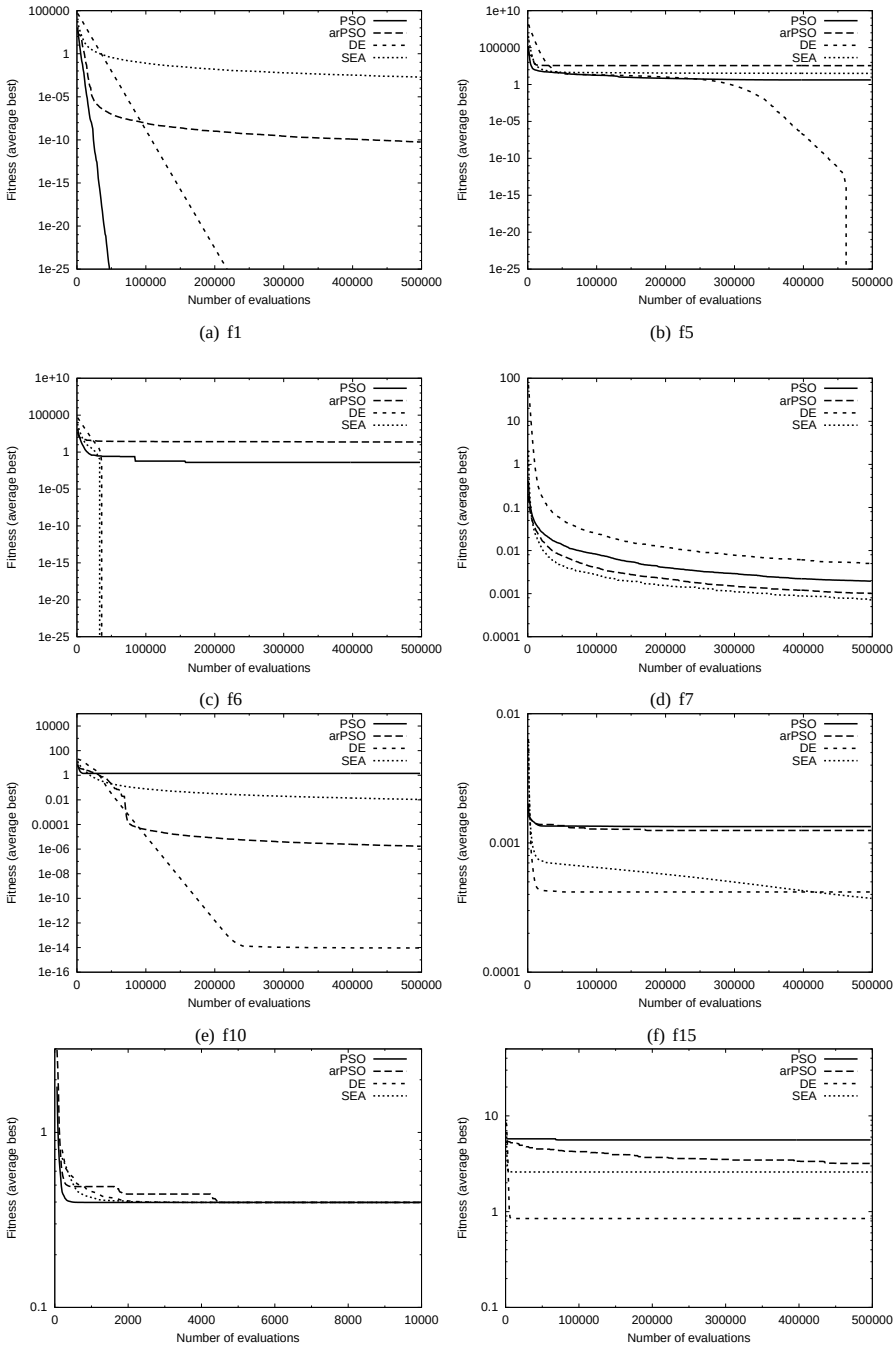
RESULTS FOR ALL ALGORITHMS ON BENCHMARK PROBLEMS OF DIMENSIONALITY 30 OR LESS (MEAN OF 50 RUNS AND STANDARD DEVIATIONS (STDDEV)). FOR EACH PROBLEM, THE BEST PERFORMING ALGORITHM(S) IS EMPHASIZED IN BOLDFACE.

Benchmark	**DE**		**PSO**		**arPSO**		**SEA**	
Problem	Mean	Stddev	Mean	Stddev	Mean	Stddev	Mean	Stddev
1	***0.0000000e+00***	0.00e+00	***0.0000000e+00***	0.00e+00	6.8081735e-13	5.30e-13	1.7894112e-03	2.77e-04
2	***0.0000000e+00***	0.00e+00	***0.0000000e+00***	0.00e+00	2.0892037e-02	1.48e-01	1.7207452e-02	1.70e-03
3	2.0200713e-09	8.26e-10	***0.0000000e+00***	0.00e+00	***0.0000000e+00***	2.13e-25	1.5891817e-02	4.25e-03
4	3.8502177e-08	9.17e-09	***2.1070152e-16***	8.01e-16	1.4183790e-05	8.27e-06	1.9827734e-02	2.07e-03
5	***0.0000000e+00***	0.00e+00	4.0263857e+00	4.99e+00	3.5509286e+02	2.15e+03	3.1318954e+01	1.74e+01
6	***0.0000000e+00***	0.00e+00	4.0000000e-02	1.98e-01	1.8980000e+01	6.30e+01	***0.0000000e+00***	0.00e+00
7	4.9390831e-03	1.13e-03	1.9082207e-03	1.14e-03	3.8866682e-04	4.78e-04	***7.1062480e-04***	3.27e-04
8	***-1.2569481e+04***	2.30e-04	-7.1874076e+03	6.72e+02	-8.5986527e+03	2.07e+03	-1.1669334e+04	2.34e+02
9	***0.0000000e+00***	0.00e+00	4.9170789e+01	1.62e+01	2.1491414e+00	4.91e+00	7.1789575e-01	9.22e-01
10	***-1.1901591e-15***	7.03e-16	1.4046895e+00	7.91e-01	1.8422773e-07	7.15e-08	1.0468180e-02	9.08e-04
11	***0.0000000e+00***	0.00e+00	2.3528934e-02	3.54e-02	9.2344555e-02	3.41e-01	4.6366988e-03	3.96e-03
12	***0.0000000e+00***	0.00e+00	3.8199611e-01	8.40e-01	8.5597888e-03	4.79e-02	4.5626102e-06	8.11e-07
13	***-1.1428244e+00***	4.45e-08	-5.9688703e-01	5.17e-01	-9.6263537e-01	5.14e-01	-1.1427420e+00	1.34e-05
14	***9.9800390e-01***	3.75e-08	1.1570484e+00	3.68e-01	9.9800393e-01	2.13e-08	9.9800400e-01	4.33e-08
15	4.1736828e-04	3.01e-04	1.3378460e-03	3.94e-03	1.2476701e-03	3.96e-03	**3.7041858e-04**	8.78e-05
16	-1.0316285e+00	1.92e-08	-1.0316284e+00	3.84e-08	-1.0316284e+00	3.84e-08	**-1.0316300e+00**	3.16e-08
17	3.9788735e-01	1.17e-08	3.9788736e-01	5.01e-09	3.9788736e-01	5.01e-09	**3.9788700e-01**	2.20e-08
18	***3.0000000e+00***	0.00e+00	***3.0000000e+00***	0.00e+00	3.5162719e+00	3.65e+00	***3.0000000e+00***	0.00e+00
21	***-1.0153201e+01***	4.60e-07	-5.3944733e+00	3.40e+00	-8.1809408e+00	2.60e+00	-8.4076288e+00	3.16e+00
22	***-1.0402943e+01***	3.58e-07	-6.9460507e+00	3.70e+00	-8.4352620e+00	2.83e+00	-8.9125580e+00	2.86e+00
23	***-1.0536412e+01***	2.09e-07	-6.7107552e+00	3.77e+00	-8.6155040e+00	2.88e+00	-9.7995696e+00	2.24e+00

TABLE III

RESULTS FOR ALL ALGORITHMS ON BENCHMARK PROBLEMS OF DIMENSIONALITY 100 (MEAN OF 50 RUNS AND STANDARD DEVIATIONS (STDDEV)). FOR EACH PROBLEM, THE BEST PERFORMING ALGORITHM(S) IS EMPHASIZED IN BOLDFACE.

Benchmark	**DE**		**PSO**		**arPSO**		**SEA**	
Problem	Mean	Stddev	Mean	Stddev	Mean	Stddev	Mean	Stddev
1	***0.0000000e+00***	0.00e+00	***0.0000000e+00***	0.00e+00	7.4869991e+02	2.31e+03	5.2291447e-04	5.18e-05
2	***0.0000000e+00***	0.00e+00	1.8045813e+01	6.52e+01	3.9637792e+01	2.45e+01	1.7371780e-02	9.43e-04
3	***5.8734789e-10***	1.83e-10	3.6666668e+03	6.94e+03	1.8174752e+01	2.50e+01	3.6846433e-02	6.06e-03
4	***1.1284972e-09***	1.42e-10	5.3121806e+00	8.63e-01	2.4367166e+00	3.80e-01	7.6708840e-03	5.71e-04
5	***0.0000000e+00***	0.00e+00	2.0203629e+02	7.66e+02	2.3609401e+02	1.25e+02	9.2492247e+01	1.29e+01
6	***0.0000000e+00***	0.00e+00	2.1000000e+00	3.52e+00	4.1183333e+02	4.21e+02	***0.0000000e+00***	0.00e+00
7	7.6640871e-03	6.58e-04	2.7845728e-02	7.31e-02	3.2324733e-03	7.87e-04	***7.0539773e-04***	9.70e-05
8	***-4.1898293e+04***	1.06e-03	-2.1579648e+04	1.73e+03	-2.1209102e+04	2.98e+03	-3.9430820e+04	5.36e+02
9	***0.0000000e+00***	0.00e+00	2.4359139e+02	4.03e+01	4.8096522e+01	9.54e+00	9.9767318e-02	3.04e-01
10	***8.0232117e-15***	1.74e-15	4.4934316e+00	1.73e+00	5.6281044e-02	3.08e-01	2.9328603e-03	1.47e-04
11	***5.4210109e-20***	0.00e+00	4.1715080e-01	6.45e-01	8.5311042e-02	2.56e-01	1.8932321e-03	4.42e-03
12	***0.0000000e+00***	0.00e+00	1.1774980e-01	1.75e-01	9.2199219e-02	4.61e-01	2.9783067e-07	2.76e-08
13	***-1.1428244e+00***	2.74e-08	-3.8604485e-01	9.47e-01	3.3010679e+02	1.72e+03	-1.1428100e+00	2.41e-08



(a) f1 (b) f5

(c) f6 (d) f7

(e) f10 (f) f15

(g) f17 (h) f21

Fig. 2. Average best fitness curves for selected benchmark problems. All results are means of 30 runs.

be used for many different problems. Previously, the DE has shown its worth on real-world problems, and in this study it outperformed PSO and EAs on a large set of real-valued, numerical benchmark problems as well. Therefore, it can rightfully be regarded an excellent first choice, when faced with a new optimization problem to solve.

ACKNOWLEDGMENTS

The authors would like to thank Wouter Boomsma for proofreading the paper. Also, we would like to thank the colleagues at BiRC for valuable comments on early versions of the manuscript. This work has been supported by the Danish Research Council.

APPENDIX

f_{14} :

$$a = \begin{pmatrix} -32,-16,0,16,32,\ldots,-32,-16,0,16,32 \\ -32,\ldots,-16,\ldots,0,\ldots,16,\ldots,32,\ldots \end{pmatrix}$$

f_{15} :

$$a = (0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246)$$

$$b = \left(\tfrac{1}{0.25}, \tfrac{1}{0.5}, \tfrac{1}{1}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{6}, \tfrac{1}{8}, \tfrac{1}{10}, \tfrac{1}{12}, \tfrac{1}{14}, \tfrac{1}{16}\right)$$

f_{21}–f_{23} :

$$a = \begin{pmatrix} 4.0 & 4.0 & 4.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \\ 8.0 & 8.0 & 8.0 & 8.0 \\ 6.0 & 6.0 & 6.0 & 6.0 \\ 3.0 & 7.0 & 3.0 & 7.0 \\ 2.0 & 9.0 & 2.0 & 9.0 \\ 5.0 & 5.0 & 3.0 & 3.0 \\ 8.0 & 1.0 & 8.0 & 1.0 \\ 6.0 & 2.0 & 6.0 & 2.0 \\ 7.0 & 3.6 & 7.0 & 3.6 \end{pmatrix}$$

$$c = \left(0.1, 0.2, 0.2, 0.4, 0.4, 0.6, 0.3, 0.7, 0.5, 0.5\right)$$

REFERENCES

[1] P. J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pp. 601–610. Springer, 1998. Lecture Notes in Computer Science 1447.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[3] L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Callahan, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems: Proc. of the 2nd Cybernetic Sciences Symposium*, pp. 131–155. Spartan Books, 1965.

[4] Y. Fukuyama, S. Takayama, Y. Nakanishi, and H. Yoshida. A particle swarm optimization for reactive power and voltage control in electric power systems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, pp. 1523–1528. Morgan Kaufmann, 1999.

[5] D. Gies and Y. Rahmat-Samii. Particle swarm optimization for reconfigurable phase-differentiated array design. *Microwave and Optical Technology Letters*, Vol. 38, No. 3, pp. 168–175, August 2003.

[6] J. H. Holland. *Adpatation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[7] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Vol. 4, pp. 1942–1948, 1995.

[8] I. Rechenberg. *Evolution strategy: Optimization of technical systems by means of biological evolution*. Fromman-Holzboog, 1973.

[9] J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer – the arpso. Technical report, EVALife, Dept. of Computer Science, University of Aarhus, 2002.

[10] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *In Proceedings of the IEEE Conference on Evolutionary Computation*, pp. 69–73. IEEE Press, 1998.

[11] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, Berkley, 1995.

[12] R. Thomsen. Flexible ligand docking using differential evolution. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC-2003)*, pp. 2354–2361, 2003.

[13] R. Thomsen. Flexible ligand docking using evolutionary algorithms: investigating the effects of variation operators and local search hybrids. *Biosystems*, Vol. 72, No. 1-2, pp. 57–73, November 2003.

[14] R. K. Ursem and P. Vadstrup. Parameter identification of induction motors using differential evolution. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC-2003)*, pp. 790–796, 2003.

[15] J. S. Vesterstrøm and J. Riget. Particle swarms: Extensions for improved local, multi-modal, and dynamic search in numerical optimization. Master's thesis, EVALife, Dept. of Computer Science, Aarhus Universitet, 2002.

[16] X. Yao and Y. Liu. Fast evolutionary programming. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proc. 5th Ann. Conf. on Evolutionary Programming*, pp. 451–460, Cambridge, MA, 1996. MIT Press.

Appendix F

Flexible Secondary Structure Based Protein Structure Comparison Applied to the Detection of Circular Permutation

The paper *Flexible Secondary Structure Based Protein Structure Comparison Applied to the Detection of Circular Permutation* [158] has been has been accepted for publication in the *Journal of Computational Biology*.

Flexible Secondary Structure Based Protein Structure Comparison Applied to the Detection of Circular Permutation

Jakob Vesterstrøm[†]
and
William R. Taylor[‡]

† BiRC - Bioinformatics Research Center
University of Aarhus, Høegh-Guldbergs Gade 10, Bldg. 090
DK-8000 Aarhus C, Denmark

‡ Division of Mathematical Biology,
the National Institute for Medical Research,
Mill Hill, London NW7 1AA, UK.

Keywords:
protein structure comparison, circular permutation

November 14, 2005

Abstract

We present a novel method for the comparison of protein structures. The approach consists of two main phases: 1) An initial search phase where, starting from aligned pairs of secondary structure elements, the space of 3D transformations is searched for similarities, 2) A subsequent refinement phase where interim solutions are subjected to parallel, local, iterative dynamic programming in the areas of possible improvement. The proposed method combines dynamic programming for finding alignments but does not restrict solutions to be sequential. In addition, to deal with the problem of non-uniqueness of optimal similarities, we introduce a consensus scoring method in selecting the preferred similarity and provide a list of top-ranked solutions. The method, called FASE (Flexible structural Alignment from Secondary structure Elements), was tested on well-known data and various standard problems from the literature. The results show that FASE is able to find remote and weak similarities consistently using a reasonable run-time. The method was tested (using the SCOP database) on its ability to discriminate inter-fold pairs from intra-fold pairs at the level of the best existing methods. The method was then applied to the problem of finding circular permutations in proteins.

1 Introduction

1.1 Overview of Structure Comparison

During evolution, the structural features of proteins are more conserved than their corresponding sequences. This implies that a possible common evolutionary origin may in some cases only be revealed by comparison of the structures [1, 2]. More information may therefore be gained about the relationship of pairs of structures, and more generally about protein evolution and present day proteins, by analysing structures in addition to their sequences.

The number of known protein structures is increasing rapidly [3] and it has become necessary to develop methods for analysing and comparing them automatically, since it gets increasingly infeasible to do this "by eye" and to maintain structure classifications manually. Structural comparison is central to the organisation of databases such as SCOP[4], CATH[5] and FSSP[6], either as a tool to guide manual classification or to provide the metric used to classify structures.

Structural comparison has direct applications in many areas such as aiding in the prediction of protein function by structural similarity, evaluation of predicted structure models in CASP [7], and as a 'gold' standard for sequence alignment. It is crucial to be able to compare structures just as reliably, accurately, and routinely as we today compare sequences in order to unravel the information encoded in the protein structures [8]. Development of structural comparison methods can thus be seen as implementing the basic operators on the set of protein structures such as similarity, equality, contained-in, distance, and so forth – operators already existing for strings and hence their primary sequence.

Sequence and structure comparison are inherently different problems with structural comparison being less well-defined and more difficult [9]. Many formulations of structural alignment as a computational problem have been presented. Most have been based on the Root Mean Square Deviation ($RMSD$) that measures the deviation between two sets of points with given correspondences between residues. The measure can be applied directly to the evaluation of predicted or NMR models where the correspondences between all residues of the structures are given. However, as the structures become dissimilar in size and shape, and when there is no obvious one-to-one residue correspondence, the $RMSD$ approach becomes increasingly problematic [10]. In such cases, the hard part is to find the subsets of matching residues. Specifying the size of the alignment with its $RMSD$ introduces a more informative measure but entails a trade-off between large alignment size and low $RMSD$. One approach is to set a maximum alignment distance, and

try to fit as many residue pairs under this constraint, using the number of included residues as an indicator of similarity [11]. Regardless of the approach taken, when using the combined $RMSD$/alignment size measure, it remains difficult deciding whether an alignment of, say, size 50 and $RMSD$ 2.0Å is better than an alignment of size 60 and $RMSD$ 2.2Å. Attempts to improve the $RMSD$/alignment size based measures include the use of gap-penalties [12], parameters for disregarding (or scaling down significance of) poorly aligned residues [13], and relating the alignment-size to the size of the two structures [14].

In addition to the $RMSD$ based formulations, structural comparison has also been stated as an optimisation problem over a contact map [15] or distance map [13] [16], as the comparison of normalised inter-residue vectors [17], or as the minimisation of the Area-C^{α} distance [18]. No single measure or formulation has proved superior in reflecting structural similarity as defined by accepted classifications, and no fast, reliable, and convergent method is yet available [8]. This illustrates, that it is just as difficult and important for the progress of the field to find and evaluate appropriate measures for structural similarity as it is to develop efficient algorithms for finding near-optimal solutions within a given problem definition.

1.2 Outline of the FASE Method

We describe a method called FASE (Flexible structural Alignment from Secondary structure Elements), which compares, aligns, and superposes two protein structures by searching for maximal similarities between them in a similarity-space. Our search-space is defined by superpositions of pairs of structures that have at least one pair of well-aligned secondary structure elements (SSEs) in common. We assume that an optimal superposition must have this property. The sought for optimality is required primarily in relation to the overall alignment of the structures (known as the architecture level in CATH) and not on the residue level. Although FASE puts emphasis on and uses roughly fifty percent CPU time on the residue alignment, fine tuning of residue correspondences is an issue that is much more ambiguous and debatable than the overall type of alignment discussed above. The success or failure on this point will have no relation to the general principle of the method on a global fold level scale, but only to the particular heuristics used for fine-tuning the alignment.

Methods for structural comparison can be distinguished by their ability to discover non-sequential alignments. In a sequential structural alignment, if any two aligned residue pairs, (r_i^A, r_j^B) and (r_k^A, r_l^B), are chosen from the alignment and $i < k$, then it is also required that $j < l$. In figure 1,

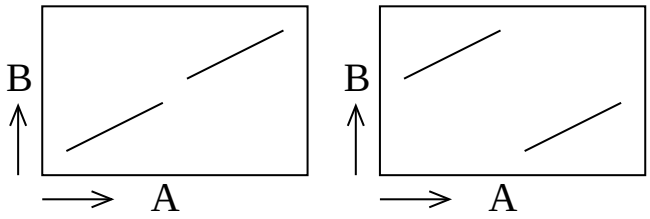


Figure 1: Two schematic alignments are shown. The left alignment possesses the sequentiality property, whereas the right does not, because increasing structure A indices does not correspond to increasing structure B indices.

two alignments have been plotted in a dot-plot like fashion to illustrate the concept of sequentiality and non-sequentiality. Some proteins are clearly related by a non-sequential alignment, which makes the ability to discover such alignments an attractive feature of any alignment algorithm designed to find similarities on a fold/architecture level.

The FASE approach has of two main stages: 1) An initial search phase in which the alignment of SSEs are starting-points for finding tentative alignments using a nearest neighbour alignment scheme, 2) A refinement phase where the solutions are subjected to iterative dynamic programming (DP) in unaligned areas. The method applies DP to several sub-alignments "in parallel". One single best solution is returned, but FASE also returns a ranked list of the best non-redundant solutions (where the granularity of redundancy may be specified by the user). The quality of the solutions is determined by a consensus scoring method that includes several measures from known methods (and some new ones) in order to improve consistency and robustness in its similarity assessment.

1.3 Relationship to Other Comparison Methods

The TOP method [14] is perhaps the method with most similarities to FASE. It utilises secondary structures and has two phases. There are, however, some important differences: TOP uses a mutually nearest (see section 2.1) requirement in alignment, which might prevent it from discovering some cryptic alignments (see e.g. figure 5). In FASE, the motivation to produce better alignments than the ones produced after the first phase using the nearest neighbour principle, prompted us to introduce the second phase where the alignments are refined using DP. Moreover, TOP uses a threshold for the percentage of matched secondary structures to decide further searching; this is a

rather sensitive measure and is difficult to set; if set too high, similarities are missed, and if set too low, the method runtime will dramatically increase. Finally, in the initial phase where secondary structures are compared, all pairs from the first structure are compared to all pairs from the second. This gives a time-complexity of $O(n^4)$. In FASE this problem is reduced by only aligning each SSE from the first structure in a fixed number of ways with every other SSE from the second structure, giving a time-complexity of $O(n^2)$.

MASS [19, 20] can find non-sequential alignments, and has resemblances with the present method and the TOP approach. As in TOP, the MASS method searches for pairs of matching SSEs from the two structures. This gives the same $O(n^4)$ time-complexity as TOP has. Regarding the residue alignment, MASS uses a linear-time method [21], which – as TOP – only makes use of the simplistic nearest neighbour scheme.

Recently a method called SSM was described which can find non-sequential similarities [22]. It begins with SSE matching followed by several heuristics to make the residue alignment. The SARF method [11] also finds sets of compatible SSEs using a heuristic for maximum clique discovery. Subsequently these sets are refined and extended on the residue level. SARF is also able to find non-sequential alignments.

Amongst the first methods published was SSAP [16] and later DALI [13]. Both approaches were based on alignment of distance matrices, and did not employ least squares superposition as a part of the comparison process. In contrast to SSAP and its successor, SAP [23], DALI was designed also to find non-sequential alignments. However, the only publicly available stand alone version of DALI, Dalilite [24], does not permit non-sequential alignments, nor does the web-based interface to DALI.

The GRATH program [25], an extension of the PROTEP [26] program, is able to find similar structures with respect to overall SSE architecture, that is, the fold/architecture level. Thus, in principle, non-sequential similarities should pose no problem to GRATH. However, the program only operates on the SSE level, and can therefore not give a residue alignment. LOCK [27] does not allow non-sequential residue alignments. In the newer version this feature should be present [28], but the web-based interface does not find non-sequential alignments (e.g. d1ca1_2 and d1p8ja1 from figure 11(c)). PRIDE [10] returns a value that indicates the degree of similarity, but does not make an alignment or describe where the similarities are located.

The PrISM system [29, 30, 31] also uses a two step approach to alignment. In the residue alignment process a DP global alignment procedure is made, which implies that the method cannot find non-sequential alignments. MAMMOTH [32] also requires sequential alignments. K2 [33] is based on a genetic algorithm that searches in the space of alignments. It was later

superseded by a version using simulated annealing. Gerstein [12, 34] used an iterative approach of alternating DP and superposition. Each iterative process is seeded with one out of 6 relative transformations of the structures to avoid getting trapped in a "local optimum". As with all standard applications of DP, the method enforces sequentiality in its solutions. The initial search phase of FASE is a generalisation of Gerstein's approach; instead of having only 6 starting points it takes the full step and aims at finding any interesting SSE arrangements by aligning all SSEs from structure A against all SSEs from structure B.

Chew [17] describes a method for detecting common substructures based on the Unit vector Root Mean Square (URMS) measure. The method finds *contiguous alignments*, which are simpler and more restricted than the standard *sequence alignment* (which further allows for insertions and deletions into a sequence). Thus, no non-sequentiality is allowed.

Methods based purely on SSEs include VAST [35] in which SSE is represented as a vector and all possible pairs of similar vectors from the two structures are found. Using the maximum clique detection algorithm, it finds a maximal subset of similar SSEs. The overall alignment score is based on the number of similar SSEs between the two structures. In a later version of VAST the alignment score was changed to reflect the number of aligned residues.

2 Methods

2.1 The Search Phase

Every SSE pair (one from each structure) of the same type formed the basis for a set of tentative solutions. Let a SSE pair $s = (s_A, s_B)$ be given, let $|s_A|$ and $|s_B|$ be the number of residues in each of the SSEs, and let x_A and x_B be the residue numbers of the first residue in each of them. For each SSE pair, $2 \cdot n_{off} - 1$ alignments were made by aligning $l = 4$ consecutive residues from the structures starting with residues f_A and f_B (see figure 2). Residue $f_A = x_A + (|s_A| + 1)/2 - l/2$ (the first aligned residue from structure A) was fixed in all alignments, while f_B was variable, taking on the values $f_B = x_B + (|s_B| + 1)/2 - l/2 + i$, for $i = -n_{off} + 1, \ldots, n_{off} - 1$. In figure 2 it is shown how the middle four residues of s_A are aligned with four middle residues of s_B (the case: $i = 0$) together with the four other alignments being made by displacing the four residues from s_B with one and two residues to both sides. In this way we can quickly and efficiently cover all the possible alignments of the structures where s_A and s_B are aligned.

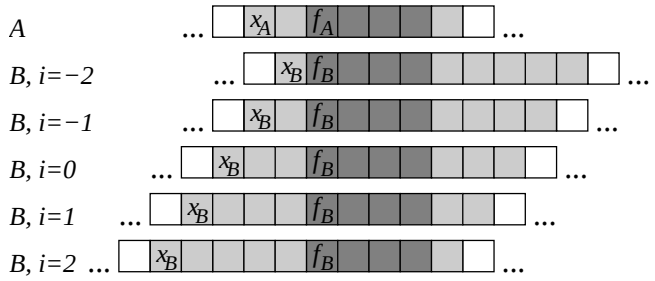


Figure 2: Illustration of alignments tried for structure A and B for two given secondary structures, s_A and s_B. The remaining values are $n_{off} = 3, l = 4, |s_A| = 7, |s_B| = 10$.

Using this strategy, a maximum of $|SSE_A| \cdot |SSE_B| \cdot (2 \cdot n_{off} - 1)$ initial alignments are made (some are discarded due to non-matching SSEs or similarity to previous ones). For each of them, the following iterative procedure was performed in order to conservatively extend and stabilise the alignments:

The structures were superposed using a standard least squares method based on the current alignment [36]. Initially, the current alignment consisted just of $l = 4$ consecutive matching residues from each structure. By selecting the pairs of residues from the two structures that were mutually nearest to each other under the superposition, a new alignment could be derived. Thus, we have gone from alignment to superposition, and from superposition to alignment (figure 3). Iteration was terminated when then alignments converged, more precisely when a cyclicity (of maximal length 10) was detected in the alignments. (The iterative process usually terminated naturally after 5-10 iterations, but an upper limit of 50 iterations was enforced.)

To avoid arriving at an alignment influenced by random residue pairings, in the i'th iteration any pairs not part of a continuous stretch of at least $\min(c, i)$ aligned residues were neglected (where the default value of c was 3).

2.2 Filtering Solutions for Refinement

A large number of tentative solutions was generated in the search phase, but only some of these were selected for further consideration as many were clearly unacceptable, perhaps matching only a single SSE pair with the remaining parts of the structures occupying different parts of space.

This selection was made by ranking the tentative solutions by the number

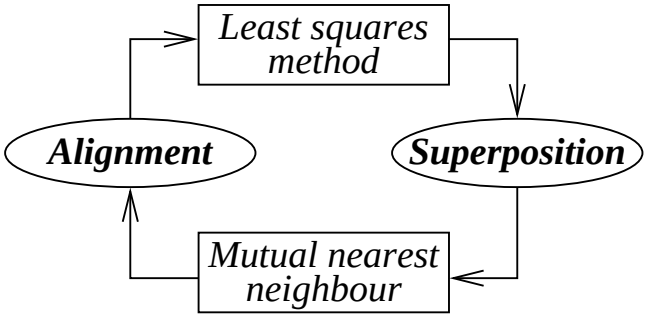


Figure 3: Illustration of alternation between alignment and superposition. An alignment leads to a superposition by a standard least squares minimisation methods, and a superposition leads to an alignment (in the search phase) by pairing of mutually nearest residues from the two structures.

of aligned residues (N) as first priority and by the root mean square deviation ($RMSD$) of the aligned residues as second priority. The resulting list was inspected starting with the top ranked solution. For a given value of N, only the six best scoring solutions (ranked by $RMSD$) were selected. The selection process terminated when solutions had less than 10 aligned residues, or a length less than 33% of the best solution.

2.3 Refinement Phase

Segments of residues falling between matched SSEs were aligned using DP (see section 2.4 for details). This approach, illustrated in figure 4, allows discovery of non-sequential alignments because the DP algorithm is only applied locally.

Let a tentative alignment be on the form $(s_1, \ldots, s_n)$, where s_i is a sub-alignment of length $l_i + 1$ of continuously selected residues, $(r^A_{p_i}, \ldots, r^A_{p_i} + l_i)$ from structure A and $(r^B_{q_i}, \ldots, r^B_{q_i} + l_i)$ from structure B. The sub-alignments define $(n+1)^2$ unaligned areas (at most) in the alignment matrix shown in figure 4. Each unaligned area a_{ij} was tested to see if the DP algorithm gave a positive (good) score that could indicate local structural similarity. If so, the sub-alignment was added to the overall alignment. After performing DP for all sub-matrices, it was tested if the new pairings caused any alignment conflicts. Any conflicts were resolved by choosing the pair of residues with the lowest distance. Moreover, residue pairs that were further apart than a specified upper limit (d_{max}) were subsequently removed. In addition, pairs

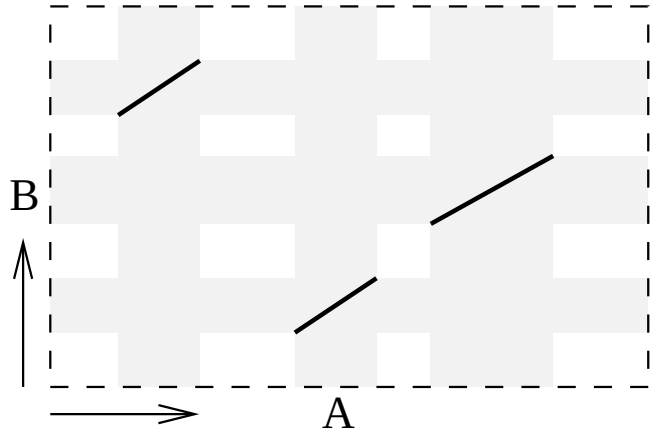


Figure 4: This hypothetic tentative alignment has three sub-alignments. In the shaded regions no new pairs can be aligned because of existing sub-alignments. However, in every white, unaligned region the algorithm attempts to align the residues by DP. In this example, DP will be tried in $4 \cdot 4 = 16$ regions and accepted if the DP-score is good enough.

where one (or both) residue(s) was not part of a consecutive stretch of at least c_{min} residues, were also removed from the alignment (as in the search phase). Between two iterations, the structures were re-superposed (as in the search phase) in order to update the input to the next phase. Thus, the sub-alignment would change between iterations (and possibly merge or split) influencing the number of sub-regions where DP was applied.

The four step alignment strategy of 1) DP, 2) conflict resolution, 3) weeding by distance, and 4) constraint of consequentially, was repeated iteratively until alignment convergence using the same principles for deciding this as in the search phase.

2.4 Dynamic Programming

We used the standard DP algorithm [37] in the refinement phase to align residues in the sub-areas that were defined by the already continuously aligned residue segments. That means that every execution of the DP algorithm was always performed under a fixed relative transformation (or superposition), T, between the structures. Entry (i, j) of the DP matrix M

was defined by:

$$M[i,j] = e^{-\frac{d^2}{10y}}$$

where $y = 1$ and d is a short notation for the distance between residue i in structure A and residue j in structure B under transformation T, $dist_T(r_i^A, r_j^B)$. However, if $d > d_{max}$, then $M[i,j]$ was set to 0.

DP with linear as well as with constant gap-penalty was tested. Linear gap-penalty was eventually preferred. One may note that a constant gap-penalty does not discriminate between small and large insertions. This might work well globally, because an optimal alignment on a global level might need to accommodate large insertions in order to reach global optimum. However, we used DP locally to align parts of the structures, and our focus was therefore in a greedy way to align as many pairs as possible. Tests showed (as one would expect) that the linear penalty was best to find the type of sub-alignments needed.

3 Implementation

3.1 Finding Nearest Neighbours

When finding mutually nearest neighbours during the search phase the following task was performed a large number of times: Under a given superposition (transformations T_A and T_B of the structures A and B), for every residue in structure A, its nearest neighbour in structure B had to be found (and vice versa). See figure 5 for an illustration.

Let r_A and r_B be the number of residues in structure A and B. It is straightforward to solve the task in time $O(r_A r_B)$ by simply calculating the distances between all $r_A r_B$ pairs. However, this was a core operation of the algorithm that had to be performed repeatedly, and therefore this gave an unsatisfactory time-complexity, especially as the structures increased in size.

Instead, a space-partitioning data-structure was used, also known as a quad-tree [38]. For simplicity, consider a simple hash-table and a structure B. For each 1Å cube in 3D, the hash-table stores all the residues from the structure that can be nearest neighbour to any point inside the given cube. Once calculated and initialised, such a data-structure can be used to find residue i in structure A's nearest neighbour (located in structure B) under the transformations T_A and T_B: Firstly, the transformation $T_B^{-1}T_A$ was applied to the original coordinates of residue i in structure A. This translates residue i from structure A into the original coordinate system for the residues from structure B. The residues associated with the 1Å cube in the structure

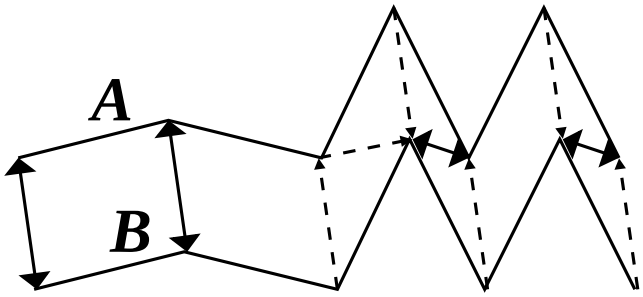


Figure 5: Mutual nearest neighbours. An arrow from A to B indicates that A has B as its nearest neighbour. There are seven residues in each of the substructures, but only four pairs of mutually nearest neighbours (marked with fully drawn lines).

B hash-table (where the transformed residue "lands") may now be checked to find the one that is actually nearest to residue i from structure A.

This approach, which "costs" one transformation, a hash-table query, and then calculating c distances (typically less than 10), is considerably faster than calculating all r_A (or r_B) distances for the given residue – in particular when structure sizes increase. Even for an average sized structure, the use of the efficient data-structure gives a factor 5 speed-up compared to the simple approach (data not shown). Assuming momentarily that the two structures are of equal size n, this reduces the time-complexity of nearest neighbourhood finding from $O(n^2)$ to $O(cn)$, where c is the average number of residues in a 1Å cube.

By decreasing the size of the cubes, the query speed of the hash-table can be improved (because c will decrease). However, this dramatically increases time required for building the hash-table: Every little 3D cube would require processing during creation of the table to find the residues to which it should point. We realised that the space had to be structured and checked in a more efficient manner than using a standard hash-table. To harvest the advantages of a fine granularity hash-table while avoiding a durable initialisation, a hierarchical space-partitioning hash-table like data-structure was used. This is a special case of a normal hash-table and can be used as such in terms of insertions and queries. It is much faster to construct and uses less memory, while having the same query speed.

First, a sufficiently large bounding box was first defined in 3D around the structure. It was then divided by a plane at the midpoint of its largest

dimension. For instance, the box $C = ((1,9),(12,17),(-4,3))$ would be split by the plane $x = 5$ along its x-dimension resulting in two new boxes: $C_1 = ((1,5),(12,17),(-4,3))$ and $C_2 = ((5,9),(12,17),(-4,3))$ respectively. Now, C_1 and C_2 could then recursively be divided. Since all residues by construction were inside the "root" box, they were all added as a possible nearest residue for a point in the root box and stored in the root box's near-list. Each time a box was divided (as with C above), the near-lists of its "children" were calculated by selecting a subset of the elements from the parent's near-list. The recursive division continued until the largest dimension of a box dropped below a fixed value (default was 2Å), or when the near-list of the box only contained a default value of two points.

A query into this data-structure started at the root and continued to select the child box in which the query point was contained until the current box had no children. At this point, the near-list was searched for the nearest neighbour. Because of the hierarchical structure used to divide 3D-space, the initialisation time and space usage are much lower than a standard hash-table. The 3D-space is not uniformly divided and the granularity is naturally adapted to the residue density of the region. This saves space and time compared to a normal hash-table.

3.2 Scoring of Similarity

A consensus score function based on five individual measures was used to rank the final set of solutions internally. As some of these measures have already been used in previous work, one could reasonably suppose they possessed both good intra-fold and inter-fold discrimination. However, a consensus approach adds an extra level of robustness. The five measures were the following:

- **S1** First priority: Alignment size, N, in percent of the number of residues in the smallest protein. Second priority[1]: $RMSD$
- **S2** Alignment size and $RMSD$ based score[2]: $\frac{3}{100}N - RMSD$
- **S3** Structal [39] score function: $\left(\sum_{i=1}^{N} \frac{20}{1+d_i/5}\right) - 10 \cdot (gapsA + gapsB)$
- **S4** GaFit [40] score function: $\sum_{i=1}^{N} c^2 - d_i^2$
- **S5** TOP [14] score function no. 1: $\frac{RMSD}{\left(\frac{N}{avg(|A|,|B|)}\right)^{1.5}}$

[1]The second criterion was only used when ranking solutions with equal N.

[2]Tests showed that comparisons of randomised structures almost never reached scores where $N > 100/(3 \times RMSD)$, implying that $3/100N - RMSD > 0$.

These five measures will be tested individually in the section 4.4.

3.3 A Measure for Alignment Linearity

A measure for alignment linearity (resemblance to the identity alignment, or the "straightness" of the alignment) was defined. Let the residues of structure A be named $r_1^A, \ldots, r_m^A$ and the residues of structure B be named $r_1^B, \ldots, r_n^B$. An alignment of A and B may be described as a list of residue-pairs. For instance an alignment of size 3 could be: $(r_1^A, r_2^B), (r_4^A, r_4^B), (r_7^A, r_8^B)$.

We defined a metric for the divergence of an alignment from linearity as the summed difference of the indices of each aligned pair. For example; in the alignment, $(r_1^A, r_1^B), (r_2^A, r_2^B), \ldots, (r_m^A, r_n^B)$, The sum (v) is: $0 + 0 + \ldots + (m - n)$. This difference was then normalised with the theoretical maximum divergence, $0.75 r_A^m r_B^n$. Finally the value v was modified to $1 - v$ such that the identity alignment would get 1 and the most deviant alignment would get 0.

3.4 A Measure for Circular Permutation

The result of a circular permuation (CP) of a gene is that the N- and C-terminal parts of the protein become exchanged [41]. If an alignment corresponds to a CP, it can be detected by scanning through the list of aligned residues and for each position, i, calculate the average of the structure B components to the left of i,

$$avg_L(i) = \frac{1}{i} \sum_{j=0}^{i-1} r_{alB(j)}^B,$$

and to the right of i,

$$avg_R(i) = \frac{1}{N - i} \sum_{j=i}^{N} r_{alB(j)}^B,$$

where $alB(j)$ refers to the residue number of the structure B component of the j'th aligned pair and N is the alignment size. If there is a CP the there will be an i for which $avg_L - avg_R$ is positive, and the index where $avg_L - avg_R$ is maximal (i_{max}) will match the beginning of structure B.

As we would like to measure the significance of the permutation, we also take into account the size of the two parts before and after the break. If, for example, structure A starts with matching the last five residues from

structure B, whereafter A starts to match the beginning of structure B, then the CP cannot be claimed to be very significant. Thus if we define

$$rearrscore_A(i) = \frac{avg_L(i) - avg_R(i)}{N_B} \cdot \frac{i}{N_A} \cdot \frac{N-i}{N_A}$$

where N_A and N_B are the sizes of structures A and B, we can define that $rearrscore_A = max_i(rearrscore_A(i))$. The final score for the degree of CP takes the degree of the permutation into account as seen from both structures, and we define it as:

$$rearrscore = 4 \cdot (rearrscore_A + rearrscore_B)$$

The $rearrscore$ has a theoretical maximum value of 1 (a perfect CP), and a minimum value of -1 (corresponding to a fully sequential alignment). Values less than 0 are truncated to 0. Significant matches for CPs can, typically, be detected at values from 0.25-0.3 and above. The maximum value is 1 because $rearrscore_A$ (and $rearrscore_B$) has a maximum of 0.125, which can be seen since its maximum value may be calculated as

$$\frac{0.5N_B}{N_B} \cdot \frac{0.5N_A}{N_A} \cdot \frac{0.5N_A}{N_A} = \frac{1}{8}.$$

This measure takes three factors info account: 1) The size of the alignment, 2) the partition of the permutation, and 3) the "pureness" of the permutation. The product $\frac{i}{N_A} \cdot \frac{N-i}{N_A}$ account for 1) and 2), whereas $\frac{avg_L(i)-avg_R(i)}{N_B}$ account for 3).

To further test the nature of the permutation, the "straightness" measure was applied locally to the two parts of the alignment as defined by the circularity. This enables us to distinguish a real CP alignment (figure 6, right) from one that just gets the same CP score but is not a CP (figure 6, left).

4 Experiments and Results

The FASE method was tested for its ability to reproduce results observed by other methods, to determine the significance of its output values and to show its ability to find CPs. These tests were performed on 2932 structures from SCOP40 database (release 1.65). SCOP40 (also known as ASTRAL) [42] is a subset of SCOP, in which no two structures have a sequence similarity higher than 40%. From this database, we selected the domains with a size between 100 and 250 residues, from here on called SCOP40s.

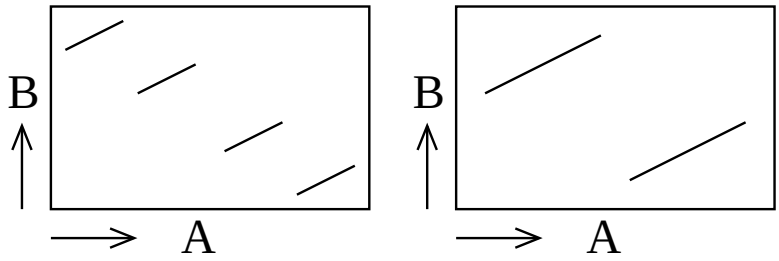


Figure 6: Two schematic alignments are shown. The right alignment might be a real CP, whereas the left is definitely not, but still gets an equal CP score. However, using the measure for straightness of the left and right halves, it is possible to make distinction.

Proteins		VAST	DALI	CE	ProSup	LGA	FASE
1bge-B	2gmf-A	71/2.3 **0**	94/3.3 **0**	107/3.9 **0**	87/2.4 **0**	91/2.5 **0**	89/2.54 **0**
1cew-I	1mol-A	75/2.0 **-1**	81/2.3 **0**	81/2.3 **0**	76/1.9 **1**	79/2.0 **0**	83/2.44 **0**
1cid	2rhe	78/2.0 **0**	96/3.1 **-1**	97/2.9 **1**	84/2.3 **0**	93/2.3 **0**	86/2.33 **0**
1crl	1ede	186/3.7 **-2**	212/3.6 **1**	219/3.8 **0**	161/2.6 **0**	182/2.6 **0**	205/2.95 **1**
1fxi-A	1ubq	48/2.1 **0**	52/2.5 **-1**	64/3.8 **0**	54/2.6 **-1**	61/2.6 **0**	55/2.12 **2**
1ten	3hhr-B	76/1.5 **0**	86/1.9 **0**	87/1.9 **0**	85/1.7 **0**	87/1.9 **0**	80/1.50 **0**
1tie	4fgf	76/1.6 **0**	114/3.1 **-1**	116/2.9 **1**	101/2.4 **-1**	104/2.3 **1**	110/2.83 **0**
2sim	1nsb-A	299/4.2 **0**	289/3.2 **0**	275/3.0 **0**	248/2.6 **0**	269/2.6 **0**	270/2.81 **0**
2aza-A	1paz	70/2.1 **0**	82/3.0 **-2**	84/2.9 **0**	82/2.6 **0**	80/2.2 **0**	88/2.74 **2**
3hla-B	2rhe	58/2.3 **0**	74/3.0 **0**	83/3.3 **0**	71/2.7 **-1**	74/2.5 **1**	69/2.36 **0**
sum		-3	-4	2	-2	2	5

Table 1: Comparison of structural comparison results on 10 structure pairs. For each comparison, the alignment size and $RMSD$ are provided. Numbers in bold are the number of other methods that perform worse minus the number of other methods that perform better on both measures on the same comparison.

4.1 Test Set of 10 Pairs

For comparison with existing methods, FASE was tested on a set of 10 pairs of structures (see table 1) taken from the literature [43]. Each row shows the results for a structure pair and each column shows the results for a method. The notation x/y means that an alignment of size x was found, having an $RMSD$ of y. The numbers in bold to the right are the number of other methods that performed worse minus the number of other methods that performed better on both measures on the same comparison.

From the table it is evident that FASE compares well with the other methods, in particular, there are no comparisons where another method finds lower $RMSD$ and higher N than FASE and there are five comparisons where FASE has a lower $RMSD$ and higher N than another method.

4.2 Randomised Models

The generation of randomised objects is a well-established method to determine the significance of a similarity score between two objects. The idea is well-known from sequence alignment, where Z scores are calculated by relating the raw scores to the density function of the raw comparison scores between random sequences. In structural alignment it is generally less clear what should be understood by a random structure.

Randomised structures (or 'decoys') were based on the SSE content of the original structure by performing a self-avoiding, compact random walk guided by the fold of the reversed backbones [44]. Thus, any structural similarities above the super-secondary structure level will be obscured. The input to the randomisation method was the secondary structure element definitions. 50 structures were selected, and for each of those 10 new models were generated. All against all comparisons were then made on the resulting set of 500 models.

4.2.1 Comparison With Real Structures

We compared the statistics for the comparisons on the randomised models and the real structures, on which the randomised models were based. In the topmost plot of figure 7 we have plotted the density functions for score function S1 (the percentage of aligned residues) for the comparisons of the random structures, SCOP structures, and SCOP structures from different families. The use of a log-scale on the y-axis emphasises the differences for the high-scoring comparisons (app. above 75). The density functions are somewhat similar below 75 – with the random comparisons shifted slightly towards lower values. However, above 75 the frequency of the random comparisons

drops compared to the SCOP comparisons. By discarding the SCOP comparisons from the same family, the densities match quite well, with the random comparisons having marginally more high scoring comparisons. Thus, when comparing two structures from different families, the expected similarity score (at least for the high values) can be quite well modelled by a comparison of randomised structures.

4.2.2 Circular Permutation in Decoy Structures

We can now ask whether the amount of CP in protein structures is higher than what one would expect to see by comparing random structures with the same degree of similarity. This may be checked by comparing the general similarity scores from the two sets. If these agree, then the CP scores should also agree, and if not, we may conclude that the CP content is not merely a random effect. Of course, given our knowledge of occurrences of CP that do exist, we would expect the CP content in SCOP to be above the random level.

The distributions for the CP scores were not identical (see figure 7, middle), in spite of their similarity score densities being roughly similar in the high scoring region. Thus, when scrambling up the structures, CP content is lost to a higher degree than the structural similarity is lost. This shows, that the amount of CP in the SCOP set is a real property of the set, and not just an artefact of structural constraints. To further support the above arguments, we plotted the similarity score versus CP score for the SCOP comparisons (see figure 7, bottom). This shows that to get CP values above 0.5 (the value from which they were "missing" from the randomised models), similarity values above 70 percent of matched residues are required. However, in this range the density of the randomised similarity scores is higher than the density for the real structures, so the lack of CP scores for the randomised models cannot be ascribed to a possible lack of general similarity in the range, which confirms our conclusion made above.

4.3 Fold Detection and Classification

One of the most important applications of structural comparison is the ability to decide if a pair of structures may be related through evolution or not. A suitable test for a structural comparison algorithm is fold classification; given two structures it must be decided if they belong to the same fold in SCOP. In the SCOP classification of protein structures the top level is called *class*, and below this level comes the *fold* level. The test is difficult because relatively remote similarities must be found, while still discriminating between

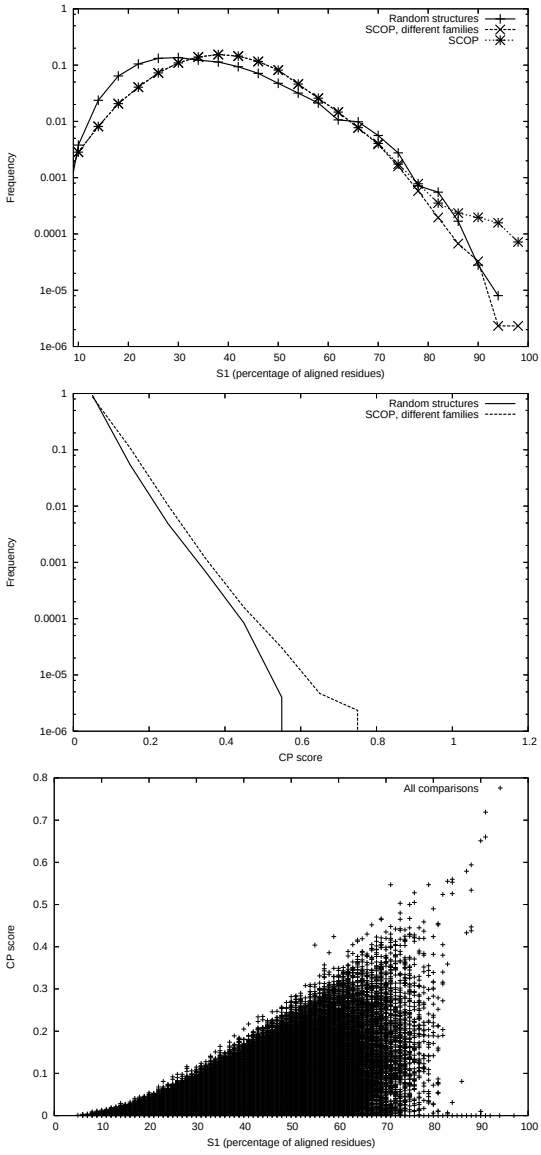


Figure 7: *Top*: Frequencies for the similarity scores (score function S1) for comparisons of randomised structures, of structures from SCOP, and structures from SCOP from different families. *Middle*: Frequencies for the CP scores for randomised structures and structures from SCOP from different families. *Bottom*: Correlation between similarity scores and CP scores.

folds. We also use this benchmark to derive a measure for the significance of the similarities. In the experiment we derive a measure for the problem of deciding if two structures are the same fold, but a similar methodology can be used to find the significance level for deciding other levels of similarity (given a classification of a set of structures as in SCOP).

In order to test FASE, we analysed the distribution of similarity scores for FASE for two groups of comparisons: Pairs of structures in the same fold, and pairs in different folds. The distribution of scores can be seen in figure 8 for both groups. An extreme value distribution,

$$f(x) = \frac{1}{b} e^{-e^{-(x-a)/b}}$$

was fitted to the distribution for the scores for different folds as in [32]. This makes it possible to decide if two structures are from the same fold in SCOP and to associate this answer with a probability of being true: Let us assume that FASE has compared the structures A and B and has returned the S5 score function value (TOP no. 1) S5 = 5.5. The probability of getting such a score or lower by comparing two structures from different folds is $F(5.5) = 1.04\%$, where $F(x) = \int_0^x f(x) = e^{\frac{-(x-a)}{b}}$ (the area under the density function f from 0 to x). Such a probability of false positive classification can also be reported as a Z-score. For the extreme-value distribution it holds that $\mu = a + \gamma b$ and $\sigma^2 = \frac{\pi^2}{6} b^2$, where γ is the Euler-Mascheroni constant (0.5772). Thus, using the same naming as above, we get $\mu = 13.29$ and $\sigma = 4.77$. The score S5 = 5.5 therefore gets a Z-value of $Z = \frac{5.5 - 13.29}{4.77} = -1.63$. A frequently used cutoff value for the Z-score for deciding significance is $Z = 2.0$ (see e.g. [13]). Using this threshold for fold classification would result in only accepting pairs with a S-score less than 3.8, corresponding to only classifying 0.07% of the structures from different folds into same fold (false positive rate).

As the frequency distributions overlap, it is impossible to make a perfect discrimination between true and false matches. The range of compromise can be pictured as a coverage-error plot (see figure 9) in which each line corresponds to one of the five similarity measures described above (section 3.2).

Further complicating the problem is the fact that the SCOP fold level does not fully correspond with the detection strategy of FASE which finds near maximal similarities between two structures with the same SSE layout – even if they have different SSE connectivity. In SCOP, structures in the same fold class must have identical SSE connectivity. Thus, FASE will have a built in tendency to detect "false positives" according to SCOP. Some might also arise because SCOP is created manually.

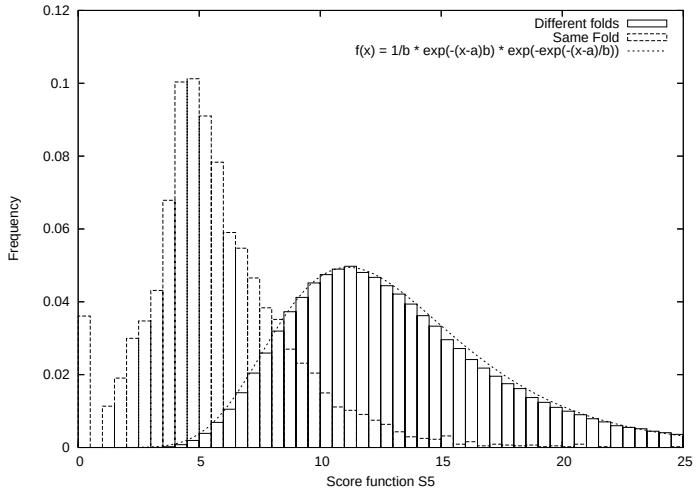


Figure 8: Frequency plots for the score function S5 (TOP no. 1) divided with respect to comparisons on structures in same fold and different folds ($a = 11.1435, b = 3.71623$).

4.4 Finding the Best Structural Similarity Measure

In the previous section we used the TOP score function no. 1 (S5) to measure the degree of similarity found between a structure pair in the significance analysis of FASE for same-fold detection. The measure gave, as can be seen from figure 9, the best coverage versus error rates. Another way to visualise this can be seen in figure 10. Here we have plotted the mean, standard deviation and minimum-maximum values for all 5 measures for each level of SCOP similarity. For instance, for x-value 0 we have structures not even in the same class, for x-value 2 we have structures in the same fold, for x-value 4 we get the analysis for comparisons of structures in same family (but not same protein-level in SCOP), and for x-value 7 we have self-comparisons.

A visual inspection reveals that the first graph (score function S1, percentage of matched residues) and the last graph (S5, TOP score no. 1) show the best segregation. For instance, only for measure S1 and S5 it holds that the intervals spanned by the standard deviations for levels 0 and 1 do not contain any mean-values from SCOP levels 2-7. Another useful property of the two measures is their assignment of a unique value to identical structures. Finally, there are two interesting observations to make: Firstly, measures S1 and S5 do not discriminate between SCOP levels 2 and 3 (while the other

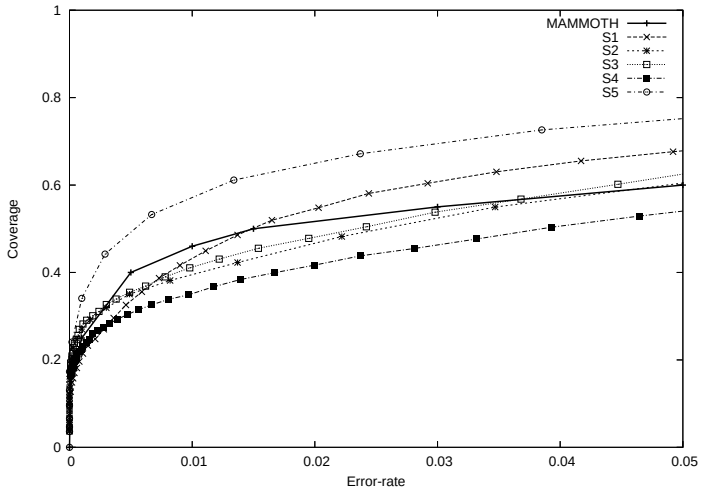


Figure 9: Coverage-error plots for all scoring schemes and MAMMOTH. A given point on a given line gives the classification ability for a given threshold setting. The x-axis is the error-rate, defined as the number of false positives per query. The y-axis is the coverage, defined as percentage of pairs from same fold that are discovered out of the total number of queries. For instance, for S5 we may notice that it is possible to get a coverage of 72% while only having a false positive rate of 3.8%. This is achieved with a threshold of S5 = 7. It is also clear from the figure that S5 outperforms MAMMOTH [32] overall and that S1 outperforms MAMMOTH if coverages above 50% are required.

measures show a very weak differentiation). Secondly, on average all measures find level 5 comparisons more similar than level 6 comparisons.

4.5 Detection of SCOP Circular Permutations

A substantial number of examples of naturally occurring CPs have been reported including concanavalin A, Bacterial β-glutanases, Swaposin, glucosyltransferases, transadolase, and C2 domains [45], FMN binding/ferredoxin [46]. If we want to establish evolutionary relationships between proteins then it is clear that a comprehensive approach cannot neglect CPs.

We can apply our measures to any solution to determine whether a similarity can be classified as a CP. Using FASE, we wanted to find CP similarities among the 8,596,624 all-against-all comparisons of the 2932 structures in the SCOP40s subset. All comparisons were made and the results analysed for the existence of CPs.

For each comparison, three alignments were recorded: 1) the best alignment found, 2) the best alignment found with a reasonable similarity to the identity alignment, and 3) the best alignment found with a reasonable degree of CP. The calculation of the measures for resemblance to the identity alignment and a CP alignment are described in the methods section. For each of these three alignments several properties of the alignments were logged: Rank amongst the solutions, alignment size (N), $RMSD$, number of aligned SSEs, sequence identity for the aligned residues, CP value, linearity value and the five similarity score values.

From the 8.6 million comparisons, a set of 1930 comparisons were selected of the most convincing CP pairs. The selection was based on four criteria: 1) The degree of which the best circularly permuted similarity was better than the best linear solution (measured in number of aligned residues), 2) the TOP score function no. 1 (S5), 3) the alignment size, and 4) CP score. The cutoffs used to select the 1930 comparisons were 5, 6, 60, and 0.3 (for 1–4, respectively). The first value of 5 ensures than the best CP alignment has at least 5 more residues that the best linear alignment. Further, we also restrict alignments to be at least of size 60. The two important values are the cutoffs for similarity score and CP score, having values 6 and 0.3. For structures in different folds, only 1.5% get a similarity score less than 6, and only 0.1% get a CP score higher than 0.3.

To find new CPs that were not annotated in SCOP, we checked if one of the structures or any of their family had any comment in the database regarding a CP. This excluded 109 pairs. Next, we wanted to find permutations between structures in different folds. By adding this restriction only 931 comparisons were left. The number of folds involved in these inter-fold

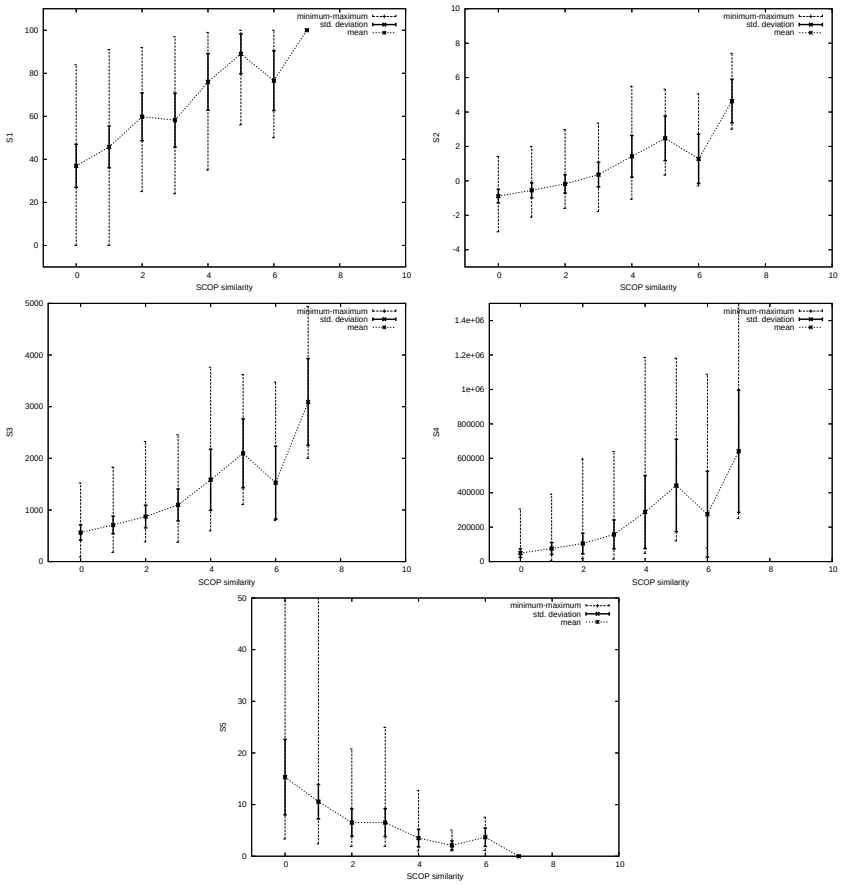


Figure 10: Analysis of mean ($*$), std. deviation (full-bar), and minimum-maximum (dashed-bar) values for five similarity measures split upon SCOP level of similarity (x-axis). For further explanation of the x-axis, see the text. The plots are: *top left* = **S1**, *top right* = **S2**, *mid left* = **S3**, *mid right* = **S4**, *bottom* = **S5**.

possible CP relationship were 104 (out of the total of 968 folds).

Are these CPs the result of evolutionary change or do they just occur by chance, perhaps caused by structural folding constraints? For instance, according to Copley *et al.* [47], a homology exists within many TIM-barrels despite the fact that, at first sight, it might look like a chance similarity based on its strong internal symmetry. Like the convergent/divergent evolution dilemma, this is not a simple matter to resolve as the symmetry itself may have been induced by CP.

4.6 Novel Circular Permutations

We inspected a selection of the many potential new circular permutations to assess whether the relationship between the proteins might have emerged as a result of internal symmetry or through ancient duplication/deletion events. The proteins included in table 2 present a selection of novel CPs ranging from the clearly trivial to those that cannot easily be explained by any simple symmetry in the structures. (Annotated as "non-trivial" in the table).

The relationships that are most likely to be based on 'chance' are those with fewer SSEs as these have higher symmetry and can therefore be transformed more easily by cyclic permutation to produce a match between two structures. Since the α-helix has a shorter length/residue ratio than the more extended β-strand, this effect is more common in the all-alpha proteins, ranging from the trivial transposition of two helices in a coiled-coil seen in the prefoldin/spectrin-like comparison (d1fxka_, d1qsda_) to the cyclic shift of a four helix-bundle by one helix position seen in the 4-helical-cytokine/Ferritin comparison (d1bgc__, d1nfva_). The effect is also obvious in the all-beta class with cyclic permutations possible in the up/down (porin-like) beta-barrels observed in the Streptavidin-like/OMPA-like comparison (d1jmxa5, d1qjpa_) and less obviously in the Alpha-toxin/furin comparison (d1ca1_2, d1p8ja1) in which a jelly-roll[3] fold is permuted (figure 11).

More complex relationships emerged in the β/α fold class. One involves two proteins (d1kjqa2, d1o97d2) with β-sheet strand order 32145 and 21345. At first sight, this does not appear to be a CP as strict permutation would require the last strand to reoccupy the first strand position giving: 21534. However, the solution found by FASE turns one protein through 180° giving 54123 which is a full CP. An approximation of the direct permutation is also seen in the Folylpolyglutamate synthetase comparison with a CheY-like protein (d1jbwa1, d1mb3a_) in which the Che-Y order 21345 shifts to (1)2534.

[3]The "jelly-roll" is a simple fold in which the backbone winds like a double-helix through two β-sheets. In figure 11a, this double-helix starts at the top-right of the diagram and winds in a right-handed helix through $1\frac{1}{2}$ turns to the termini (bottom-left).

SCOP codes	N/RMSD	ID	CP	proteins	comments
d1bgc__, d1nfva_	117/1.98	3	0.328	4-helical-cytokine/Ferritin	4-helix bundle helix shift
d1jmxa5, d1qjpa_	98/1.82	12	0.376	Streptavidin-like/OMPA-like	barrel hairpin shift
d1fxka_, d1qsda_	72/1.39	11	0.467	prefoldin/spectrin-like	trivial coiled coil shift
d1jbwa1, d1mb3a_	87/2.73	12	0.473	Folylpolyglutamate synth./CheY-like	non-trivial
d1jbwa1, d1mkza_	95/2.69	10	0.452	Folylpolyglutamate synth./MogA-like	non-trivial
d1ca1_2, d1p8ja1	103/2.74	7	0.412	Alpha-toxin/furin	jelly-roll permutation
d1f86a_, d1ncna_	81/3.10	9	0.405	Transthyretin/Immunoglobulin	non-trivial
d1fjgc2, d1dhn__	81/2.86	4	0.494	DHN aldolase/ribosomal S3	reconnected KH-like dom.
d1bj7__, d1nqna_	84/2.42	9	0.330	Lipocalin/Avidin	cyclic shift in barrel
d1eaza_, d1n9la_	50/1.94	6	0.202	PH-domain/PAS domain	non-trivial
d1cid_1, d1dcea2	78/2.84	8	0.518	Immunoglobulin/C2 domain-like	non-trivial
d1j5ya2, d1p1la_	63/2.79	7	0.000	HPr-like/Ferredoxin-like	just a large insert
d1bdfa1, d1jjcb4	74/3.31	6	0.434	RNA polymerase alpha/tRNA synth.	terminal strand switch
d1kjqa2, d1o97d2	77/2.37	14	0.377	PreATP-grasp/DHS-like	strand order 32145 to 21345

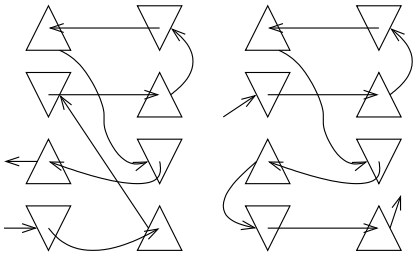
Table 2: **Novel circular permutations**. Protein pairs are identified both by their SCOP database codes and by a more trivial name under *proteins*. The number of aligned residues (N) and the $RMSD$ for the comparison is given ($N/RMSD$) along with the percent sequence identity over the alignment (ID) and the CP score (CP). A brief comment is included and further details for some pairs can be found in the text. A fuller analysis of the remainder will be published elsewhere (in preparation).

A more convincing direct permutation is seen in the Folylpolyglutamate synthetase comparison with a MogA-like protein (d1jbwa1, d1mkza_). Here the 5-strand sheet superposition is retained with a simple cyclic shift through two strands from the 12534 order to the 45312 order in MogA (figure 12).

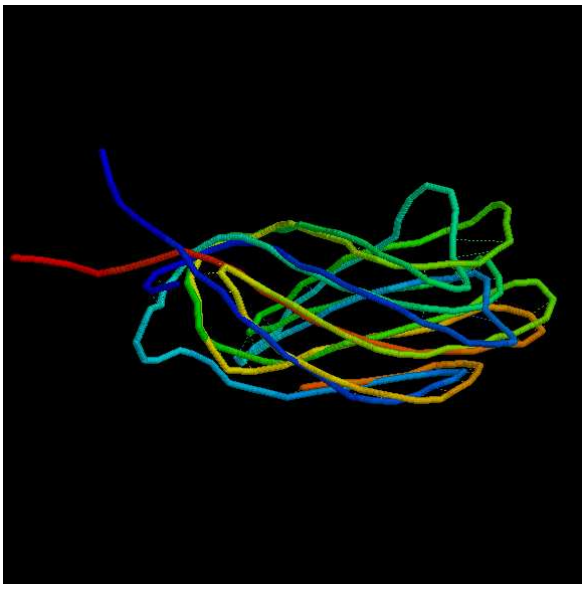
5 Conclusions

We have presented a new method for structural comparison of proteins called FASE that can find structural similarities even when these correspond to a non-sequential alignment of the sequences. This is an ability that most structural comparison methods do not possess. FASE also has some additional advantages:

- it is robust with default parameter settings and, thus, no tuning is necessary;
- it can find and detect semi-difficult residue alignments by local dynamic programming and is, thus, not restricted to mutually nearest neighbour alignments;
- it only performs $O(n^2)$ SSE alignments as opposed to $O(n^4)$ for similar methods;
- it uses a consensus scoring method for added robustness in the ranking of solutions.

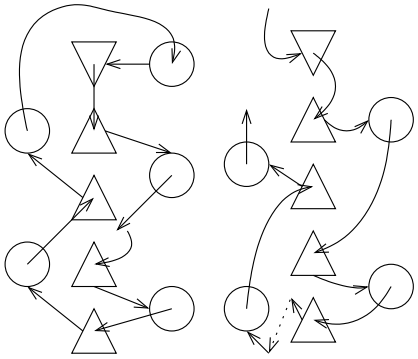


(a) d1p8ja1 (b) d1ca12

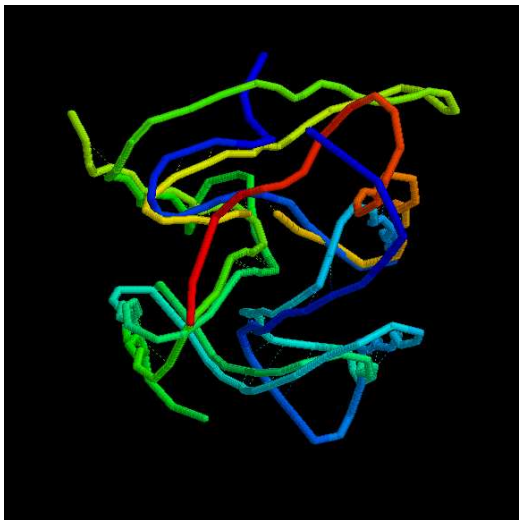


(c) d1cal on d1p8ja1

Figure 11: **Circular permutation in two all-beta proteins** Alpha-toxin and furin (d1ca1_2, d1p8ja1). (*a*) Topology representation of the Alpha-toxin fold. (*b*) Topology representation of the furin domain. (*c*) Superposition of the two structures using the alignment calculated by FASE. To make the transformation, the two extended termini in the alpha-toxin (left) should be joined and the cyan loop (lower left) should be deleted. The traces are coloured blue→red corresponding to the amino (N) to carboxy (C) direction of the backbone. The backbone has been slightly smoothed for clarity.



(a) d1mkza (b) d1jbwa1



(c) d1jbwa1 on d1mkza

Figure 12: **Circular permutation in two beta/alpha proteins**: folylpolyglutamate synthetase (domain 1) and the MogA-like protein MoaB which is a molybdenum cofactor biosynthesis protein (d1jbwa1, d1mkza_). (*a*) Topology representation of the folylpolyglutamate synthetase domain. (*b*) Topology representation of the MogA-like fold. (*c*) Superposition of the two structures using the alignment calculated by FASE. The molecules are orientated as in their topology diagrams and coloured blue→red for N→C.

FASE combined the DP algorithm with a search algorithm to find non-sequential similarities. This local application of the DP algorithm made it possible to use a simple implementation with a linear gap penalty that performed well.

FASE was tested on benchmark sets of structure pairs, and has been shown to be comparable with (and in some cases better than) existing structural comparison methods. This performance was confirmed when FASE was applied to the pairwise comparison of a wide set of proteins and its results compared to the classification found in the SCOP database. On the SCOP "fold" level, FASE performed well and even joined some folds by correctly identifying remote similarities that are difficult to detect. In addition, many structures known to be related by circular permutations were found and some new relationships were detected. FASE is available from `http://www.daimi.au.dk/~jve/`.

References

[1] Nick V. Grishin. Fold change in evolution of protein structures. *Journal of Structural Biology*, 134:167–185, 2001.

[2] S Sri Krishna and Nick V Grishin. Structural drift: a possible path to protein fold change. *Bioinformatics*, 21(8):1308–1310, Apr 2005.

[3] H M Berman, J Westbrook, Z Feng, G Gilliland, T N Bhat, H Weissig, I N Shindyalov, and P E Bourne. The Protein Data Bank. *Nucleic Acids Res*, 28(1):235–242, Jan 2000.

[4] Murzin A. G., Brenner S. E., Hubbard T., and Chothia C. Scop: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247:536–540, 1995.

[5] Pearl F.M.G, Lee D., Bray J.E, Sillitoe I., Todd A.E., Harrison A.P., Thornton J.M., and Orengo C.A. Assigning genomic sequences to cath. *Nucleic Acids Research*, 28(1):277–282, 2000.

[6] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[7] E. E. Lattman. Fifth meeting on the critical assessment of techniques for protein structure prediction. *Proteins*, 53(Supplement 6):333–595, 2003.

[8] Patrice Koehl. Protein structure similarities. *Current Opinion in Structural Biology*, 11:348–353, 2001.

[9] Michael L. Sierk and William R. Pearson. Sensitivity and selectivity in protein structure comparison. *Protein Sci*, 13(3):773–785, 2004.

[10] O. Carugo and S. Pongor. Protein fold similarity estimated by a probabilistic approach based on ca-ca distance comparison. *J. Mol. Biol.*, 315:887–898, 2002.

[11] N. Alexandrov. Sarfing the pdb. *Protein Engineering*, 9(9):727–732, 1996.

[12] Mark Gerstein and Michael Levitt. Using iterative dynamic programming to obtain accurate pairwise and multiple alignments of protein structures. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, pages 59–67. AAAI Press, 1996.

[13] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–138, September 1993.

[14] G. Lu. Top: a new method for protein structure comparisons and similarity searches. *J. Appl. Cryst*, 33:176–183, 2000.

[15] A. Caprara. Structural alignment of large-size proteins via lagrangian relaxation. In *RECOMB*, pages 100–108, 2002.

[16] W. R. Taylor and C. A. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208:1–22, 1989.

[17] L. Paul Chew, Dan Huttenlocher, Klara Kedem, and Jon Kleinberg. Fast detection of common geometric substructure in proteins. *Journal of Computational Biology*, 6(3-4):313–325, 1999.

[18] A. Falicov and F. E. Cohen. A surface of minimum area metric for the structural comparison of proteins. *Journal of Molecular Biology*, 258:871–892, 1996.

[19] O. Dror, H. Benyamini, R. Nussinov, and H. J. Wolfson. Mass: multiple structural alignment by secondary structures. *Bioinformatics*, 19(Suppl. 1):95–104, 2003.

[20] O. Dror, H. Benyamini, R. Nussinov, and H. J. Wolfson. Multiple structural alignment by secondary structures: Algorithm and applications. *Protein Science*, 12:2492–2507, 2003.

[21] O Bachar, D Fischer, R Nussinov, and H Wolfson. A computer vision based technique for 3-D sequence-independent structural comparison of proteins. *Protein Eng*, 6(3):279–288, Apr 1993.

[22] E Krissinel and K Henrick. Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions. *Acta Crystallogr D Biol Crystallogr*, 60(Pt 12 Pt 1):2256–2268, Dec 2004.

[23] W R Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Sci*, 8(3):654–665, Mar 1999.

[24] L Holm and J Park. DaliLite workbench for protein structure comparison. *Bioinformatics*, 16(6):566–567, Jun 2000.

[25] Andrew Harrison, Frances Pearl, Ian Sillitoe, Tim Slidel, Richard Mott, Janet Thornton, and Christine Orengo. Recognizing the fold of a protein structure. *Bioinformatics*, 19(14):1748–1759, 2003.

[26] H M Grindley, P J Artymiuk, D W Rice, and P Willett. Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm. *J Mol Biol*, 229(3):707–721, Feb 1993.

[27] A. P. Singh and D. L. Brutlag. Hierarchical protein structure superposition using both secondary structure and atomic representations. In *Intl. Conf. on Intelligent Systems in Molecular Biology*, pages 284–293, 1997.

[28] Jessica Shapiro and Douglas Brutlag. FoldMiner: structural motif discovery using an improved superposition algorithm. *Protein Sci*, 13(1):278–294, Jan 2004.

[29] A S Yang and B Honig. An integrated approach to the analysis and modeling of protein sequences and structures. I. Protein structural alignment and a quantitative measure for protein structural distance. *J Mol Biol*, 301(3):665–678, Aug 2000.

[30] A S Yang and B Honig. An integrated approach to the analysis and modeling of protein sequences and structures. II. On the relationship between sequence and structural similarity for proteins that are not obviously related in sequence. *J Mol Biol*, 301(3):679–689, Aug 2000.

[31] A S Yang and B Honig. An integrated approach to the analysis and modeling of protein sequences and structures. III. A comparative study of sequence conservation in protein structural families using multiple structural alignments. *J Mol Biol*, 301(3):691–711, Aug 2000.

[32] Angel R. Ortiz, Charlie E.M. Strauss, and Osvaldo Olmea. Mammoth (matching molecular models obtained from theory): An automated method for model comparison. *Protein Science*, 11:2606–2621, 2002.

[33] Joseph D. Szustakowski and Zhiping Weng. Protein structure alignment using a genetic algorithm. *Proteins: Structure, Function, and Genetics*, 38(4):428–440, 2000.

[34] M. Gerstein and M. Levitt. Comprehensive assessment of automatic structural alignment against a manual standard, the scop classification of proteins. *Protein Sci*, 7(2):445–456, 1998.

[35] J F Gibrat, T Madej, and S H Bryant. Surprising similarities in structure comparison. *Curr Opin Struct Biol*, 6(3):377–385, Jun 1996.

[36] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opts. Soc. Amer.*, 4:629–642, 1987.

[37] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[38] R. A. Finkel and J. L. Bentley. Quad-trees: a data structure for retrieval on composite keys. *Informatica*, 4:1–9, 1974.

[39] S. Subbiah, D. V. Laurents, and M. Levitt. Structural similarity of DNA-binding domains of bacteriophage repressors and the globin core. *Curr. Biol.*, 3(3):141–148, 1993.

[40] J. V. Lethonen, K. Denessiouk, A. C. W. May, and M. S. Johnson. Finding local structural similaries among families of unrelated protein structures: a genetic non-linear alignment algorithm. *Proteins: struct. funct. genet.*, 34:341–355, 1999.

[41] A Jeltsch. Circular permutations in the molecular evolution of DNA methyltransferases. *J Mol Evol*, 49(1):161–164, Jul 1999.

[42] S E Brenner, P Koehl, and M Levitt. The ASTRAL compendium for protein structure and sequence analysis. *Nucleic Acids Res*, 28(1):254–256, Jan 2000.

[43] A. Zemla. Lga - a method for finding 3d similarities in protein structures. *Nucleic Acids Research*, 31(13):3370–3374, 2003.

[44] W. R. Taylor and I. Jonassen. A structural pattern-based method for protein fold recognition. *Proteins, struct. funct. genet.*, pages 222–234, 2004.

[45] Y. Lindqvist and G. Schneider. Circular permutations of natural protein sequences: Structural evidence. *Curr. Opin. Struct. Biol.*, 7:422–427, 1997.

[46] R. B. Russell. Detection of protein three-dimensional side-chain patterns: New examples of convergent evolution. *J. Mol. Biol.*, 279:1211–1227, 1998.

[47] Richard R. Copley and Peer Bork. Homology among ([beta][alpha])8 barrels: implications for the evolution of metabolic pathways. *Journal of Molecular Biology*, 303(4):627–641, November 2000.

Appendix G

Detection and Evaluation of Novel CPs using FASE

The paper *Detection and evaluation of novel CPs using FASE* is in submission for a bioinformatics journal.

Detection and Evaluation of Novel Circular Permutations using FASE

Jakob Vesterstrøm
BiRC - Bioinformatics Research Center
University of Aarhus, Høegh-Guldbergs Gade 10, Bldg. 090
DK-8000 Aarhus C, Denmark

***Abstract*— Circular permutations (CPs) in proteins are studied to determine the role and extent of CPs in protein evolution, and to explore the usability of CPs as a tool in protein engineering. In this paper, a number of novel CPs are reported between pairs of protein domains. The pairs were found by pairwise structural comparisons of domains from the SCOP database (version 1.65) using the structural alignment program FASE. The similarities and differences between the structures are thoroughly described and analysed to assess the validity of the proposed relations. In most cases, the hypothesis that the domains are homologous and differ by a CP event, is supported by the analysis. In the analysis, the various similarity measures returned by FASE are considered, including sequence similarity after structural alignment (SSSA); also visual inspection of the superpositions is taken into account. The SSSA measure has in previous protein structure studies been proposed as a significant indicator of homology. Next, sub-trees from SCOP that are related by CPs (as annotated in SCOP) are analysed. This analysis shows that some of these sub-trees are surprisingly weakly related, while others can be related just as well or better by a simple identity alignment. The same analysis is then done for the sub-trees whose relation is implied by the novel CPs reported in this study. It is found that some of these sub-trees are indeed significantly related, thus suggesing homology between these sub-trees. Other novel CPs, however, are less similar and rather suggest a structural analogy between the pairs and sub-trees. A subsequent comparative analysis of the SCOP CP sub-trees and the novel CP sub-trees seemed to further confirm the validity of some of the novel CPs. Specifically, the two superfamilies *MurD peptides* and *CheY* are clearly related wrt. structural similarity, SSSA measure, CP measure, and lack of an alternative identity alignment of the structures. In fact, up to 50% of the SCOP CP sub-trees, are weaker related than the most plausible novel CP sub-trees.**

I. Introduction

A circular permutation (CP) [17] is a structural difference between two protein molecules that can be conceived as the result of a linkage between the C and N termini, followed by a linearision of the temporary, circular polypeptide by a cleavage at the surface [11] (see figure 1). A CP can also be defined on the sequence level, where it may be defined as the effect obtained by dividing a string in two, followed by concatenation of the parts in opposite order than the original. In both cases, the result of the CP is a circularly permuted version of the protein sequence/structure.

A. Natural CPs

Many circularly permuted proteins are known and have been found to occur naturally in many protein families [17].

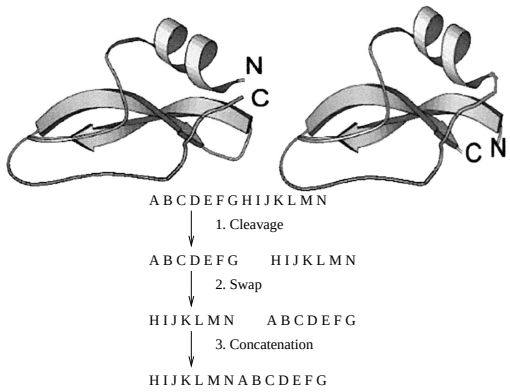


Fig. 1. The effect of a circular permutation is exemplified through the two protein structures (left/upper): The short gap between the N and C termini have been moved along the backbone. Moreover, a more formal and constructive description of the creation of circularly permuted sequence is given (right/bottom): It is similar to a swap of two substrings. Note, as desribed in the text, that CPs do not arise in this manner during evolution; rather they are probably the result of duplication, fusion, and trimming of the termini.

Among these are the saposins [27], TIM-barrel enzymes like aldolases [14], FMN-binding proteins [24], β-glucanases [10], SLH domains [20], cytosine-C^5 methyltransferases [40]. All of the above are examples of CPs occurring on the gene level. However, before these discoveries, starting with the discovery of the "Swaposin" domain in 1995 [27], only CPs arising as a result of post-translational modifications had been observed. An example of this is *concanavalin A* [6], [29], where the observable CP in the folded protein is not a result of changes after translation (and during folding) of the polypeptide chain. To summarise, many naturally occurring CPs are known, and it is generally accepted that CPs are part of the evolution of proteins, although they are not as frequent as other changes [13], [9], [17], [15], [11] such as domain duplication, swapping [12], and insertion [31].

B. Artificial CPs

In addition to CPs observed in natural processes, artificial CPs have been created in several studies. One of the first examples was reported by Goldenberg and Creighton [8], who made a CP version of *Bovine Pancreatic Trypsin Inhibitor* by linking the termini together in a peptide bond and cleaving

the temporarily circular molecule between the residues Lys15 and Ala16; the new molecule could still refold to the native conformation; Reeke *et al.* [19] created a circularly permuted version of the gene *cPRAI* in *E. coli*, and (also) found that it could actually fold. The protein structure appeared to be similar to the original as judged from its expression levels, activity, and several other criteria. Similarly, Hahn *et al.* [10] rearranged the gene encoding *H(A16-M)* by a CP, and found, by directly determining the structures, that the original and the permutant were almost identical.

Although a CP might seem as a radical change to a protein, in many cases the protein surprisingly continues to fold into a similar architecture [7] as before. This is because the majority of the intramolecular interactions stabilising the native structure are unaffected by the modification and still can exist in the CP version [11]. For instance, in a recent study of *myoglobin*, it was found that introducing a CP into the protein did not obstruct native folding; however, it decreased the stability of the structure [30]. This seems to be a general trend: The CP version folds into approximately the same architecture and often retains its function, but it is less stable and more vulnerable to sub-optimal environmental conditions. The change in stability – and in general the effect of a CP – seems to be comparable for natural and artificial CPs [11], [30].

C. Mechanism Behind CPs

The number of naturally occurring CPs found is rather limited, and indeed CPs are infrequent in evolution. Still, CPs do take place, and some instances probably remain to be discovered. The mechanism behind CPs is not completely determined. An evolutionary event causing a swap of two parts of a gene in one unit operation – while still preserving a functional gene – seems highly improbable [17], [38]. One popular hypothesis that has been suggested by several authors [27], [13] is the one that CPs might arise by gene duplication of the precursor protein leading to an in frame fusion of the genes to form a tandem repeat gene variant. The duplicated gene could exist in folded state as a dimer consisting of the two old folds. Next, an excision of a piece of the new gene would have to take place (e.g. by mutations causing stop codons to arise). Provided that the size of this excised part is roughly equal to the size of the original gene, this final product is likely to fold into the original architecture. The CP variant might be longer or shorter than the original due to “imprecision” of the cutoffs; hence the termini of the circularly permuted fold might differ significantly from its original counterpart. An alternative hypothesis for the formation of CPs is that the proteins related by CP were formed by fusion of two smaller units into one larger unit [38]. This process would then have to happen independently twice in evolution with the same subunits – and in different sequential order.

D. Motivation for Investigating CPs

To sum up, CPs are interesting for at least two reasons: Firstly, they play an active part in evolution of genes and proteins. It has been suggested that CP may serve as a constructive mechanism in evolution because the function is usually maintained in proteins subjected to CP [38]. Moreover, the CP mechanism provides a natural form of flexibility as witnessed by the fact that many inserted domains are CPs of known homologous domains [9], which might in some cases ease the insertion of the domain. This drives an area of research to uncover the extent and function of CPs in evolution. One way to conduct this research is to search for new CP relations between protein sequences or structures to determine the extent of the CPs.

Secondly, CPs can be engineered in the laboratory while the proteins continue to remain functional (although often less stable, as mentioned). This makes CPs interesting from a more practical and chemical point of view: New compounds with altered but comparable properties might be created. The ability to create such compounds might be beneficial in for instance drug design and in crystallisation studies. Artificial CP creation of a protein may ease the crystallisation and ultimately the structure determination of a new protein [33]. As the CP variant often retains the same overall fold and have unaltered binding pockets, this can be a very useful method in determining the actual structure of drug-binding sites, when only the CP variant can be crystallised.

Finally, a central question in protein folding is whether the sequential order of residues is critical to the stability and folding of protein structures [19]. CPs are ideal for studying this issue [10] and the closely related vectorial folding hypothesis [5] (stating that the N-terminus folds into a fixed core structure, and thus directs folding of the remainder of the polypeptide chain). The hope to answer to the above question is a further motivation for studying CPs.

E. Contribution of This Study

This study investigates 11 novel pairs of structurally related proteins that are believed to be evolutionary related. They were identified in SCOP using the program FASE [39]. In this paper we closely examine each of the pairs using a methodology based on previous research on detecting homologous proteins in general and CPs in particular (see section II) . In this way, it is possible to argue for the existence of a CP relation between many of the pairs. The weight of of these arguments varies from case to case, but are very convincing in at least a handful of instances. The same is the case for the quantitative, measurable values of the novel CPs compared to the SCOP annotated CPs. In all cases, the novel CPs have values that — compared to the known SCOP annotated CPs — indicate CP relation to at similar or higher degree.

II. Detecting Circular Permutations

When detecting a CP, the most direct and obvious method is to look at the amino acid sequences, here denoted by A and B. Let A' denote the permuted version of A, according to the hypothesised circular permutation. Now A' and B can be compared using standard sequence comparison methods, and if the similarity score is significant, the pair can rightfully be claimed to be homologous. In an approach similar to the above described, Ponting and Russell [27] analysed sequences

from the *Saposin* superfamily, and the result was the discovery of a fourth family within it, namely the *Swaposin* family. Similarly, Garcia-Vallve *et al.* found evidence for existence of circular permutations within the *$(\beta/\alpha)_8$-barrel*-fold using sequence data from 18 *β-glucosidase* sequences (plus the patterns of predicted secondary structures). Finally also in a study by MacGregor *et al.* [22], sequence alignments of glucosyltransferases and the α-amylase family was sufficient to conclude that the α/β-barrel of the transferases was circularly permuted in relation to the one in the α-amylases. In neither of the three cases were structural data known and thus not used.

If the 3D structures are known, the CP can be investigated using structural comparison methods. Regardless of the comparison method (sequence or structure comparison), one will eventually obtain a score of similarity, which may be analysed to assess the significance of the match. In principle, structural alignment works just like sequence alignment or any other method to assess similarity. However, because structure is more conserved than sequence during evolution (see e.g. [9]), it is a more powerful and reliable method for detecting homology than sequence based methods.

There have been various examples of algorithms for finding CPs on the sequence level. A common method is to duplicate for instance the second argument and to search for similarities between this doublet and the original first argument [38].

There has, however, only been one study performing a large scale search for circularly permuted protein pairs using structural comparison, namely a study by Jung and Lee. In that study [15], the method used was restricted to finding sequential alignments, and therefore the structures had to be permuted (or renumbered) in order to find the circular permutation. Such a modification causes the new termini to be located at the cleavage point, and one new backbone binding (of possibly non-standard binding distance) between the old termini to be introduced. More precisely, the idea was the following: Structure B was permuted on the middle, and based on the alignment of A and the modified version of B, B', a second and final permutation between A and a further modification of B', B'', was made. Thus, the above described approach searches for the appropriate CP in a series of comparisons using a linearly restricted structural alignment algorithm. In contrast, the present study uses a fully automated and general method, FASE, for large scale search. Moreover, the found CPs are analysed and assessed in great detail, which also contrasts with the approach taken by Jung and Lee, who do not analyse single comparisons.

A. Distinguishing From the Identity Alignment

There is one important issue to keep in mind when searching for CPs: As many structures have symmetric structural elements (e.g. β-trefoils, Jelly-rolls, Immunoglobulins, TIM-barrels, and Ferredoxins (see [35])) and are repetitive on the sequence level, one must carefully consider if a CP match is merely a result of the symmetry or not. More specifically, often one is faced with two possible superpositions of proteins A and B (when allowing non-linearity). One corresponds to a relative CP of the proteins, and one corresponds to the identity alignment, thus implying simple similarity of the proteins. It is a logical conclusion and a frequently used approach to argue that two structures are most likely related by a circular permutation, if the CP alignment is more significant than the identity alignment [23], [32]. When using sequence or linearly restricted structure comparison methods, they question is if the similarities between A and B' are more significant than the similarities between A and B (as above, B' denotes a circularly permuted variant of B). When allowing non-linearity one must simply compare the significance of the two alternative similarities.

Sergeev [34] used such arguments on *$(\beta/\alpha)_8$-barrel* structures; he found that a special scoring function for structural and sequential similarity was optimised precisely when two structures was compared with one of them being permuted.

Aloy *et al.* [1] compared structural similarities between *transit peptide cleavage system* (scop d1hpca_) and *glucose permase* (scop d1gpr_ _) with the similarities found when *transit peptide cleavage system* was circularly permuted. They found (backed up by other evidence) that it was most likely that the pair of structures was related by a circular permutation, because 10 more residues could be superposed when *transit peptide cleavage system* was permuted; moreover, the $RMSD$ when aligning with respect to the CP was 0.7 lower than using the identity alignment. Finally the sequence similarity was 33% for the CP instead of 4% for the id.

Jung and Lee [15] compared the similarities between A and B' to those found between A and B by defining a threshold (5 standard deviations away from the mean similarity score). If the A-B' match was judged significant by this threshold and A-B was not, the pair was defined as being related by a circular permutation, but if A-B was significant as well, the CP similarities were ascribed to symmetries in the structures. Of course, any approach using thresholds in this fashion, might prove very sensitive and instable when both scores lie close, and on each side of, the threshold, This is because a pair, A and B, might be considered to be best related by the identity alignment (or alternatively by a CP alignment) based on two similarity values that might be arbitrary close.

B. Detection Based on Non Structural Similarities

Structural similarities, however, alone are in general not sufficient to conclude that two protein structures are descendants of a common ancestor (are homologous) [21]. There are a number of examples of convergent evolution, where two structures have the same fold but are not evolutionary related [9]. However, given structural similarity, one may make an informed decision [21] regarding the likelihood of a seemingly homologous pair by analysing similarities for different properties of the structures such as for instance molecular function, unusual features, binding sites, and sequence similarity after structural alignment [21]. Function can be inferred from key active site residues, or protein binding sites or surfaces [1], and thus be used to detect or reject homology.

In particular, the sequence similarity between a pair of structures after a structural alignment has been shown to

be one of the best discriminators of homology and even functional similarity [1]. Murzin [23] derived a measure for the significance of a sequence similarity present in a structural alignment, and specifically concluded that a pair of structures were significantly homologue based on a sequence similarity of 23% after structural alignment. Russell [32] analysed 335 pairs of structurally aligned proteins from SCOP. They were divided into analogues, and remote, medium and close homologies, and it was found that there was a higher degree of sequence similarity in remote homologies than in analogues in structural alignments. Thus, this measure, sequence similarity after structural alignment, could help in distinguishing between homology and analogy. In [28] it was stated (by referring to [32]) that *sequence identities in excess of 12% were more likely to be associated with superfamily rather than mere fold similarities*. Also in [28], sequence similarity (and derived Murzins P-value) for structural alignments was used to argue for homology between domains of the β-trefoil structure.

To illustrate the difficulty in detecting similarities for remotely related proteins, we may take a look at *human Glutathione synthetase (GS)*, which belongs to the recently identified *ATP-grasp* superfamily, despite the fact that it displays no detectable sequence identity with other family members. *hGS* has a rare gene permutation which has resulted in a circular shift of the conserved secondary structure elements in it with respect to the other known *ATP-grasp* proteins. Nevertheless, it appears likely that the enzyme shares the same general catalytic mechanism as other ligases [26]. The pairwise sequence identity between *hGS* and *ecGS* after structure-based sequence alignment is only 10% [26].

We should mention finally that the detection of CPs, and in general making hypotheses about homology, is a non-exact endeavour, where nothing can be proved with certainty, and where the hypotheses indeed will have to remain (possibly very likely!) hypotheses. In this paper, we study remote CP similarities whose true origin may therefore debated to a high degree.

We know that a common ancestor does not imply same structural fold and the same structural fold does not imply common ancestor, which makes fold definition an empirical and approximative art with unavoidable contradictions in classifications [9]. We do not wish to contribute further to the fruitless debate of homology or analogy (divergent or convergent evolution) that ultimately is unverifiable [37], [2], but simply to use previously introduced techniques to point out interesting and novel relationships that could cause some reflection about the existing methods to define CPs and classifications (and classification methods) in general.

III. Methods

For the detection of the proposed CPs that we shall review below, we used the program FASE [39] (Flexible structural Alignment from Secondary structure Elements). FASE is an "all in one" embedded tool for structural alignment and superposition of proteins. It can find sequential and non-sequential alignments, it aligns sequences based structural superposition, and calculates sequence similarity after superposition based on a flexible algorithm and robust scoring scheme. To closer examine the nature and validity of the found CPs, we shall review each of them below in detail using arguments and methodology derived from previous research on CPs. However, before presenting the results, we will first define the scores for alignment linearity and circular permutation and then describe the methods and the approach taken for finding the novel CP pairs.

A. A Score for Alignment Linearity

The score for alignment linearity is defined in the following way. Let the residues of structure A be $r_1^A, \ldots, r_m^A$ and the residues of structure B be $r_1^B, \ldots, r_n^B$. An alignment of A and B is a list of pairs of residues. For instance an alignment of size 3 could be: $(r_1^A, r_3^B), (r_4^A, r_4^B), (r_7^A, r_6^B)$.

The linearity score was defined as the summed difference of the indices of each aligned pair from the identity alignment. For example in the alignment, $(r_1^A, r_1^B), (r_2^A, r_2^B), \ldots, (r_m^A, r_n^B)$, the sum ($v$) is: $0 + 0 + \ldots + (m - n)$. This difference was then normalised with the theoretical maximum divergence, $0.75 r_A^m r_B^n$, from the identity alignment. The value v was then modified to $1 - v$, causing the identity alignment to get 1 and the most deviant alignment to get 0 in linearity scores.

B. A Score for Alignment Circular Permutation

The systematic search for CPs in a database of course requires a precise and stringent definition of what one is looking for [38]. Therefore, below a precise description of the CP score will be given. In defining a score for CP, there are two aspects that must be addressed, namely detection and assessment. Firstly, the CP must be detected (is there a CP at all, and where is the "break"). When this is resolved, one may assess the degree of the found CP.

If an alignment corresponds to a CP, it can be detected by scanning through the list of aligned residues and for each position, i, calculate the average of the structure B components to the left of i,

$$avg_L(i) = \frac{1}{i} \sum_{j=0}^{i-1} r_{alB(j)}^B,$$

and to the right of i,

$$avg_R(i) = \frac{1}{N-i} \sum_{j=i}^{N} r_{alB(j)}^B,$$

where $alB(j)$ refers to the residue number of the structure B component of the j'th aligned pair and N is the alignment size. If there is a CP the there will be an i for which $avg_L - avg_R$ is positive, and the index where $avg_L - avg_R$ is maximal (i_{max}) will match the beginning of structure B.

In the assessment of the CP we take into account the size of the two parts before and after the break. If, for example, structure A starts with matching the last five residues from structure B, whereafter A starts to match the beginning of structure B, then the CP cannot be claimed to be very significant.

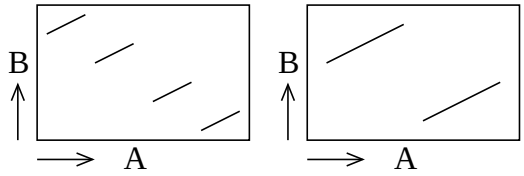


Fig. 2. Two schematic alignments are shown. The right alignment might be a real CP, whereas the left is definitely not, but still gets an equal CP score. However, using the measure for straightness of the left and right halves, it is possible to make distinction.

Thus if we define

$$rearrscore_A(i) = \frac{avg_L(i) - avg_R(i)}{N_B} \cdot \frac{i}{N_A} \cdot \frac{N-i}{N_A}$$

where N_A and N_B are the sizes of structures A and B, we can define that $rearrscore_A = max_i(rearrscore_A(i))$. The final score for the degree of CP takes the degree of the permutation into account as seen from both structures, and we define it as:

$$rearrscore = 4 \cdot (rearrscore_A + rearrscore_B)$$

The $rearrscore$ has a theoretical maximum value of 1 (a perfect CP), and a minimum value of -1 (corresponding to a fully sequential alignment). Values less than 0 are truncated to 0. This measure takes three factors info account: 1) The size of the alignment, 2) the partition of the permutation, and 3) the "pureness" of the permutation. The product $\frac{i}{N_A} \cdot \frac{N-i}{N_A}$ account for 1) and 2), whereas $\frac{avg_L(i)-avg_R(i)}{N_B}$ account for 3).

By applying the "straightness" measure locally to the two parts of the alignment (as defined by the circularity), it is possible to distinguish a real CP alignment (figure 2, right) from one that just gets the same CP score but is not a CP (figure 2, left).

C. Finding the Novel Circularly Permuted Pairs

The SCOP40 database (also known as the ASTRAL database) [3] is a subset of the SCOP classification, in which the pairwise sequence similarity between any pair of sequences is below 40%. This ensures that very similar sequences are removed, and the result is a non-redundant and diverse selection of protein domains. In the present work we have used SCOP40 version 1.65, which contains 9455 domains. We made a further subset selection from SCOP40 by picking domains having between 100 and 250 residues. We will refer to this subset, containing 2932 domains, as the SCOP40s database.

Subsequently, we performed an all against all comparison of the SCOP40s database. From the 8.6 million comparisons, we selected 1930 of the most promising candidate CP pairs. The selection was based on four criteria.

The first criterion was the degree of which the best circularly permuted similarity was better than the best identity solution (measured in number of aligned residues). A cutoff value of 5 was used, which means that the best CP alignment has at least 5 more residues in its alignment that the best linear alignment for the particular pair of structures.

The second criterion was the TOP score function no. 1 (see FASE paper). For this measure the cutoff was 6, implying that only significantly similar structures were included. For structures in different folds, only 1.5% get a similarity score less than 6 (see FASE paper).

The third criterion was alignment size. A minimum size of 60 was enforced to exclude possibly insignificant matches of little structures.

The fourth and final criterion, and the most important one, was the CP score. This criterion ensured that the alignment of the chosen candidate pairs resembled a CP to a significant degree. The cutoff value used was 0.3. Only 0.1% of domains from different folds get a CP score higher than 0.3.

Since the objective was to find novel CPs, it was necessary to exclude all the pairs, where one or both of the structures was located in a SCOP sub-tree that was known to be CP related to another SCOP sub-tree. This completely excluded re-discovery of known CPs from SCOP. This requirement excluded 109 pairs

In addition we only selected pairs from different folds. By adding this restriction only 931 comparisons were left. The final 931 possible inter-fold CP relationship came from a total of 104 folds (SCOP has 968 folds in total).

The presented CPs in this paper were selected in a last step from the 968 pairs by a manual inspection and selection. The most significant pairs according to the TOP and CP scores were investigated first, and the selection was terminated when a sufficient number of examples had been obtained.

Below it will be described how the best identity and CP alignments were detected and selected from the ranked list returned by FASE of the many potential ways in which two structures can be related. They were – as described above – used in the enforcement of criterion number one. The best CP alignment describes the actual similarity between any potential novel CP pair, and is for that reason crucial to the whole method.

D. Finding the Best Identity Alignment

The best identity alignment is found among the list of ranked solutions by the following procedure: The list is checked from the top. If a solution has a straightness value above 0.75 and if the product of its straightness value and its alignment size exceeds that of the previous, then it is selected as the current candidate for the best identity solution.

This ensures selection of a solution that resembles the identity alignment to a sufficient degree (above 0.75), and it moreover finds large alignments (crucial in the first criterion of the CP selection, where any potential CP must have an alignment with at least 5 more residues than the identity alignment) with a small bias in favour of really straight alignments (the multiplication of the straightness factor onto the size).

E. Finding the Best CP Alignment

The best CP alignment is found using the following procedure: Again the list is inspected in a top-down manner. A candidate must have a CP score above 0.1 and both the

straightness values of the left and the right half of the alignment must be above 0.4. Moreover, to replace the previous candidate, a solution must have a better CP score than the old one and either the left or the right part of CP must have a better straightness value than the previous selected solution.

The scheme ensures a certain lower bound in the quality of the chosen CPs (CP score above 0.1). The straightness requirements on the two halves weeds out some improper CPs that are falsely recognised as true CPs by the CP score measure. The requirement for replacement of the previous CP ensures that we get the best CP that has at least one very straight half.

F. Definition of RMSD

The $RMSD$ measure between two protein structures expresses the quality of the match of the C_α atoms from the two backbones under a given superposition.

Thus, if $(a_1, a_2, \ldots, a_N)$ and $(b_1, b_2, \ldots, b_N)$ are the matching C_α atoms from protein A and B respectively, with the coordinates updated with respect to the chosen transformation/superposition T, and with a_i matching b_i for $1 \leq i \leq N$, then the $RMSD$ is defined as

$$RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N} |a_i - b_i|^2}$$

where $|x - y|$ is Euclidean 3D distance between x and y.

IV. Results – Novel Possible Circular Permutations

In this first results section (of the two results sections in the paper), pairs of protein structures proposed to be related by a CP are described and analysed. These novel pairs were found using the approach described in the methods section. The purpose of the present results section is to investigate the plausibility of the found pairs — the investigation is necessary because the numerical measures for similarity and CP value are insufficient in fully describing the range of relationships that can exist between pairs of related protein structures.

A. α-toxin and Furin

The first pair to be reviewed is α-*toxin* from the *Lipase*-fold and *Furin* from the *Galactose*-fold. The α-*toxin* domain has the SCOP sccs identifier `d1ca1_2` and belongs to the α-*toxin* family (3), from the *Lipase* superfamily (1), from the *Lipase* fold (12), from the *All beta*-class (b). In SCOP, this hierarchical placement is written as b.12.1.3. In the following we shall therefore refer to sccs name and SCOP placement using a more compact notation, which in the present case yields (scop `d1ca1_2` b.12.1.3). Similarly, a compact description of *Furin* is (scop `d1p8ja1` b.18.1.20).

Both domains have an all-β sandwich architecture with 8 strands and 2 sheets. However, their topologies are slightly different (figure 3), and they are therefore classified into different folds. Using FASE to detect similarities across different topologies it is nevertheless evident that the similarities between α-*toxin* and *Furin* are as pronounced as those found between α-*toxin* and some members of its superfamily, e.g. (scop `d1f8na2` b.12.1.1). In particular, α-*toxin* and *Furin* can be superposed over 94 residues with a deviation of 2.14 *RMSD*. This gives a structural similarity score of 3.42 for the TOP score function no. 1 [18] (found to be a good fold discriminator and measure for evolutionary relatedness in previous work [39]). In the following we shall refer to this score as the "S-score". The lower the S-score, the more similar are the two compared structures. The sequence similarity (measured only for the equivalenced residues in the superposition) is 7%. We will in the following use a compact notation for these similarities; in the present case it yields (*N/RMSD* 94/2.14 *S* 3.42 *ID* 7). For comparison, the similarity between α-*toxin* and `d1f8na2` from the superfamily is only (*N/RMSD* 80/1.81 *S* 3.52 *ID* 6). Thus, despite α-*toxin* and *Furin* belonging to different folds, their similarity is higher than similarities between α-*toxin* and some of its superfamily members with which it per definition has a *probable common evolutionary origin*. In this sense, it makes sense to argue that the pair is closely enough related to be homologs.

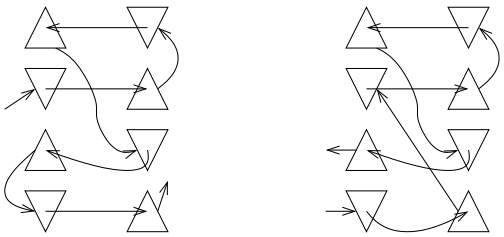


Fig. 3. TOPS cartoons of α-*toxin* left (scop `d1ca1_2` b.12.1.3) and *Furin* right (scop `d1p8ja1` b.18.1.20).

The found similarity conforms to the characteristic CP: The C-terminal (20 residues; β-strands 7 and 8 in figure 3 left) of α-*toxin* matches the N-terminal of *Furin* (β-strands 1 and 2 in figure 3 right). More detailed, the ends of α-*toxin* are connected by 9 *Furin* residues similar to a 18 Å distance, which might have arisen as a cleavage between strand 2 and 3 in *Furin*. Similarly, the ends of *Furin* could have arisen as a cleavage between strand 6 and 7 in α-*toxin*.

Except for some minor differences between the superposed structures on the loops between strands 1 and 2 and between strands 2 and 3 in α-*toxin*, the structural similarity is high (figure 4, left). The CP is not an alternate alignment to the identity alignment, but rather it *is* the identity alignment with 2 extra matched strands in the beginning and end of the structures respectively.

Initially the structural similarity seems to indicate homology by CP between α-*toxin* and *Furin*. When looking closely at the β-strand superpositions it however becomes clear that some strand residues do not superpose particularly well (figure 4, right). This combined with the fact that the fold in question is very common, and the insignificant sequence similarity after structural superposition opens the possibility that the similarity only reflects a structural analogy – the domains

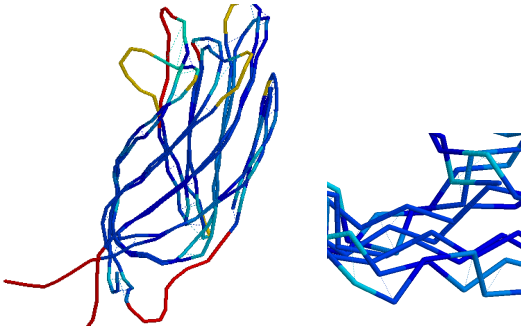


Fig. 4. Superposition of α-*toxin* and *Furin* (left) and zoom of superposition (right). Dark blue indicates good structural fit. Yellow (structure 1) and red (structure 2) indicates unaligned parts. The superposition on the left has been smoothed for clarity.

are unrelated. Therefore, the structural evidence alone is not enough to determine homology by CP between α-*toxin* and *Furin* although the pair shared architecture.

B. Immunoglobulin and C2

As in section IV-A above, the two present structures are of the β-sandwich type with 8 strands in 2 sheets, namely an *Immunoglobulin* domain (scop `d1cid_1` b.1.1.1) and a *C2* domain (scop `d1dcea2` b.7.4.1). In this case, the CP is more pronounced (40 residues and 3 strands), the structural similarities are weaker in the less important loop regions but stronger on the strands, and the sequence similarity is higher (*N/RMSD* 74/2.67 *S* 4.64 *ID* 12). The best identity alignment, to which the proposed CP must be compared, is much weaker, namely (*N/RMSD* 59/2.34 *S* 5.72 *ID* 15). The CP is stronger not only for the S-score, but for all the measures used in the consensus score introduced with FASE [39]. The higher sequence identity (*ID*) for the identity alignment arises only because this alignment is shorter; the absolute number of matched residues is the same for the CP.

It is well known that there is an internal CP in the *C2* fold [25], although this is not noted in SCOP. Moreover, according to SCOP the fold of the *TRAF* superfamily (b.8.1) is related to the *Immunoglobulin* fold (b.1) by a CP. These two CP relations do, however, not link *Immunoglobulin* and *C2* presented in this section. A possible explanation for the presented CP could be the result of two divergences: 1) Divergence of the *C2* fold from the *Immunoglobulin* fold, and 2) internal divergence of the *C2* superfamily. In CATH the two structures are placed in the same topology (*Immunoglobulin*-like), but in different homology classes (*immunoglobulins* and *transferase*). This reflects the fact that there is definitely some structural evidence for the proposed homology by CP, but possibly not enough to claim a common evolutionary origin.

C. TIM β/α-barrels: Ribulose and Aldolase

Continuing with an example of a CP within a fold, we present a *Ribulose* superfamily domain (scop `d1dbta_` c.1.2.3) and an *Aldolase* superfamily domain (scop `d1euaa_` c.1.10.1.5). In SCOP, both structures are in a sub-tree that is listed as being related to some other sub-tree of the classification by a CP: The fold of the *PLP* superfamily (c.1.6) is related to the *TIM* fold (c.1) by a CP, and the *Aldolase* family (c.1.10.1) contains an internal CP as one of its domain groups, *Transaldolase* (c.1.10.1.7), is related to the *Aldolase* family by a CP. However, acoording to SCOP *Ribulose* and *Aldolase* have no CP relation. The structures have the same architecture and even the same topology (figure 5), namely the *TIM β/α-barrels* architecture/topology.

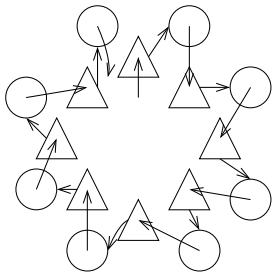


Fig. 5. TOPS cartoon of the *TIM β/α* fold that *Ribulose* (scop `d1dbta_` c.1.2.3) and *Aldolase* (scop `d1euaa_` c.1.10.1.5) share.

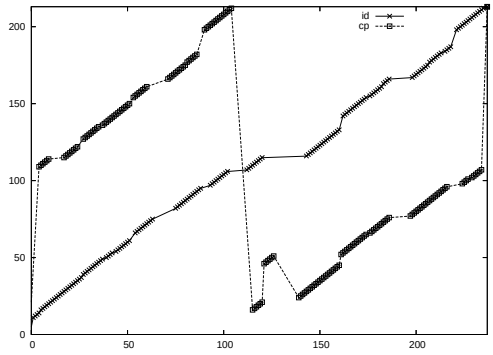


Fig. 6. Dotplot illustrating the identity alignment and the CP alignment of *Ribulose* (scop `d1dbta_` c.1.2.3) and *Aldolase* (scop `d1euaa_` c.1.10.1.5).

The *TIM β/α-barrels* have an eight-fold symmetry that can be found by rotating the structure around the axis parallel to the secondary structures. When such symmetries are present, it requires (as mentioned in section II-A) extra attention to distinguish found CPs from symmetrical artefacts. In this context, one can compare the similarities found by simple identity alignment with those found by CP alignment; this will allow for identification of the most plausible evolutionary scenario of the divergence. If the CP alignment possess many characteristics that are typical for related protein structures (e.g. high sequence similarity and a good fit of the core residues), it is more likely that the CP reflects a common evolutionary origin rather than just being a chance similarity emerging due to structural symmetries. The two different alignments of *Ribulose* and *Aldolase* are illustrated in figure 6.

Structurally, the CP solution (*N/RMSD* 162/2.56 *S* 4.20 *ID* 14) is slightly better than the identity solution (*N/RMSD* 156/2.57 *S* 4.45 *ID* 12), although it can be argued that the differences are only marginal. However, of the 6 best solutions found by FASE, the three CP solutions have sequence identities 14%, 14%, and 15%, while the three identity solutions only have sequence identities of 12%, 9%, and 12%. This speaks in favor of the CP solution type as the most probable homology relation. Also in support of this hypothesis is Ponting and Russell[28], who found that a sequence identity above 12% after structural superposition was more likely to indicate superfamily relationship instead of just a fold relationship. Furthermore, it is known that TIM barrels do contain CPs internally in the fold. Based on the above analysis it is indeed a realistic possibility that *Ribulose* and *Aldolase* are evolutionarily related by a CP.

In addition, another pair of domains was found that might indicate a stronger relatedness by CP than the former. The first structure (scop d1km3a_ c.1.2.3) is from the same family and domain as above but from a different species. Thus, d1dbta_ and d1km3a_ are very similar (*N/RMSD* 194/1.66 *S* 2.07 *ID* 23). The second structure (scop d1nsj__ c.1.2.4) is also located in the *Ribulose* superfamily as above, but belongs to a different family than d1km3a_. Thus, these structures are more closely related in SCOP (same superfamily) than those above (only same fold).

The CP and identity solutions are found with ranks 0 and 1, respectively. They seem equally good from a manual inspection as well as judging from the structural similarity scores: CP (*N/RMSD* 166/2.57 *S* 3.62) and identity (*N/RMSD* 157/2.34 *S* 3.59). However, when considering the sequence similarities for the aligned residues for the CP and the identity solutions respectively, the difference was striking: The best CP solutions class had sequence similarities of 18%, 18%, and 17%, while the best identity solutions had only 12%, 11%, and 10% sequence similarity. In other words, the sequence similarities for the CP solutions were 50% higher than for the identity solutions. This significant difference indicates that the CP is the most plausible type of homology.

The question of homology between *TIM* β/α*-barrels* has been dealt with previously. Jia *et al.* found a CP to be the most probable relation between *transaldolase* and *class I aldolase* [14]. This relationship has been annotated in SCOP, and was the one between c.1.10.1 and c.1.10.1.7 mentioned above. Interestingly, Jia *et al.* substantiate their claim be observing better structural fit for the CP alignment according to *N* and *RMSD* than for the 7 other possible alignments. Similar argumentation was used above in the present work when arguing for the CP: The CP alignment had better structural fit – although not significantly better; however, this fit was coupled with a higher sequence similarity over the alignment, which is a property that is though to be significant when deciding homology [23], [32], [28], [1].

As a final remark, in a study by Copley [4] it was found that the superfamilies of the *TIM* β/α*-barrel* fold were more related than previously thought according to the general opinion in the field: Copley found that 12 of 23 SCOP *TIM* β/α*-barrel* superfamilies shared a common origin.

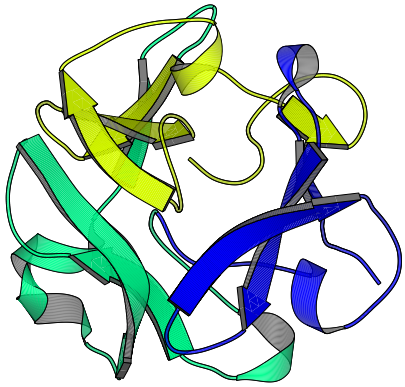


Fig. 7. The *beta-trefoil* structure of *Mannose receptor* (scop d1dqga_). The three repeated structural units are coloured blue, green, and yellow according to their sequential order in the chain.

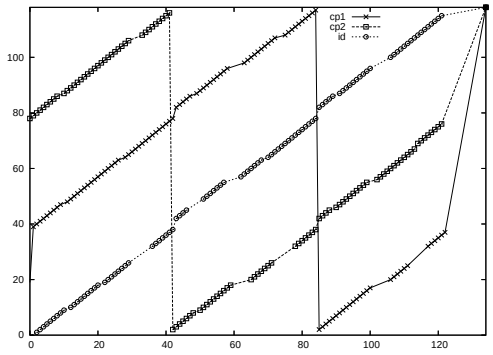


Fig. 8. The identity alignment (id), circular permutation alignment number 1 (CP1), and circular permutation alignment number 2 (CP2) of the two *beta-trefoil* structures *Mannose receptor* (scop d1dqga_) and *Hisactophilin* (scop d1hcd_).

D. Beta-trefoils

This example with two *beta-trefoil* structures, (scop d1dqga_ b.42.2.2) from the *Ricin* superfamily (figure 7) and (scop d1hcd_ b.42.5.2) from the *Actin* superfamily, is in many respects similar to the previous one with the *TIM* β/α structures: The *beta-trefoil* structures consist of three repeated units (the *TIM* β/α structures have eight) and when comparing two such structures, they may therefore be aligned in several (here three) different ways according to their internal symmetries (figure 8).

Before dealing with CPs, we may consider if all *beta-trefoil* domains can be hypothesised to homologous. Contrary to the implicit statement made by SCOP (that *not* all *beta-trefoils*

	ranks	seq. ids. (%)
id	4 7 9	8 7 6
CP1	0 3	7 7
CP2	1 2 5 6	14 15 10 10

Fig. 9. Sequence identities after structural alignment for the best alignments of d1dqga_) and d1hcd__). The similarities have been divided into three groups according to type of alignment: identity, CP1, and CP2.

necessarily are homologs, as they are located in different superfamilies), according to Ponting [28], the known families of β-trefoil structures are likely to form a single evolutionarily related superfamily of proteins. Of course this claim is debatable, but assuming that it holds, we deal below with the next question that arises, namely that regarding the *type* of relationship that exists between some of these *beta-trefoil* homologs. Are all relations between *beta-trefoils* structures given by the identity alignment (aligning unit 1 with unit 1, unit 2 with 2, and unit 3 with 3), or might some *beta-trefoils* pairs be related by a circular permutation?

As reviewed in the introduction, one has to compare the CP and the identity alignments and select the most plausible evolutionary scenario. The identity and CP alignments number one and two (CP1 and CP2) are displayed in figure 8). Looking at this particular pair only, the structural arguments are inconclusive although the two CPs have a marginally better fit than the id. The values are CP1 (*N/RMSD* 103/2.57 *S* 3.47), CP2 (*N/RMSD* 103/2.58 *S* 3.49), and identity (*N/RMSD* 97/2.73 *S* 4.05). However, looking at the sequence similarities, CP2 gets significantly higher values than id and CP1 (figure 9). The difference is even more pronounced than for the *TIM β/α-barrels* in section IV-C. In short, the sequence similarity for CP2 is twice as high as for the other alignments, which seems to indicate that if there is a homology between (scop d1dqga_ b.42.2.2) and (scop d1hcd__ b.42.5.2), then this homology is given by CP2.

In previous work [16], Koike *et al.* found evidence for CPs between *beta-trefoil* domains from different families. Specifically, it was found that (scop d1isza1 b.42.2.1) was most likely circularly related to (scop d1fwua_ b.42.2.2) – that is, a CP within the *Ricin* superfamily. The CP described in this section suggests CP between superfamilies *Ricin* b.42.2 and *Actin* b.42.5. However, further experiments revealed that this relationship was not consistent for all superfamily members (data not shown). In general, a picture of a strongly connected *beta-trefoil* fold emerges (with respect to structural similarity), where domains are related by CPs across superfamilies and families, and with little correlation between family membership and relation by CP or not. It seems likely that the beta trefoil fold has evolved through a series of duplications [28], and that the true evolutionary relationships between its domains therefore are too complex and distant for us to fully uncover.

E. Ferredoxin and CheY

The domain (scop d1fdr_2 c.25.1.1) from the *Ferredoxin* fold and the domain (scop d1jbea_ c.23.1.1) from the *Flavodoxin* is the next pair under investigation. Both structures

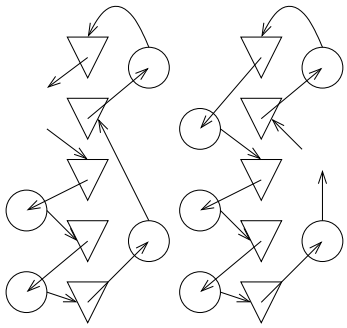


Fig. 10. TOPS cartoon of (scop d1fdr_2 c.25.1.1) on the left and (scop d1jbea_ c.23.1.1) on the right.

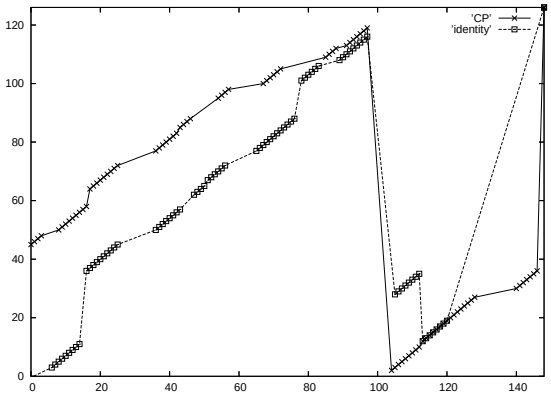


Fig. 11. Double dotplot of the CP and identity alignments of (scop d1fdr_2 c.25.1.1) and (scop d1jbea_ c.23.1.1).

have a fold with five parallel beta strands in a central sheet and with two helices on each side running in the anti-parallel direction of the sheet (see figure 10).

The domains can be related by a CP alignment and by an identity alignment. The sequence similarities for the two types of alignments are roughly equal, whereas the CP has a better structural score (*N/RMSD* 86/2.47 *S* 4.97 *ID* 16) than the identity alignment (*N/RMSD* 80/2.54 *S* 5.70 *ID* 16). (The two alignments are shown in figure 11 as dotplots.) Consequently, the CP solution is ranked 1 among the found superpositions by FASE and the identity superposition is ranked lower.

Even though the CP is structural slightly more convincing, this evidence alone does now allow us to conclude that the CP is the most likely explanation of their differences. One missing requirement is the higher sequence similarity for the CP alignment. On the other hand, looking at the so-called identity alignment, we see that it is not a perfect identity alignment. Two scenarios could explain this alignment – at least partially: One scenario is that the two last matched secondary structure elements (SSEs) in d1fdr_2 in the identity alignment are an insert, and the other is an alternative CP arising by excluding the first matched SSE of d1fdr_2 also

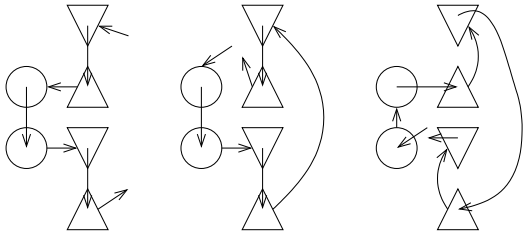


Fig. 12. TOPS cartoons of *Aldolase* (scop d1dhn_ d.96.1.3) left and *Ribosomal protein S3* (scop d1fjgc2 d.53.1.1) middle (CP/extended identity alignment) and right (weird alignment).

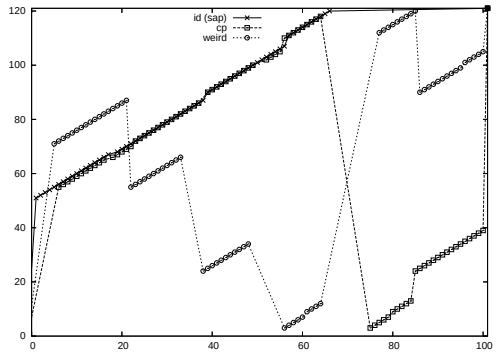


Fig. 13. The identity alignment, CP alignment, and "weird" alignment of *Aldolase* (scop d1dhn_ d.96.1.3) and *Ribosomal protein S3* (scop d1fjgc2 d.53.1.1).

in the identity alignment. Considering these scenarios, the CP alignment could well be the most likely alignment to relate the pair of domains in case they are homologous.

F. Aldolase and Ribosomal Protein S3

The following pair of structures is *Aldolase* domain (scop d1dhn_ d.96.1.3) from the *T-fold* fold and *Ribosomal protein S3* (scop d1fjgc2 d.53.1.1) from the *Ribosomal protein S3* fold. Thus, both structures belong to the α+β class, in which the α and β SSEs are segregated in groups sequentially (instead occouring alternating along the backbone). The topologies for the two structures can be seen in figure 12, where the two rightmost sub-figures illustrate the differences in orientation of the second structure relative to the fixed first structure in the two cases that we shall review just below. The domain pair is namely interesting in two respects:

The first observation is that the identity alignment can be extended (figure 13) with 30 more matching residues (if non-sequential alignments are allowed). The resulting similarities suggest that the pair could be related by a CP (*N/RMSD* 81/2.86 *S* 4.58 *ID* 4). However, the low sequence similarity suggests that the CP might not explain the divergence in the case the domains were related.

The second observation is that another superposition can be made with similar degree of structural match and a much higher sequence similarity (*N/RMSD* 73/2.26 *S* 4.24 *ID* 13). However, the superposition corresponds to a very unconventional alignment that seems unlikely in the context of protein evolution (figure 13). In fact, this alignment has a better S-score than the CP/extended identity alignment, and it has a lower *RMSD* and *N* in the trade-off between low *RMSD* and high *N*. Although this alignment requires many changes in SSE connectivity to explain a possible evolutionary divergence, the higher sequence similarity is noteworthy. The alignment is interesting because it points towards possible evolutionary mechanisms on the protein structure level. Although it may seem improbable, it is known that related domains can adopt very different folds [9], and considering the high sequence similarity it is indeed possible, despite the fact that it cannot be proved in a stringent way.

G. HPr fold and Ferredoxin fold

The two domains, (scop d1j5ya2 d.94.2.1) from the *HPr* fold and (scop d1p1la_ d.58.5.2) from the *Ferredoxin* fold, offer two seemingly equally likely ways to be related. As in section IV-F above, both structures belong to the α+β class. One alignment is the identity alignment and the other is a CP alignment. Assuming that the pair is related as indicated by each of the superpositions, a series of possible evolutionary events may explain each alignment.

The identity superposition has a structural similarity of (*N/RMSD* 63/2.79 *S* 5.80 *ID* 7). As it is clear from the identity superposition of the structures (figure 14), they superpose well (indicated with blue). This matching begins at the yellow and red N-termini that point downwards in the bottom of the figure and continues for approximately 50 residues. Hereafter, the structures begin to diverge and the *HPr* domain has an unmatched region with one strand and one helix of approximately 35 residues and the *Ferredoxin* domain has an unmatched region containing a beta-hairpin of approximately 15 residues. Then, finally, two SSEs (strand and helix) are matched, whereafter the C-terminal of the *HPr* domain is reached, while the C-terminal of *Ferredoxin* domain (approximately 22 residues) is left unmatched. This is also illustrated with TOPS cartoons in figure 15.

Assuming the identity alignment, the short loop in the *HPr* domain between SSEs *c* and *d* (a β-hairpin) must have diverged from the unnumbered strand and helix in the *Ferredoxin*. There is a length difference of 20 residues between these two substructures, and they are structurally very dissimilar. Furthermore, the end in *Ferredoxin* would have to be accommodated by *HPr*, but this would requires a significant insertion or deletion of approximately 22 residues in *Ferredoxin*.

In the CP superposition (*N/RMSD* 57/2.02 *S* 4.87 *ID* 17) (figure 16) we find again the N-terminus of the *HPr* domain coloured yellow in the bottom right corner of the figure. Following the backbone from here, it matches approximately 50 residues of the *Ferredoxin* domain (strand 1, helix 2, and strand 3 – as numbered in figure 15 right). The unmatched termini of *Ferredoxin* (red helix, upper right corner) now corresponds to an unmatched region of the *HPr* domain. Hereafter, as in the identity case, two SSEs (strand and helix)

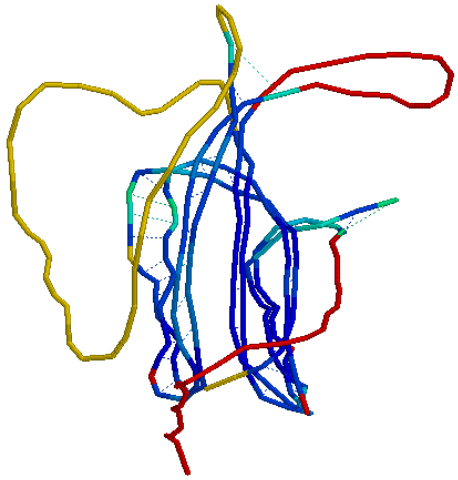


Fig. 14. Identity superposition of (scop d1j5ya2 d.94.2.1) from the *HPr* fold and (scop d1p1la_ d.58.5.2) from the *Ferredoxin* fold.

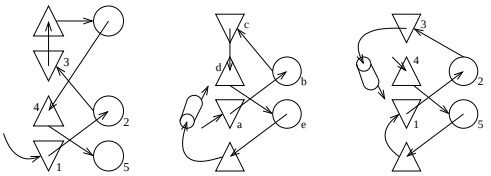


Fig. 15. TOPS cartoons of (scop d1j5ya2 d.94.2.1) from the *HPr* fold (left) and (scop d1p1la_ d.58.5.2) from the *Ferredoxin* fold (identity rotation middle and CP rotation right).

are matched. The second divergence is that the end of the *HPr* domain is reached (yellow, right), while the *Ferredoxin* domain continues and wraps back to match the N-terminus of the *HPr* domain.

It might be difficult to select either hypothesis as the most likely one. As the above structural review has shown, the quality of the matches are comparable which is also indicated by structural similarity scores. There is, however, one compelling difference between the two similarities: While the 3 best CP similarities have sequence identities of 17%, 22%, and 19%, then the 4 best identity similarities have sequence identities of only 7%, 6%, 3%, and 8%. Thus, while the structural arguments are indecisive with respect to finding the most likely evolutionary scenario, the sequence argument is clearly indicating homology by CP. The number of matching residues is three times higher than in the other alignment.

Another interesting observation is that d1j5ya2 is the only domain in the fold d.94, which is not a full chain. On the contrary, it is an insert, namely residues 68 to 174 of chain A in the protein 1j5y (pdb-id). It is known that inserts often get circularly permuted [9]; a fact that is in accordance with the hypothesis outlined above.

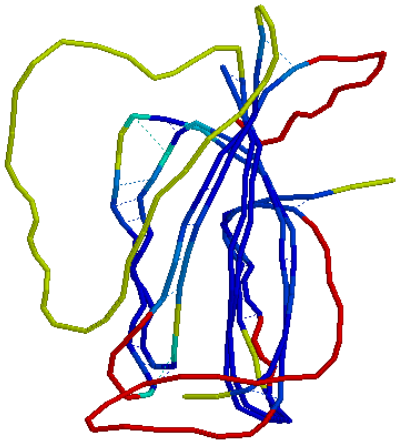


Fig. 16. Superposition CP

H. MurD and CheY

The two structures in this example are *MurD* (scop d1jbwa1 c.59.1.2) from the *MurD* fold and *CheY* (scop d1mb3a_ c.23.1.1) from the *Flavodoxin* fold.

Again, there are at least two ways of alignment, and the CP alignment (*N/RMSD* 87/2.73 *S* 4.17 *ID* 12) is slightly better than the identity alignment (*N/RMSD* 73/2.66 *S* 5.29 *ID* 8) structurally and with respect to sequence similarity. The 2 best CP similarities have sequence similarities of 12%, and 11%, while the 3 best identity similarities have sequence identities of only 8%, 8%, and 4%.

MurD (scop d1jbwa1 c.59.1.2) does also match *MoaB* (scop d1mkza_ c.57.1.1) from the *Molybdenum* fold very well. In this case, the CP is even more obvious: The identity alignment is much weaker (in fact there is not really an identity alignment of the domains) than in the previous example, and *MurD* is even better matched than before (*N/RMSD* 96/2.53 *S* 4.48 *ID* 10) (or a slightly modified CP2 (*N/RMSD* 80/2.95 *S* 6.85 *ID* 17) with higher sequence similarity; see figure 17).

I. Streptavidin and Transmembrane beta-barrel

An earlier example (section IV-D, *beta-trefoils*) showed that the sequence identity can vary a great deal for different alignments of symmetric and repetitive structures. Thus, not all of these symmetric alignments are equally likely to explain an evolutionary divergence between the two structures in question. With the β-barrel architecture, the same situation occurs: The fold has many internal symmetries, and two domains of this fold can therefore be superposed in many similar ways.

The domain (scop d1jmxa5 b.61.4.1) from the *Streptavidin* fold and (scop d1qjpa_ f.4.1.1) from the *Transmembrane beta-barrel* fold, are both structures of the β-barrel type (figure 18). Both structures have four main symmetries, and

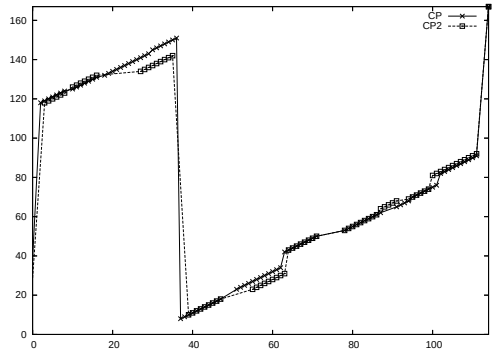


Fig. 17. Dotplots of two almost similar CP alignments of *MurD* (scop d1jbwa1 c.59.1.2) and *MoaB* (scop d1mkza_ c.57.1.1).

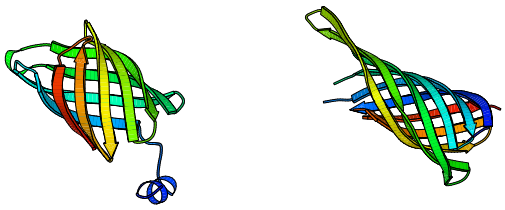


Fig. 18. The structures of (scop d1jmxa5 b.61.4.1) from the *Streptavidin* fold (left) and (scop d1qjpa_ f.4.1.1) from the *Transmembrane beta-barrel* fold (right).

immediately four good and different alignments appear (figure 19).

We have summarised the key numbers for the four alignments in figure 20. The table lists solution rank in FASE, sequence similarity, S-score, and *N* and *RMSD* values. Three numbers have been highlighted: The sequence identity of 12% for the so-called CP3 alignment, which stands out, since it is twice as high as the others sequence similarities found. This could indicate that the two structures are related by that particular alignment. Of the three other alignments we find that CP1 falls out as the weakest – the structural fit is worse than in the other cases (the *RMSD* is 2.3, which is the second highlighted number), and the similarity score is therefore also relative low (the S-score is 3.5 – the third highlighted number).

A sequential alignment program (such as sap [36] used in this case) will find the id alignment, because it is the only sequential one, and fits more structure that the sequential part of any other alignment (in this case CP1, CP2, and CP3). However, this alignment must be considered unlikely because of the low sequence similarity.

J. PreATP-grasp fold and DHS-like fold

In this final example we shall see a very clear indication of a CP similarity relative to the identity similarity, which is supported by structural as well as sequential evidence. The

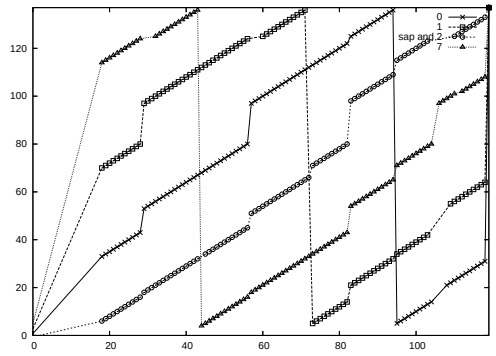


Fig. 19. Illustration of the four good and different alignments that can in general be made for two *beta-barrel* structures, and in particular for the pair under investigation: (scop d1jmxa5 b.61.4.1) from the *Streptavidin* fold and (scop d1qjpa_ f.4.1.1) from the *Transmembrane beta-barrel* fold. For each alignment its internal ranking in the FASE results list is given (see text for details).

	Rank	Seq. Sim. (%)	S-score	N/rmsd
id	2	6	2.9	94/1.8
CP1	7	5	**3.5**	96/**2.3**
CP2	1	5	2.6	92/1.6
CP3	0	**12**	2.7	98/1.8

Fig. 20. Key numbers for the four good and different alignments that can be made for the two *beta-barrels*: (scop d1jmxa5 b.61.4.1) and (scop d1qjpa_ f.4.1.1).

two structures in question (figure 21 and 22) is a domain from the *PreATP-grasp* fold (scop d1kjqa2 c.30.1.1) and a domain from the *DHS-like*-fold (scop d1o97d2 c.31.1.2). Both structures belong to the α/β-class with a central five stranded, parallel beta-sheet and 2-3 alpha helices on either side of the sheet.

The overall structure similarity of the CP alignment is very good (*N/RMSD* 77/2.37 *S* 4.44 *ID* 14) and has only got a few gaps, while the identity alignment has weaker structural similarity (*N/RMSD* 66/2.68 *S* 6.32 *ID* 6) and in general is less plausible with respect to all criteria used throughout this paper.

According to the CP hypothesis, the differences between the two structures must be explained by the event listed below: After cleavage between SSE 6 and 7 in structure A, the new 7-end curls up and back to form B's new beginning. Seen from the opposite side, a cleavage in the backbone of B creates the current termini in A. These subsequently change conformation and deviate from the "double-mini-helix" of d1o97d2 (see figure 22 top right). The assumption that ends may just change conformation and "curl up" relative to an earlier conformation after a cleavage is unrealistic: It is a known fact that protein termini are flexible [33] and may change conformation easier than other parts of the structure. For this reason they are e.g. routinely removed in x-ray crystallography studies. Thus, using the CP alignment hypothesis, the only difference between d1kjqa2 and d1o97d2, when observing on the fold level, concerns a "non-critical" part of

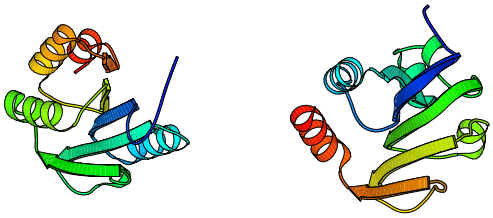


Fig. 21. The structures of *Ribonucleotide transformylase* (scop d1kjqa2) (left) and *Electron transfer flavoprotein* (scop d1o97d2) (right). The orientations of the structures matches the TOPS cartoons in figure 22.

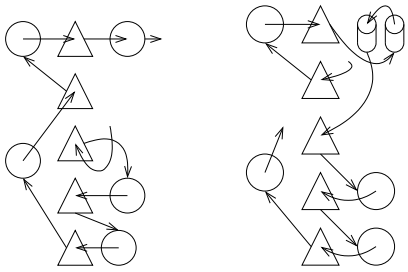


Fig. 22. TOPS cartoons for *Ribonucleotide transformylase* (scop d1kjqa2) (left) and *Electron transfer flavoprotein* (scop d1o97d2) (right).

the backbone. The identity alignment is less convincing, and more elaborate arguments must be used to explain a possible relation according to this alignment.

We also provide a dotplot of the two alignments (figure 23). From this, the CP is apparent, while identity alignment displays a more sporadic pattern which usually indicates a less good fit (the program sap finds a strictly sequential alignment, but with a less accurate fit).

The sequence identity analysis provides perhaps the final argument in this investigation: While the best CP alignments have values of 14, 14, 14, and 15%, the best identity solutions have values of 6, 4, and 9%. In summary, structural as well as sequential considerations (the SSSA measure in particular) strongly suggest a relation by CP as the most plausible relation between the two structures in question.

V. Results – Analysis of known and novel sub-tree CP relationships

In the analysis of novel CPs we have looked into different aspects of the found similarities such as overall structural similarity, CP score value, and sequence similarity after structural alignment.

Below, we will outline a "group" approach for analysis of possibly CP related sub-trees; this method is more robust than just analysing numbers from a single comparison. The approach can be used when analysing the relationship between two sub-trees of structures (not only a pair).

Each pair contributes with the top ranked solutions from the FASE list – the top 5 was used. For instance, if we have 20

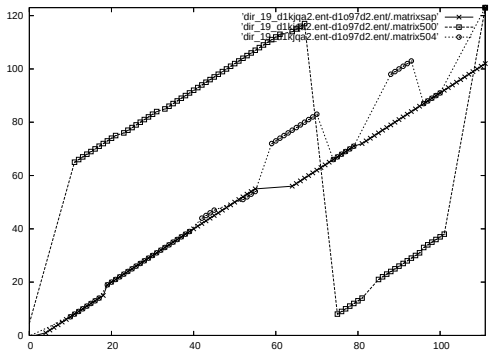


Fig. 23. Dotplot like illustration of the id and CP alignments of *Ribonucleotide transformylase* (scop d1kjqa2) (left) and *Electron transfer flavoprotein* (scop d1o97d2) (right).

comparisons of different pairs from two SCOP sub-trees, then we can plot $100 = 20{\cdot}5$ points in a S-score/sequence similarity plot, and check it for correlations and patterns. E.g. a good inverse correlation between these measures can be an indicator of homology: Alignments of structures from truly homologous sub-trees will most likely find any remote sequence similarity, while "unrelated" alignments will certainly not have any correlation between stronger structural similarity and higher sequence similarity. The absence of such a correlation is of course not a no-homology proof, but in some cases a found correlation can be very informative.

Similarly, the correlation between structural similarity and CP score can reveal if there is any CP (or perhaps if there exists two types of solutions) for the given protein pair: Identity alignments and CP alignments. If the scores obtained for the identity alignments have similar or better S-scores than the CPs, the authenticity of the claimed CP can be questioned.

A. Novel Relations

Of the novel CP relations previously identified in this study (listed in table 26 and figure 27), no. 4 and 6-13 have some degree of correlation between good structural similarity and high sequence identity. However, for no. 4, 6, 7, 9, 11, 13 we can observe a clearly increasing lower bound for sequence similarity when lowering the structural similarity scores. Similarly for no. 2, 18, and 19 there is no apparent inverse correlation between the two values. In most cases, the homology assessment made in this paper is in agreement with the correlation or lack of correlation between structure and sequence similarity.

Looking at the type of similarity found, a structure similarity/CP score plot can reveal if the homology is most likely to be an identity similarity or a CP similarity. The cases where good structural similarity is obtained with identity alignments are 1-4, 6, 8, 10-12, and 18-20. In all other cases, it is clear that the best structural fits are only obtained for high CP values.

Two of the novel CP relations (listed in table 24 as no. 15 and 16) between sub-trees have been detected as a result of the comparisons described in section IV-H. The sub-trees

are related by structural similarity scores of 5.36 and 6.44, sequence similarity in the alignment of 13.5% and 14.7%, and CP scores above 0.25. Their S/CP plots clearly shows an inverse correlation that indicates that the relatedness can be attributed to the CP. We will use these two sub-tree relations as a reference point below when comparing the known CP relations to the novel ones.

B. Known Relations

Of the known CPs relations annotated in SCOP (in table 24) no. 1-6, 8, 10-12, 31, 33, and 36 and are all very clear homologs in terms of structural similarity scores and clear CPs looking at the CP scores. This group has 13 pairs. Furthermore, in support of their evolutionary relatedness is the fact that all the sub-trees have increasingly higher sequence identities for better structural similarity scores. The weakest similarity in this group is found for the well-known *swaposin* family, which has an average structural similarity (S-score) of 3.57 between two structures in the related sub-trees. Thus, the above list of sub-trees are all strongly related.

Yet another group includes all the pairs that have roughly the same structural similarity scores, sequence similarities and CP scores as one of the strongest novel CPs – namely no. 15 between *MurD* and *CheY* (mentioned in the previous section, V-A). This group includes (from table 24) no. 7, 19-20, 24-25, 29, 32, and 34-35 – a total of 9 pairs. For no. 7, the S/Sid correlation is not perfect, but still best S score corresponds to relatively high Sid scores. No. 19, has fine S/Sid correlations indicating homology. However, good S-scores correlate with CP values of zero, so it seems likely that this relation is given by a simple identity alignment rather than as a CP relation.

The group of sub-tree pairs whose similarities are worse than the similarity between between *MurD* and *CheY* comprise no. 9, 15-18, 21-23, 26-28, and 30. This group has 12 pairs. In this group, only no. 15-16, 21-23, and 26 have good correlations between the S-score and sequence similarity in support of their homology. For the rest in the group, the sequence similarity for the top 5 found similarities are independent of the structural similarity score.

Of the total 36 known relations 12 have worse similarities than the most promising novel one, and 9 have app. the same similarity. That is, a number between 33% and 58% of the known CPs are less convincing than a representative, *MurD* and *CheY*, of the most promising novel ones.

VI. Discussion and Conclusion

The present study applied a new method called FASE (for structural similarity assessment of protein structures) to the detection of novel CPs in the SCOP database. FASE is a fully automated and general method, that can detect non-sequential alignments of proteins, which is a necessary ability in the detection of CPs. The detection and evaluation was based on a number of measures that are known to be indicators of homology. These are: Structural similarity, sequence similarity after structural alignment (SSSA), and a novel CP measure for assessing the validity of the CP relation implied by the similarities found.

CPs are of interest for mainly two reasons: The study of CPs can provide insights into the evolution of protein structures and ultimately aid in understanding protein folding; moreover CPs can be used in protein engineering experiments to (for instance) ease the determination of the 3D conformation of a protein structure or binding pocket.

There are probably many novel CPs left to be discovered in the structural data available today. Firstly, for efficiency purposes, we restricted ourselves to the SCOP40 and then SCOP40s databases. It is obvious that the 40% selection criterion removes many potentially interesting CP candidates. Moreover, the selection of domains between 100 and 250 residues also weeds out some possible novel pairs; this restriction was necessary for efficiency, as the number of comparisons otherwise would have been large to execute within reasonable time. Moreover, the thresholds from the different kinds of similarity measures were set very conservatively to avoid having to weed out too many false positives by manual inspection. This however also means that many false negatives (that is true CPs) will be weeded out at this point, and consequently never be manually detected. Also, new CPs might be found by looking through the list of 931 pairs in a more exhaustive manner. In conclusion, by increasing the amount of manual effort, more new CPs could probably be found. Because the used measures and scores do not fully capture the essence of relatedness and CP realtions, a great deal of manual verification is still needed.

In spite of the above mentioned possibilities for missing the detection of novel CPs, we have still managed to find a list of 20 pairs of possibly CP related structure pairs. By analysing the similarities for each pair and similarities between structures in the two SCOP sub-trees that they connect with their similarity, we have been able to further narrow down the list to a number of particularly interesting pairs.

We found 5 pairs of moderate to high interest and plausibility; these were no. 8, 11, 12, 19, 20 in table 26. The 6 most intersting pairs were no. 3, 4, 9, 14, 15, and 16 from table 26. The average similarity between structures in the sub-trees was low (although not the lowest found). More important is the fact that the CP alignments are convincing relative to any potential identity alignment as measured by the S-scores of the respective similarities. Also, high sequence similarity was found for the novel relations.

The characteristics of these related sub-trees were more pronounced than for about half of the known CP related families in SCOP. The detailed relation of putative novel CPs to known ones, has rarely been used when arguing for the CP of two classes of structures, and does substantiate the CP claims made in this paper significantly precisely by relating the novel findings to an accepted 'gold' standard.

References

[1] Patrick Aloy, Baldomero Oliva, Enrique Querol, Francesc X Aviles, and Robert B Russell. Structural similarity to link sequence space: new potential superfamilies and implications for structural genomics. *Protein Sci*, Vol. 11, No. 5, pp. 1101–1116, May 2002.

[2] M Bajaj and T Blundell. Evolution and the tertiary structure of proteins. *Annu Rev Biophys Bioeng*, Vol. 13, pp. 453–492, 1984.

[3] S E Brenner, P Koehl, and M Levitt. The ASTRAL compendium for protein structure and sequence analysis. *Nucleic Acids Res*, Vol. 28, No. 1, pp. 254–256, Jan 2000.

[4] Richard R. Copley and Peer Bork. Homology among ([beta][alpha])8 barrels: implications for the evolution of metabolic pathways. *Journal of Molecular Biology*, Vol. 303, No. 4, pp. 627–641, November 2000.

[5] T E Creighton. *Proteins: Structures and Molecular Properties*. Freeman, New York), 1993.

[6] B A Cunningham, Hemperly J J, Hopp T P, and Edelman G M. Favin Versus Concanavalin A: Circularly Permuted Amino Acid Sequences. *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 76, No. 7, pp. 3218–3222, Jul 1979.

[7] Pearl F.M.G, Lee D., Bray J.E, Sillitoe I., Todd A.E., Harrison A.P., Thornton J.M., and Orengo C.A. Assigning genomic sequences to cath. *Nucleic Acids Research*, Vol. 28, No. 1, pp. 277–282, 2000.

[8] D P Goldenberg and T E Creighton. Circular and circularly permuted forms of bovine pancreatic trypsin inhibitor. *J Mol Biol*, Vol. 165, No. 2, pp. 407–413, Apr 1983.

[9] Nick V. Grishin. Fold change in evolution of protein structures. *Journal of Structural Biology*, Vol. 134, pp. 167–185, 2001.

[10] M Hahn, K Piotukh, R Borriss, and U Heinemann. Native-like in vivo folding of a circularly permuted jellyroll protein shown by crystal structure analysis. *Proc Natl Acad Sci U S A*, Vol. 91, No. 22, pp. 10417–10421, Oct 1994.

[11] U Heinemann and M Hahn. Circular permutation of polypeptide chains: implications for protein folding and stability. *Prog Biophys Mol Biol*, Vol. 64, No. 2-3, pp. 121–143, 1995.

[12] J Heringa and W R Taylor. Three-dimensional domain duplication, swapping and stealing. *Curr Opin Struct Biol*, Vol. 7, No. 3, pp. 416–421, Jun 1997.

[13] A Jeltsch. Circular permutations in the molecular evolution of DNA methyltransferases. *J Mol Evol*, Vol. 49, No. 1, pp. 161–164, Jul 1999.

[14] J Jia, W Huang, U Schorken, H Sahm, G A Sprenger, Y Lindqvist, and G Schneider. Crystal structure of transaldolase B from Escherichia coli suggests a circular permutation of the alpha/beta barrel within the class I aldolase family. *Structure*, Vol. 4, No. 6, pp. 715–724, Jun 1996.

[15] J Jung and B Lee. Circularly permuted proteins in the protein structure database. *Protein Sci*, Vol. 10, No. 9, pp. 1881–1886, Sep 2001.

[16] Ryotaro Koike, Kengo Kinoshita, and Akinori Kidera. Probabilistic description of protein alignments for sequences and structures. *Proteins*, Vol. 56, No. 1, pp. 157–166, Jul 2004.

[17] Y Lindqvist and G Schneider. Circular permutations of natural protein sequences: structural evidence. *Curr Opin Struct Biol*, Vol. 7, No. 3, pp. 422–427, Jun 1997.

[18] G. Lu. Top: a new method for protein structure comparisons and similarity searches. *J. Appl. Cryst*, Vol. 33, pp. 176–183, 2000.

[19] K Luger, U Hommel, M Herold, J Hofsteenge, and K Kirschner. Correct folding of circularly permuted variants of a beta alpha barrel enzyme in vivo. *Science*, Vol. 243, No. 4888, pp. 206–210, Jan 1989.

[20] A Lupas. A circular permutation event in the evolution of the SLH domain? *Mol Microbiol*, Vol. 20, No. 4, pp. 897–898, May 1996. Letter.

[21] A N Lupas, C P Ponting, and R B Russell. On the evolution of protein folds: are similar motifs in different protein folds the result of convergence, insertion, or relics of an ancient peptide world? *J Struct Biol*, Vol. 134, No. 2-3, pp. 191–203, May 2001.

[22] E A MacGregor, H M Jespersen, and B Svensson. A circularly permuted alpha-amylase-type alpha/beta-barrel structure in glucan-synthesizing glucosyltransferases. *FEBS Lett*, Vol. 378, No. 3, pp. 263–266, Jan 1996.

[23] A G Murzin. Sweet-tasting protein monellin is related to the cystatin family of thiol proteinase inhibitors. *J Mol Biol*, Vol. 230, No. 2, pp. 689–694, Mar 1993.

[24] A G Murzin. Probable circular permutation in the flavin-binding domain. *Nat Struct Biol*, Vol. 5, No. 2, pp. 101, Feb 1998. Comment.

[25] E A Nalefski and J J Falke. The C2 domain calcium-binding motif: structural and functional diversity. *Protein Sci*, Vol. 5, No. 12, pp. 2375–2390, Dec 1996.

[26] G Polekhina, P G Board, R R Gali, J Rossjohn, and M W Parker. Molecular basis of glutathione synthetase deficiency and a rare gene permutation event. *EMBO J*, Vol. 18, No. 12, pp. 3204–3213, Jun 1999.

[27] C P Ponting and R B Russell. Swaposins: circular permutations within genes encoding saposin homologues. *Trends Biochem Sci*, Vol. 20, No. 5, pp. 179–180, May 1995. Letter.

[28] C P Ponting and R B Russell. Identification of distant homologues of fibroblast growth factors suggests a common ancestor for all beta-trefoil proteins. *J Mol Biol*, Vol. 302, No. 5, pp. 1041–1047, Oct 2000.

[29] G N Jr Reeke and J W Becker. Three-dimensional structure of favin: saccharide binding-cyclic permutation in leguminous lectins. *Science*, Vol. 234, No. 4780, pp. 1108–1111, Nov 1986.

[30] E. Ribeiro and C. Ramos. Circular Permutation and Deletion Studies of Myoglobin Indicate that the Correct Position of Its N-Terminus Is Required for Native Stability and Solubility but Not for Native-like Heme Binding and Folding. *Biochemistry*, Vol. 4, pp. 4699–4709, 2005.

[31] R B Russell and C P Ponting. Protein fold irregularities that hinder sequence analysis. *Curr Opin Struct Biol*, Vol. 8, No. 3, pp. 364–371, Jun 1998.

[32] R B Russell, M A Saqi, R A Sayle, P A Bates, and M J Sternberg. Recognition of analogous and homologous protein folds: analysis of sequence and structure conservation. *J Mol Biol*, Vol. 269, No. 3, pp. 423–439, Jun 1997.

[33] Thomas U Schwartz, Rudolf Walczak, and Gunter Blobel. Circular permutation as a tool to reduce surface entropy triggers crystallization of the signal recognition particle receptor beta subunit. *Protein Sci*, Vol. 13, No. 10, pp. 2814–2818, Oct 2004.

[34] Y Sergeev and B Lee. Alignment of beta-barrels in (beta/alpha)8 proteins using hydrogen-bonding pattern. *J Mol Biol*, Vol. 244, No. 2, pp. 168–182, Nov 1994.

[35] Johannes Soding and Andrei N Lupas. More than the sum of their parts: on the evolution of proteins from peptides. *Bioessays*, Vol. 25, No. 9, pp. 837–846, Sep 2003.

[36] W R Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Sci*, Vol. 8, No. 3, pp. 654–665, Mar 1999.

[37] W. R. Taylor and A. Aszodi. *Protein Geometry, Classification, Topology and Symmetry*. Institute of Physics Publishing, 2005.

[38] S Uliel, A Fliess, and R Unger. Naturally occurring circular permutations in proteins. *Protein Eng*, Vol. 14, No. 8, pp. 533–542, Aug 2001.

[39] J Vesterstrø m and W R Taylor. Flexible Secondary Structure Based Protein Structure Comparison Applied to the Detection of Circular Permutation. *Journal of Computational Biology*, Vol. 99, No. 99, pp. 999–9999, XXX 2005.

[40] S Xu, J Xiao, J Posfai, R Maunus, and J 2nd Benner. Cloning of the BssHII restriction-modification system in Escherichia coli : BssHII methyltransferase contains circularly permuted cytosine-5 methyltransferase motifs. *Nucleic Acids Res*, Vol. 25, No. 20, pp. 3991–3994, Oct 1997.

	Sub-Tree 1			*Sub-Tree 2*			*Similarities*			
no	sunid	sccs	name	sunid	sccs	name	n	Score	Seq	CP
01	17496	a.44.1.1	d1dyva1	17497	a.44.1.1	d1dsba1	1	0.52 (-)	100.00 (-)	-.– (-)
02	24488	b.34.2.1	d1g2ba_	50059	b.34.2.1	Chicken (Gall	11	1.02 (0.9)	94.18 (8.0)	0.68 (0.1)
03	24489	b.34.2.1	d1tud__	50059	b.34.2.1	Chicken (Gall	11	0.66 (0.2)	96.45 (3.7)	0.69 (0.0)
04	24492	b.34.2.1	d1tuc__	50059	b.34.2.1	Chicken (Gall	11	1.47 (0.6)	96.36 (3.9)	0.57 (0.1)
05	32790	c.47.1.4	d1dyva2	32791	c.47.1.4	d1dsba2	1	1.07 (-)	100.00 (-)	-.– (-)
06	42707	e.3.1.1	d1alq__	56611	e.3.1.1	Staphylococcu	11	0.38 (0.1)	99.09 (0.3)	0.47 (0.0)
07	46887	a.4.5.25	Methionine am	46785	a.4.5	”Winged helix	20	4.60 (2.1)	15.30 (3.9)	0.19 (0.1)
08	47866	a.64.1.2	Swaposin	47862	a.64.1	Saposin	12	3.57 (1.3)	28.17 (9.6)	0.55 (0.3)
!09	49599	b.8.1	TRAF domain-l	48725	b.1	Immunoglobuli	20	6.78 (0.8)	10.15 (2.6)	0.20 (0.0)
!10	49704	b.11.1.1	Rat (Rattus n	49703	b.11.1.1	Cow (Bos taur	17	1.34 (0.3)	52.65 (27.5)	0.60 (0.2)
11	49901	b.29.1.1	Concanavalin	49900	b.29.1.1	Legume lectin	20	1.64 (0.2)	42.60 (3.1)	0.76 (0.3)
!12	49929	b.29.1.2	Bacillus mace	49929	b.29.1.2	Bacillus mace	20	0.55 (0.2)	97.30 (2.7)	0.58 (0.3)
!13	50343	b.40.7.1	Histidine kin	69271	b.40.7.1	Chemotaxis pr	2	2.98 (0.1)	20.50 (2.1)	0.40 (0.0)
!14	50449	b.43.3.1	Elongation fa	50448	b.43.3.1	Elongation fa	18	2.06 (0.7)	28.17 (10.0)	0.35 (0.1)
15	51419	c.1.6	PLP-binding b	51350	c.1	TIM beta/alph	20	7.71 (2.1)	10.70 (2.2)	0.23 (0.1)
16	51588	c.1.10.1	Transaldolase	51570	c.1.10.1	Class I aldol	20	6.71 (0.6)	10.50 (2.5)	0.20 (0.0)
17	51731	c.1.23.1	Methylenetetr	82279	c.1.23.2	Proline dehyd	3	6.84 (0.5)	8.33 (0.6)	0.26 (0.0)
18	52331	c.23.16.4	Aspartyl dipe	52317	c.23.16	Class I gluta	19	6.74 (1.3)	12.95 (1.3)	0.15 (0.0)
!19	52460	c.30.1.4	Eukaryotic gl	52457	c.30.1.3	Prokaryotic g	13	4.60 (0.2)	13.92 (1.6)	0.17 (0.1)
20	52587	c.37.1.6	Pantothenate	31952	c.37.1.6	d1a7j__	7	4.21 (0.3)	16.86 (0.7)	0.20 (0.0)
21	52799	c.45.1	(Phosphotyros	52788	c.44.1	Phosphotyrosi	20	11.57 (2.1)	11.70 (2.3)	0.11 (0.0)
22	52821	c.46.1	Rhodanese/Cel	52788	c.44.1	Phosphotyrosi	19	8.30 (1.4)	12.11 (2.3)	0.18 (0.0)
23	53213	c.56.6	LigB subunit	53162	c.56	Phosphorylase	20	8.14 (1.3)	11.65 (3.3)	0.17 (0.0)
24	53377	c.66.1.11	Type II DNA	53334	c.66	S-adenosyl-L-	20	6.36 (1.9)	15.60 (3.6)	0.27 (0.1)
25	54842	d.55	Ribosomal pro	54825	d.54	Enolase N-ter	20	5.82 (0.7)	16.10 (2.5)	0.24 (0.0)
26	55186	d.67.1	Threonyl-tRNA	64294	d.185.1.2	Autoinducer-2	11	7.80 (0.9)	12.73 (2.2)	0.16 (0.0)
!27	55926	d.127.1.1	Archaeon Pyro	46785	a.4.5	”Winged helix	20	19.47 (6.0)	16.20 (6.9)	0.04 (0.0)
!28	55927	d.127.1.1	Human (Homo s	46785	a.4.5	”Winged helix	20	20.33 (7.9)	13.15 (3.2)	0.05 (0.0)
29	56088	d.142.1.6	Eukaryotic gl	56061	d.142.1.1	Prokaryotic g	10	7.86 (4.7)	12.50 (0.8)	0.32 (0.1)
30	56091	d.142.2	DNA ligase/mR	56059	d.142.1	Glutathione s	19	10.17 (2.8)	11.58 (2.8)	0.13 (0.0)
31	60058	c.86.1.1	d1fw8a_	53751	c.86.1.1	Baker’s yeast	2	2.45 (0.0)	84.00 (9.9)	0.48 (0.0)
32	64153	c.104.1	YjeF N-termin	75292	c.72.1.4	YjeF C-termin	2	9.78 (0.4)	14.50 (0.7)	0.05 (0.0)
33	79717	b.89.1.1	d1n02a_	51325	b.89.1.1	Cyanobacteriu	13	2.47 (0.7)	90.62 (1.2)	0.17 (0.4)
!34	82199	b.85.7	SET domain	88466	b.85.1.1	d1ucsa_	16	9.17 (3.0)	19.19 (5.4)	0.38 (0.1)
35	89669	c.37.1.8	Probable GTPa	52592	c.37.1.8	G proteins	20	5.13 (1.1)	16.65 (2.0)	0.22 (0.0)
!36	90168	g.15.1.2	Wound-induced	57486	g.15.1.2	Plant protein	5	3.30 (1.3)	64.60 (5.1)	0.55 (0.1)

Fig. 24. Statistics for the solutions found for sample of max 20 comparisons from 36 pairs of sub-trees related by a CP according to ***SCOP***. Here, the statistics are not for the best solution found, but the most advantageous solution of the five best according to the measure in question. CP scores for relations 1 and 5 have been left out because d1dyva1 and d1dyva2 have been been re-permuted in SCOP, which means that the FASE comparison output indicates identity alignment despite the very clear CP. Moreover, the standard deviations in parentheses have been left out for the comparisons of the sub-trees with only one comparison.

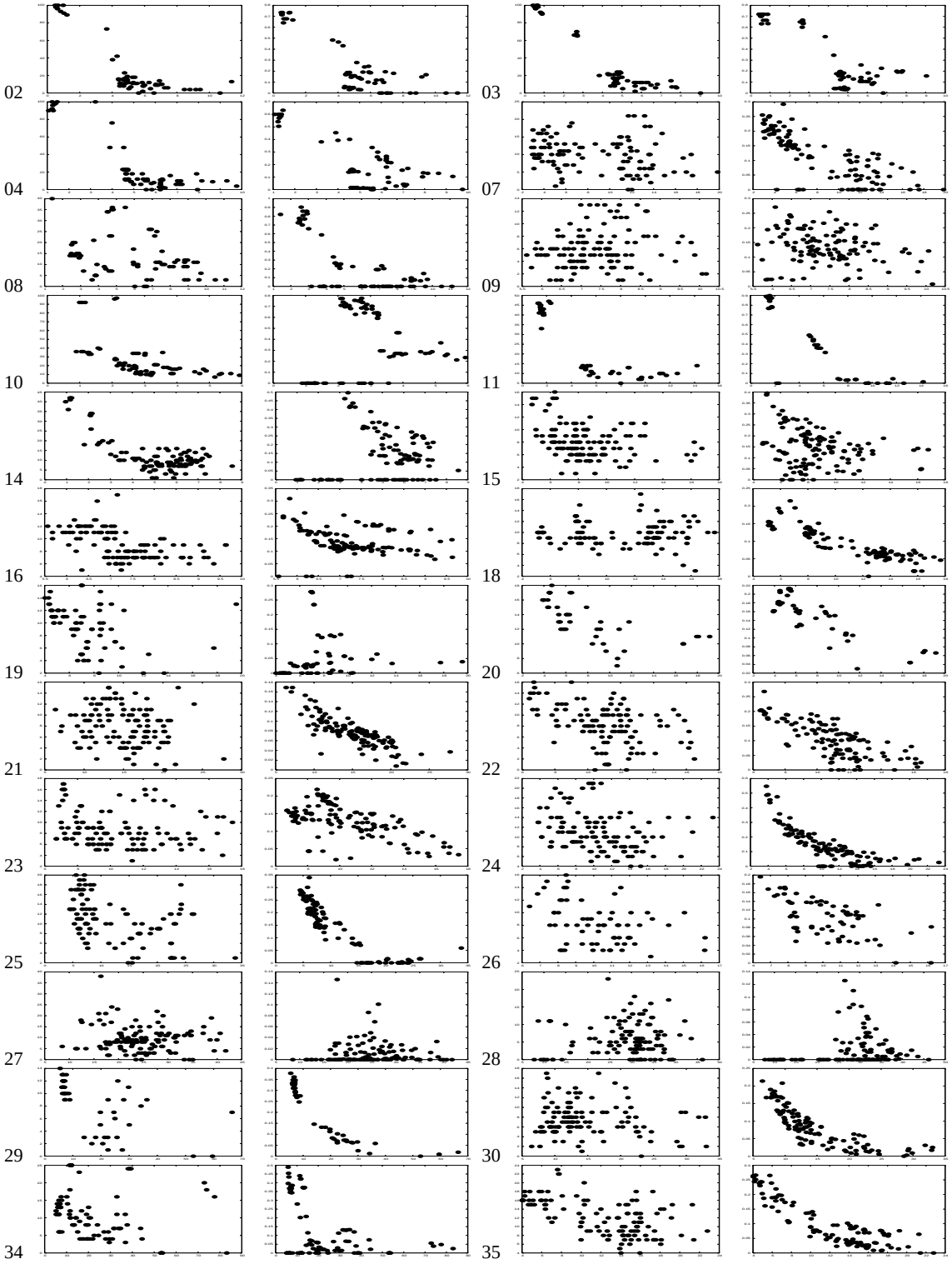


Fig. 25. Correlation between Score and Seq.id (left) and Score and CP (right) calculated over the 5 best found solutions taken from a sample of max 20 comparisons from 36 pairs of sub-trees related by a CP according to ***SCOP***.

	Sub-Tree 1			*Sub-Tree 2*			*Similarities*			
no	sunid	sccs	name	sunid	sccs	name	n	Score	Seq	CP
01	46579	a.2.5	Prefoldin	46988	a.7.5	Tubulin chape	9	3.85 (1.9)	14.56 (2.3)	0.37 (0.2)
02	47266	a.26.1	4-helical cyt	47240	a.25.1	Ferritin-like	40	8.33 (4.9)	10.55 (2.4)	0.19 (0.1)
03	48726	b.1.1	Immunoglobuli	49594	b.7.4	Rab geranylge	20	4.74 (0.4)	10.60 (2.5)	0.26 (0.1)
04	49472	b.3.4	Transthyretin	48726	b.1.1	Immunoglobuli	20	5.92 (0.7)	12.85 (3.1)	0.24 (0.0)
05	49723	b.12.1	Lipase/lipoox	49785	b.18.1	Galactose-bin	20	7.25 (1.4)	10.65 (1.9)	0.18 (0.1)
06	49899	b.29.1	Concanavalin	49899	b.29.1	Concanavalin	20	5.07 (3.2)	23.15 (16.2)	0.31 (0.3)
07	49904	b.29.1.1	Legume lectin	49901	b.29.1.1	Concanavalin	20	1.48 (0.2)	43.90 (2.8)	0.84 (0.1)
08	50370	b.42.2	Ricin B-like	50405	b.42.5	Actin-crossli	19	2.28 (0.2)	13.21 (1.9)	0.63 (0.0)
09	50729	b.55.1	PH domain-lik	55785	d.110.3	PYP-like sens	20	8.76 (1.8)	11.20 (2.0)	0.15 (0.1)
10	50814	b.60.1	Lipocalins	50876	b.61.1	Avidin/strept	20	6.06 (1.0)	13.20 (3.7)	0.25 (0.1)
11	51366	c.1.2	Ribulose-phos	51569	c.1.10	Aldolase	20	4.89 (1.1)	13.45 (2.5)	0.33 (0.1)
12	51375	c.1.2.3	Decarboxylase	51381	c.1.2.4	Tryptophan bi	20	4.20 (0.6)	13.80 (2.3)	0.40 (0.1)
13	52343	c.25.1	Ferredoxin re	52172	c.23.1	CheY-like	20	6.64 (1.3)	10.75 (3.5)	0.27 (0.1)
14	52440	c.30.1	PreATP-grasp	52467	c.31.1	DHS-like NAD/	20	8.75 (3.0)	15.90 (4.1)	0.21 (0.1)
15	53244	c.59.1	MurD-like pep	52172	c.23.1	CheY-like	20	5.36 (0.9)	13.50 (2.9)	0.27 (0.1)
16	53244	c.59.1	MurD-like pep	53218	c.57.1	Molybdenum co	20	6.44 (0.8)	14.70 (2.4)	0.26 (0.1)
17	55257	d.74.3	RBP11-like su	54991	d.58.13	Anticodon-bin	19	5.31 (0.6)	11.58 (2.2)	0.32 (0.1)
18	55620	d.96.1	Tetrahydrobio	54821	d.53.1	Ribosomal pro	20	6.20 (1.1)	14.10 (2.2)	0.24 (0.1)
19	69298	b.61.4	Quinohemoprot	56925	f.4.1	OMPA-like	16	3.86 (1.0)	10.88 (2.3)	0.46 (0.1)
20	75500	d.94.2	Putative tran	54913	d.58.5	GlnB-like	13	5.11 (0.5)	14.62 (2.9)	0.40 (0.1)

Fig. 26. Statistics for the solutions found for sample of max 20 comparisons from 20 pairs of sub-trees related by a CP according to ***novel findings presented in this paper***. Here, the statistics are not for the best solution found, but the most advantageous solution of the five best according to the measure in question. Standard deviations are given in the parentheses.

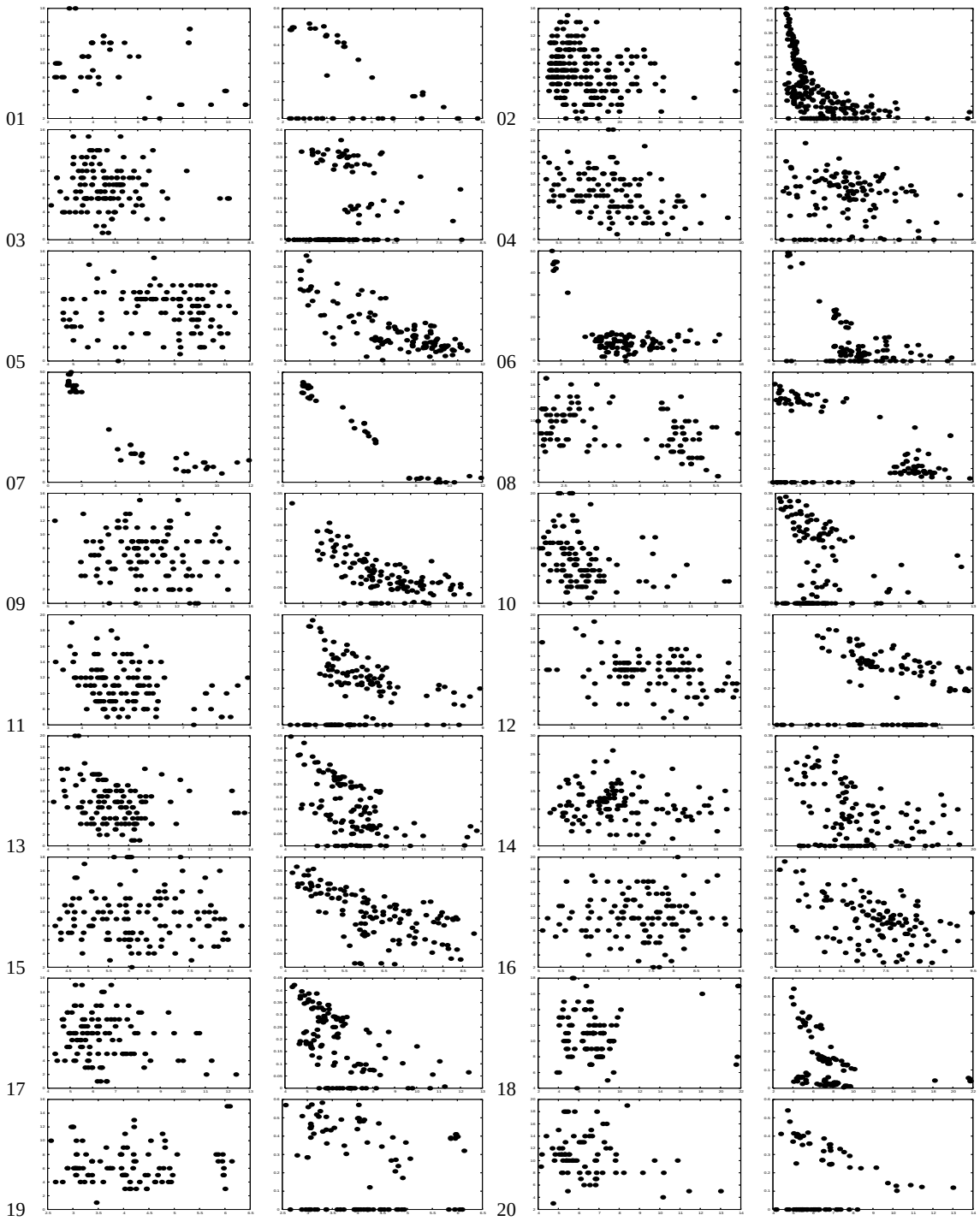


Fig. 27. Correlation between Score and Seq.id (left) and Score and CP (right) calculated over the 5 best found solutions taken from a sample of max 20 comparisons from 20 pairs of sub-trees related by a CP according to ***novel findings presented in this paper***.

Bibliography

[1] B. Alberts. *Molecular biology of the cell*. Garland Science, 4 edition, 2002.

[2] N. Alexandrov. Sarfing the pdb. *Protein Engineering*, 9(9):727–732, 1996.

[3] F. Alizadeh, R. M. Karp, D. K. Weisser, and G. Zweig. Physical mapping of chromosomes using unique probes. In *Symposium on Discrete Algorithms*, pages 489–500, 1994.

[4] P. Aloy, B. Oliva, E. Querol, F. X. Aviles, and R. B. Russell. Structural similarity to link sequence space: new potential superfamilies and implications for structural genomics. *Protein Sci*, 11(5):1101–1116, May 2002.

[5] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, Sep 1997.

[6] A. Andreeva, D. Howorth, S. E. Brenner, T. J. P. Hubbard, C. Chothia, and A. G. Murzin. SCOP database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Res*, 32(Database issue):226–229, Jan 2004.

[7] C. B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(96):223–230, Jul 1973.

[8] P. J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *Lecture Notes in Computer Science 1447*, Evolutionary Programming VII, pages 601–610. Springer, 1998.

[9] J. Ay, M. Hahn, K. Decanniere, K. Piotukh, R. Borriss, and U. Heinemann. Crystal structures and properties of de novo circularly permuted 1,3-1,4-beta-glucanases. *Proteins*, 30(2):155–167, Feb 1998.

[10] O. Bachar, D. Fischer, R. Nussinov, and H. Wolfson. A computer vision based technique for 3-D sequence-independent structural comparison of proteins. *Protein Eng*, 6(3):279–288, Apr 1993.

[11] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[12] M. Bajaj and T. Blundell. Evolution and the tertiary structure of proteins. *Annu Rev Biophys Bioeng*, 13:453–492, 1984.

[13] M. J. Bennett, S. Choe, and D. Eisenberg. Domain swapping: entangling alliances between proteins. *Proceedings of the National Academy of Sciences USA*, 91(8):3127–3131, Apr 1994.

[14] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Res*, 28(1):235–242, Jan 2000.

[15] M. Berthold and D. Hand, editors. *Intelligent Data Analysis*. Springer, 2003.

[16] S. Bhandarkar and S. Machaka. Chromosome reconstruction from physical maps using a cluster of workstations. *The Journal of Supercomputing*, 11(1):61–86, 1997.

[17] P. E. Black, editor. *Algorithms and Theory of Computation Handbook*, pages 1–26. CRC Press LLC, 1999. Internet: http://www.nist.gov/dads/HTML/greedyHeuristic.html and http://www.nist.gov/dads/HTML/greedyalgo.html.

[18] K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13:335– 379, 1976.

[19] S. E. Brenner, P. Koehl, and M. Levitt. The ASTRAL compendium for protein structure and sequence analysis. *Nucleic Acids Res*, 28(1):254–256, Jan 2000.

[20] A. Caprara. Structural alignment of large-size proteins via lagrangian relaxation. In *RECOMB*, pages 100–108, 2002.

[21] O. Carugo and S. Pongor. Protein fold similarity estimated by a probabilistic approach based on ca-ca distance comparison. *J. Mol. Biol.*, 315:887–898, 2002.

[22] L. P. Chew, D. Huttenlocher, K. Kedem, and J. Kleinberg. Fast detection of common geometric substructure in proteins. *Journal of Computational Biology*, 6(3-4):313–325, 1999.

[23] C. Chothia and A. M. Lesk. The relation between the divergence of sequence and structure in proteins. *EMBO J*, 5(4):823–826, Apr 1986.

[24] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, GSIA, Carnegie-Mellon University, 1976.

[25] D. Clark. *Evolutionary Algorithms in Molecular Design*. Wiley-VCH, Weinheim, 2000.

[26] F. E. Cohen and M. J. Sternberg. On the prediction of protein structure: The significance of the root-mean-square deviation. *J Mol Biol*, 138(2):321–333, Apr 1980.

[27] F. S. Collins, M. Morgan, and A. Patrinos. The human genome project: Lessons from large-scale biology. *Science*, 300(5617):286–290, 2003.

[28] R. R. Copley and P. Bork. Homology among $\beta\alpha_8$ barrels: implications for the evolution of metabolic pathways. *Journal of Molecular Biology*, 303(4):627–641, November 2000.

[29] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 1998.

[30] T. E. Creighton. *Proteins: Structures and Molecular Properties*. Freeman, New York), 1993.

[31] B. A. Cunningham, J. J. Hemperly, T. P. Hopp, and G. M. Edelman. Favin Versus Concanavalin A: Circularly Permuted Amino Acid Sequences. *Proceedings of the National Academy of Sciences USA*, 76(7):3218–3222, Jul 1979.

[32] A. Cuticchia. The use of simulated annealing in chromosome reconstruction experiments based on binary scoring. *Genetics*, 132(2):591–601, 1992.

[33] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons., 2001.

[34] C. Dennis and R. Gallagher. *The human genome*. Palgrave, 2001.

[35] O. Dror, H. Benyamini, R. Nussinov, and H. J. Wolfson. MASS: multiple structural alignment by secondary structures. *Bioinformatics*, 19(Suppl. 1):95–104, 2003.

[36] O. Dror, H. Benyamini, R. Nussinov, and H. J. Wolfson. Multiple structural alignment by secondary structures: Algorithm and applications. *Protein Science*, 12:2492–2507, 2003.

[37] I. Eidhammer, I. Jonassen, and W. R. Taylor. Structure comparison and structure patterns. *J Comput Biol*, 7(5):685–716, 2000.

[38] I. Eidhammer, I. Jonassen, and W. R. Taylor. *Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis*. Wiley, 2004.

[39] J. Enkerli, H. Reed, A. Briley, G. Bhatt, and S. Covert. Physical Map of a Conditionally Dispensable Chromosome in Nectria haematococca Mating Population VI and Location of Chromosome Breakpoints. *Genetics*, 155(3):1083–1094, 2000.

[40] A. Falicov and F. E. Cohen. A surface of minimum area metric for the structural comparison of proteins. *Journal of Molecular Biology*, 258:871–892, 1996.

[41] A. Fersht. *Structure and Mechanism in Protein Science*. W. H. Freeman and Company, New York, 1999.

[42] R. A. Finkel and J. L. Bentley. Quad-trees: a data structure for retrieval on composite keys. *Informatica*, 4:1–9, 1974.

[43] T. P. Flores, D. S. Moss, and J. M. Thornton. An algorithm for automatically generating protein topology cartoons. *Protein Eng*, 7(1):31–37, Jan 1994.

[44] D. B. Fogel, editor. *Evolutionary Computation: The Fossil Record*. IEEE Press, 1998.

[45] L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Callahan, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems: Proc. of the 2nd Cybernetic Sciences Symposium*, pages 131–155. Spartan Books, 1965.

[46] S. Garcia-Vallve, A. Rojas, J. Palau, and A. Romeu. Circular permutants in beta-glucosidases (family 3) within a predicted double-domain topology that includes a $\beta\alpha_8$ barrel. *Proteins*, 31(2):214–223, May 1998.

[47] M. Gerstein and M. Levitt. Using iterative dynamic programming to obtain accurate pairwise and multiple alignments of protein structures. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, pages 59–67. AAAI Press, 1996.

[48] M. Gerstein and M. Levitt. Comprehensive assessment of automatic structural alignment against a manual standard, the scop classification of proteins. *Protein Sci*, 7(2):445–456, 1998.

[49] J. F. Gibrat, T. Madej, and S. H. Bryant. Surprising similarities in structure comparison. *Curr. Opin. Struct. Biol*, 6:377–385, 1996.

[50] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass, 1989.

[51] P. W. Goldberg, C. Golumbic, Martin, H. Kaplan, and R. Shamir. Four strikes against physical mapping. *Journal of Computational Biology*, 2(1), 1995.

[52] D. P. Goldenberg and T. E. Creighton. Circular and circularly permuted forms of bovine pancreatic trypsin inhibitor. *J Mol Biol*, 165(2):407–413, Apr 1983.

[53] D. Greenberg and S. Istrail. Physical mapping by STS hybridization: algorithmic stratgies and the challenge of software evaluation. *J. Comp. Biol.*, 2(2):219–273, 1995.

[54] D. R. Greening. *Parallel Simulated Annealing Techniques*, pages 293–306. Emergent Computation. MIT/North Holland, 1991.

[55] H. M. Grindley, P. J. Artymiuk, D. W. Rice, and P. Willett. Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm. *Journal of Molecular Biology*, 229:707–720, 1993.

[56] N. V. Grishin. Fold change in evolution of protein structures. *Journal of Structural Biology*, 134:167–185, 2001.

[57] M. Hahn, K. Piotukh, R. Borriss, and U. Heinemann. Native-like in vivo folding of a circularly permuted jellyroll protein shown by crystal structure analysis. *Proceedings of the National Academy of Sciences USA*, 91(22):10417–10421, Oct 1994.

[58] D. Hall, S. Bhandarkar, and J. Wang. ODS2: A Multiplatform Software Application for Creating Integrated Physical and Genetic Maps. *Genetics*, 157(3):1045–1056, 2001.

[59] E. Harley, A. Bonner, and N. Goodman. Revealing hidden interval graph structure in STS-content data. *Bioinformatics*, 15(4):278–285, 1999.

[60] A. Harrison, F. Pearl, I. Sillitoe, T. Slidel, R. Mott, J. Thornton, and C. Orengo. Recognizing the fold of a protein structure. *Bioinformatics*, 19(14):1748–1759, 2003.

[61] U. Heinemann and M. Hahn. Circular permutation of polypeptide chains: implications for protein folding and stability. *Prog Biophys Mol Biol*, 64(2-3):121–143, 1995.

[62] U. Heinemann and M. Hahn. Circular permutations of protein sequence: not so rare? *Trends Biochem Sci*, 20(9):349–350, Sep 1995. Comment.

[63] J. Heringa and W. R. Taylor. Three-dimensional domain duplication, swapping and stealing. *Curr Opin Struct Biol*, 7(3):416–421, Jun 1997.

[64] B. Hocker, S. Beismann-Driemeyer, S. Hettwer, A. Lustig, and R. Sterner. Dissection of a $\beta\alpha_8$ barrel enzyme into two folded halves. *Nat Struct Biol*, 8(1):32–36, Jan 2001.

[65] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[66] L. Holm and J. Park. DaliLite workbench for protein structure comparison. *Bioinformatics*, 16(6):566–567, Jun 2000.

[67] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–138, September 1993.

[68] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[69] L. Holm and C. Sander. Decision support system for the evolutionary classification of protein structures. *Proc Int Conf Intell Syst Mol Biol*, 5:140–146, 1997.

[70] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. opt. Soc. Amer*, 4(4):629–642, 1987.

[71] http://whyfiles.org/126dna_forensic/images/dna.gif.

[72] X. Huang. *Bioinformatics – From Genomes to Drugs*, volume 2. Wiley-VCH, 2002.

[73] H. D. Höltje, W. Sippl, D. Rognan, and G. Folkers. *Protein-based Virtual Screening*, chapter 5, pages 145–168. Wiley-VCH Inc., 2 edition, 2003.

[74] A. Jeltsch. Circular permutations in the molecular evolution of DNA methyltransferases. *J Mol Evol*, 49(1):161–164, Jul 1999.

[75] J. Jia, W. Huang, U. Schorken, H. Sahm, G. A. Sprenger, Y. Lindqvist, and G. Schneider. Crystal structure of transaldolase B from Escherichia coli suggests a circular permutation of the $\beta\alpha$ barrel within the class I aldolase family. *Structure*, 4(6):715–724, Jun 1996.

[76] D. S. Johnson and L. A. McGeoch. *The Traveling Salesman Problem: A Case Study in Local Optimization, in: Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons Ltd., 1997.

[77] J. Jung and B. Lee. Protein structure alignment using environmental profiles. *Protein Eng*, 13(8):535–543, Aug 2000.

[78] J. Jung and B. Lee. Circularly permuted proteins in the protein structure database. *Protein Sci*, 10(9):1881–1886, Sep 2001.

[79] W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, Dec 1983.

[80] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[81] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[82] P. Koehl. Protein structure similarities. *Current Opinion in Structural Biology*, 11:348–353, 2001.

[83] R. Koike, K. Kinoshita, and A. Kidera. Probabilistic description of protein alignments for sequences and structures. *Proteins*, 56(1):157–166, Jul 2004.

[84] R. Kolodny and N. Linial. Approximate protein structural alignment in polynomial time. *Proceedings of the National Academy of Sciences USA*, 101(33):12201–12206, Aug 2004.

[85] D. L. Kreher and D. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search.* Boca Raton, FL: CRC Press, 1999.

[86] E. Krissinel and K. Henrick. Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions. *Acta Crystallogr D Biol Crystallogr*, 60(Pt 12 Pt 1):2256–2268, Dec 2004.

[87] E. E. Lattman. Fifth meeting on the critical assessment of techniques for protein structure prediction. *Proteins*, 53(Supplement 6):333–595, 2003.

[88] J. V. Lehtonen, K. Denessiouk, A. C. May, and M. S. Johnson. Finding local structural similarities among families of unrelated protein structures: a generic non-linear alignment algorithm. *Proteins*, 34(3):341–355, Feb 1999.

[89] A. M. Lesk. *Protein Architecture, A practical Approach*. Oxford University Press, 1991.

[90] J. V. Lethonen, K. Denessiouk, A. C. W. May, and M. S. Johnson. Finding local structural similaries among families of unrelated protein structures: a genetic non-linear alignment algorithm. *Proteins: struct. funct. genet.*, 34:341–355, 1999.

[91] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, 21:498–516, 1973.

[92] Y. Lindqvist and G. Schneider. Circular permutations of natural protein sequences: structural evidence. *Curr Opin Struct Biol*, 7(3):422–427, Jun 1997.

[93] G. Lu. Top: a new method for protein structure comparisons and similarity searches. *J. Appl. Cryst*, 33:176–183, 2000.

[94] K. Luger, U. Hommel, M. Herold, J. Hofsteenge, and K. Kirschner. Correct folding of circularly permuted variants of a $\beta\alpha$ barrel enzyme in vivo. *Science*, 243(4888):206–210, Jan 1989.

[95] A. N. Lupas. A circular permutation event in the evolution of the SLH domain? *Mol Microbiol*, 20(4):897–898, May 1996. Letter.

[96] A. N. Lupas, C. P. Ponting, and R. B. Russell. On the evolution of protein folds: are similar motifs in different protein folds the result of convergence, insertion, or relics of an ancient peptide world? *J Struct Biol*, 134(2-3):191–203, May 2001.

[97] E. A. MacGregor, H. M. Jespersen, and B. Svensson. A circularly permuted alpha-amylase-type $\alpha\beta$-barrel structure in glucan-synthesizing glucosyltransferases. *FEBS Lett*, 378(3):263–266, Jan 1996.

[98] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.

[99] G. Meyers. Whole-genome dna sequencing. *Computational Engineering and Science*, 3(1):33–43, 1999.

[100] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

[101] Z. Michalewicz and D. B. Fogel. *How To Solve It: Modern Heuristics*. Springer-Verlag, 2000.

[102] D. W. Mount. *Bioinformatics. Sequence and Genome Analysis*. Cold Spring Harbor, New York, 2 edition, 2004.

[103] I. Muegge and M. Rarey. *Small Molecule Docking and Scoring*, volume 17 of *Reviews in Computational Chemistry*, chapter 1, pages 1–60. Wiley-VCH Inc., 2001.

[104] A. G. Murzin. Sweet-tasting protein monellin is related to the cystatin family of thiol proteinase inhibitors. *J Mol Biol*, 230(2):689–694, Mar 1993.

[105] A. G. Murzin. Probable circular permutation in the flavin-binding domain. *Nat Struct Biol*, 5(2):101, Feb 1998. Comment.

[106] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247:536–540, 1995.

[107] E. A. Nalefski and J. J. Falke. The C2 domain calcium-binding motif: structural and functional diversity. *Protein Sci*, 5(12):2375–2390, Dec 1996.

[108] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, Mar 1970.

[109] R. Nussinov and H. Wolfson. Efficient detection of three-dimensional motifs in biological macromolecules by computer vision techniques. In *Proceedings of the National Academy of Sciences USA*, pages 10495–10499, 1991.

[110] C. A. Orengo, D. T. Jones, and J. M. Thornton. *Bioinformatics: Genes, Proteins & Computers*. BIOS, 2003.

[111] A. R. Ortiz, C. E. Strauss, and O. Olmea. MAMMOTH (matching molecular models obtained from theory): An automated method for model comparison. *Protein Science*, 11:2606–2621, 2002.

[112] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, chapter 19, pages 454–486. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[113] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.

[114] L. Patthy. *Protein Evolution*. Blackwell Science, 1999.

[115] F. Pearl, D. Lee, J. Bray, I. Sillitoe, A. Todd, A. Harrison, J. Thornton, and C. Orengo. Assigning genomic sequences to cath. *Nucleic Acids Research*, 28(1):277–282, 2000.

[116] P. A. Pevzner and H. Tang. Fragment assembly with double-barreled data. *Bioinformatics*, 1(1):1–9, 2001.

[117] G. Polekhina, P. G. Board, R. R. Gali, J. Rossjohn, and M. W. Parker. Molecular basis of glutathione synthetase deficiency and a rare gene permutation event. *EMBO J*, 18(12):3204–3213, Jun 1999.

[118] C. P. Ponting and R. B. Russell. Swaposins: circular permutations within genes encoding saposin homologues. *Trends Biochem Sci*, 20(5):179–180, May 1995. Letter.

[119] C. P. Ponting and R. B. Russell. Identification of distant homologues of fibroblast growth factors suggests a common ancestor for all beta-trefoil proteins. *J Mol Biol*, 302(5):1041–1047, Oct 2000.

[120] K. Price and R. Storn. Differential evolution: A simple evolution strategy for fast optimization. *Dr. Dobb's Journal of Software Tools*, 22(4):18–24, 1997.

[121] Y. Qi and N. V. Grishin. Structural classification of thioredoxin-like fold proteins. *Proteins*, 58(2):376–388, Feb 2005.

[122] I. Rechenberg. *Evolution strategy: Optimization of technical systems by means of biological evolution*. Fromman-Holzboog, 1973.

[123] G. N. J. Reeke and J. W. Becker. Three-dimensional structure of favin: saccharide binding-cyclic permutation in leguminous lectins. *Science*, 234(4780):1108–1111, Nov 1986.

[124] C. R. Reeves and J. E. Beasley. Introduction. In C. R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, page 6. McGraw-Hill, 1995.

[125] E. Ribeiro and C. Ramos. Circular Permutation and Deletion Studies of Myoglobin Indicate that the Correct Position of Its N-Terminus Is Required for Native Stability and Solubility but Not for Native-like Heme Binding and Folding. *Biochemistry*, 4:4699–4709, 2005.

[126] J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer – the arpso. Technical report, EVALife, Dept. of Computer Science, University of Aarhus, 2002.

[127] R. B. Russell and C. P. Ponting. Protein fold irregularities that hinder sequence analysis. *Curr Opin Struct Biol*, 8(3):364–371, Jun 1998.

[128] R. B. Russell, M. A. Saqi, R. A. Sayle, P. A. Bates, and M. J. Sternberg. Recognition of analogous and homologous protein folds: analysis of sequence and structure conservation. *J Mol Biol*, 269(3):423–439, Jun 1997.

[129] T. U. Schwartz, R. Walczak, and G. Blobel. Circular permutation as a tool to reduce surface entropy triggers crystallization of the signal recognition particle receptor beta subunit. *Protein Sci*, 13(10):2814–2818, Oct 2004.

[130] H.-P. Schwefel. *Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie*, volume 26. Birkhäuser Verlag, 1977.

[131] Y. Sergeev and B. Lee. Alignment of β-barrels in $\beta\alpha_8$ proteins using hydrogen-bonding pattern. *J Mol Biol*, 244(2):168–182, Nov 1994.

[132] J. Shapiro and D. Brutlag. FoldMiner: structural motif discovery using an improved superposition algorithm. *Protein Sci*, 13(1):278–294, Jan 2004.

[133] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 69–73. IEEE Press, 1998.

[134] I. Shindyalov and P. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng.*, 11(9):739–747, 1998.

[135] M. L. Sierk and W. R. Pearson. Sensitivity and selectivity in protein structure comparison. *Protein Sci*, 13(3):773–785, 2004.

[136] A. P. Singh and D. L. Brutlag. Protein structure alignment: A comparison of methods.

[137] A. P. Singh and D. L. Brutlag. Hierarchical protein structure superposition using both secondary structure and atomic representations. In *Intl. Conf. on Intelligent Systems in Molecular Biology*, pages 284–293, 1997.

[138] J. Soding and A. N. Lupas. More than the sum of their parts: on the evolution of proteins from peptides. *Bioessays*, 25(9):837–846, Sep 2003.

[139] R. Storn. Designing digital filters with differential evolution. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 109–125. McGraw-Hill, 1999.

[140] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, Berkley, 1995.

[141] R. Storn and K. Price. Differential evolution a simple and efficient heuristic for global optimisation over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[142] S. Subbiah, D. V. Laurents, and M. Levitt. Structural similarity of DNA-binding domains of bacteriophage repressors and the globin core. *Curr. Biol.*, 3(3):141–148, 1993.

[143] J. D. Szustakowski, november 2005. http://zlab.bu.edu/k2sa/index.shtml.

[144] J. D. Szustakowski and Z. Weng. Protein structure alignment using a genetic algorithm. *Proteins: Structure, Function, and Genetics*, 38(4):428–440, 2000.

[145] W. R. Taylor. Random Structural Models for Double Dynamic Programming Score Evaluation. *J Mol Evol*, 44(7):174–180, 1997.

[146] W. R. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Sci*, 8(3):654–665, Mar 1999.

[147] W. R. Taylor. Defining linear segments in protein structure. *J Mol Biol*, 310(5):1135–1150, Jul 2001.

[148] W. R. Taylor and A. Aszodi. *Protein Geometry, Classification, Topology and Symmetry*. Institute of Physics Publishing, 2005.

[149] W. R. Taylor, J. Heringa, F. Baud, and T. P. Flores. A Fourier analysis of symmetry in protein structure. *Protein Eng*, 15(2):79–89, Feb 2002.

[150] W. R. Taylor and C. A. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208:1–22, 1989.

[151] R. Thomsen. Flexible ligand docking using evolutionary algorithms: investigating the effects of variation operators and local search hybrids. *Biosystems*, 72(1-2):57–73, Nov. 2003.

[152] R. Thomsen. *Evolutionary Algorithms and their Application in Bioinformatics*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 2004.

[153] S. Uliel, A. Fliess, and R. Unger. Naturally occurring circular permutations in proteins. *Protein Eng*, 14(8):533–542, Aug 2001.

[154] J. Vesterstrøm. Physical mapping using simulated annealing and evolutionary algorithms. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 327–334, Canberra, 8-12 December 2003. IEEE Press.

[155] J. Vesterstrøm. Evolutionary computation and other heuristics in bioinformatics. Progress report, Department of Computer Science, University of Aarhus, Denmark, 2004.

[156] J. Vesterstrøm. Detection and evaluation of novel CPs using FASE. , 2005. In preparation.

[157] J. Vesterstrøm and J. Riget. Particle swarms: Extensions for improved local, multi-modal, and dynamic search in numerical optimization. Master's thesis, EVALife, Dept. of Computer Science, Aarhus Universitet, 2002.

[158] J. Vesterstrøm and W. R. Taylor. Flexible Secondary Structure Based Protein Structure Comparison Applied to the Detection of Circular Permutation. *Journal of Computational Biology*, 2005. To appear.

[159] J. Vesterstrøm and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1980–1987, Portland, Oregon, 20-23 June 2004. IEEE Press.

[160] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1(4):337–348, 1994.

[161] Y. Wang, R. Prade, J. Griffith, W. Timberlake, and J. Arnold. A Fast Random Cost Algorithm for Physical Mapping. *PNAS*, 91(23):11094–11098, 1994.

[162] J. D. Watson and F. Crick. A structure for deoxyribose nucleic acid. *Nature*, 171:737, April 1953.

[163] E. W. Weisstein. *The CRC concise encyclopedia of mathematics*. CRC Press LLC, 1998. ISBN 0-8493-9640-9.

[164] D. R. Westhead, T. W. Slidel, T. P. Flores, and J. M. Thornton. Protein structural topology: Automated analysis and diagrammatic representation. *Protein Sci*, 8(4):897–904, Apr 1999.

[165] Wikepedia. http://en.wikipedia.org/wiki/Bioinformatics. November 2005.

[166] Wikepedia. http://en.wikipedia.org/wiki/Heuristic_(computer_science). November 2005.

[167] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1(1):67–82, 1997.

[168] T. C. Wood and W. R. Pearson. Evolution of protein sequences and structures. *J Mol Biol*, 291(4):977–995, Aug 1999.

[169] S. Xu, J. Xiao, J. Posfai, R. Maunus, and J. n. Benner. Cloning of the BssHII restriction-modification system in Escherichia coli : BssHII methyltransferase contains circularly permuted cytosine-5 methyltransferase motifs. *Nucleic Acids Res*, 25(20):3991–3994, Oct 1997.

[170] A. S. Yang and B. Honig. An integrated approach to the analysis and modeling of protein sequences and structures. I. Protein structural alignment and a quantitative measure for protein structural distance. *J Mol Biol*, 301(3):665–678, Aug 2000.

[171] A. S. Yang and B. Honig. An integrated approach to the analysis and modeling of protein sequences and structures. II. On the relationship between sequence and structural similarity for proteins that are not obviously related in sequence. *J Mol Biol*, 301(3):679–689, Aug 2000.

[172] A. S. Yang and B. Honig. An integrated approach to the analysis and modeling of protein sequences and structures. III. A comparative study of sequence conservation in protein structural families using multiple structural alignments. *J Mol Biol*, 301(3):691–711, Aug 2000.

[173] X. Yao and Y. Liu. Fast evolutionary programming. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 451–460. MIT Press, 1996.

[174] G. Zweig. An effective tour construction and improvement procedure for the traveling salesman problem. *Operations Research*, 43(6):1049–1057, 1995.

Index