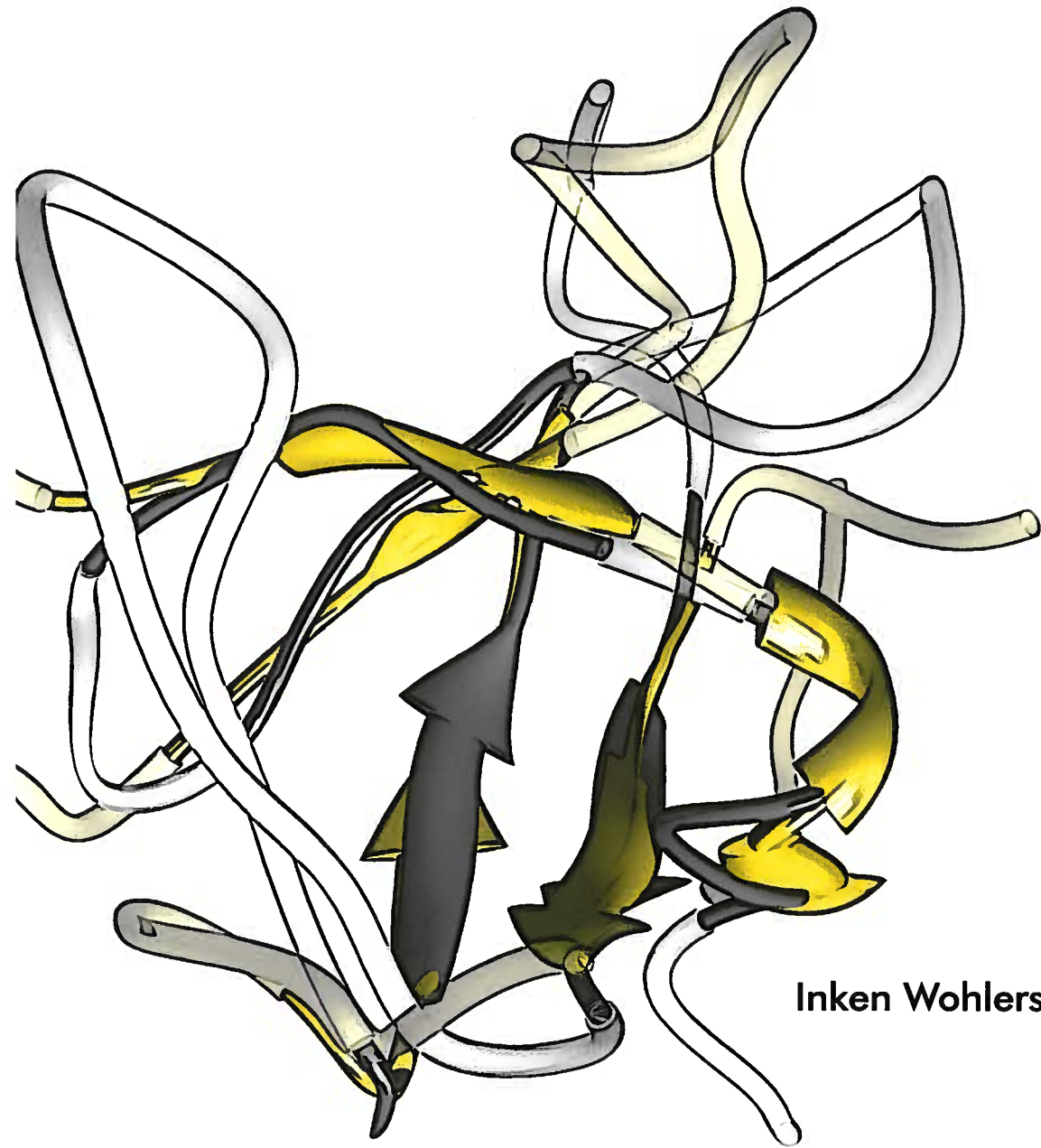


Exact algorithms for pairwise protein structure alignment



Inken Wohlers

Exact algorithms for pairwise protein structure alignment

Inken Wohlers

promotor:

prof.dr. G.W. Klau

leescommissie:

prof.dr. R. Andonov

prof.dr. J. Heringa

prof.dr. K. Reinert

prof.dr. L. Stougie



ISBN: 978-94-6191-495-8



This research has been carried out at the Centrum Wiskunde & Informatica (CWI) in Amsterdam and was partially supported by the Deutsche Forschungsgemeinschaft (DFG).

VRIJE UNIVERSITEIT

Exact algorithms for pairwise protein structure alignment

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Economische Wetenschappen en Bedrijfskunde
op dinsdag 11 december 2012 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Inken Wohlers

geboren te Itzehoe, Duitsland

promotor: prof.dr. G.W. Klau

Contents

1	Introduction	2
1.1	Proteins and protein structures	2
1.2	Protein structure alignment	7
1.2.1	Importance and applications	11
1.2.2	Related work and current state of research	12
1.3	Contributions of this thesis	17
2	Preliminaries from combinatorial optimization	18
2.1	Graph theory	18
2.2	Linear optimization	19
2.3	Integer linear programming	20
2.4	Lagrangian relaxation	21
2.5	Cutting plane method	24
2.6	Branch-and-bound	25
2.7	Dynamic programming	26
3	Mathematical models and structure alignment algorithms	31
3.1	Structure alignment representations	31
3.2	Distance matrix alignment	33
3.3	Graph-based formalizations	35
3.4	Mathematical model	39
3.5	Computing lower and upper bounds	40
3.5.1	Cutting planes	40
3.5.2	Lagrangian relaxation using variable splitting	43
3.5.3	An alternative Lagrangian relaxation	50
3.6	Variable elimination	59
3.7	Branch-and-bound.	61
3.8	Implementations	63
3.8.1	Branch-and-cut	63
3.8.2	PAUL: a Lagrangian approach using variable splitting	65
3.8.3	Alternative Lagrangian approach and branch-and-bound	66

Contents

4 Scoring schemes and quality criteria for structure alignment	68
4.1 Scoring schemes	69
4.1.1 Superposition-based scoring schemes	69
4.1.2 Inter-residue distance-based scoring schemes	71
4.1.3 Dedicated scoring scheme: PAUL	74
4.2 Gold standards	77
4.2.1 Protein classification	77
4.2.2 Reference alignments	78
4.2.3 Data sets	78
4.3 Web server for comparative structural alignment	80
4.3.1 Implementation	83
4.3.2 Case studies	84
5 Computational results	91
5.1 PAUL optimization and evaluation	91
5.1.1 Parameter optimization	92
5.1.2 Evaluating alignment accuracy	98
5.2 Optimal distance matrix alignment using DALI scoring	112
5.2.1 Importance of variable elimination	113
5.2.2 Branch-and-cut and preprocessing	116
5.2.3 DALIX: Optimal distance matrix alignment	118
6 Conclusion	126
6.1 Future work	127
6.2 Concluding remarks	129
Summary	131
Samenvatting	133
Acknowledgments	135
Curriculum vitae	137
Publications	138
Bibliography	139

Come... dry your eyes, for you are life, rarer than a quark and unpredictable beyond the dreams of Heisenberg; the clay in which the forces that shape all things leave their fingerprints most clearly. Dry your eyes... and let's go home.

— From Watchmen (Alan Moore)

Introduction

Proteins are the building blocks of the molecular machinery of life, a complex system that started off about 3.5 billion years ago, approximately 10 billion years after the big bang. The biological systems that we observe today therefore needed a fourth of the time that the universe exists in order to evolve; an evolution that generated an entire tree of life in an incredible process of repeated mutation and selection. A process which finally also led to our existence. And we are now on a quest to decipher this molecular assembly, because it determines to a great part who we are, what we look like and feel, and whether we are healthy. Our ultimate goal is to reverse engineer the molecular machinery [20]: to specify building blocks, detect recurrences, figure out how the components interact within modules and how these modules are joined together.

Similar to complex, man-made machines, the molecular machinery has a limited number of basic elements that are varied and combined: “Nature is a tinkerer and not an inventor” [49]. We observe this for example with the overall structural arrangements of proteins in 3D space, the folds. Although there is a huge number of different protein molecules, there is only an extremely limited number of in the order of thousands of such protein folds [84]. Also the evolutionary units of protein structure, the domains, have repeatedly been duplicated and rearranged to emerge again in different or novel proteins. Hence proteins share a vast amount of structural similarities, like the many different types of screws, threads and wheels in man-made machines.

In order to unravel the molecular machinery, we naturally start by inventorying its building blocks and figuring out how these components relate to each other in a structural, functional and evolutionary context. The way to tackle this is to detect and quantify similarities of protein structures reliably and correctly. This is the scope of this thesis.

1.1 Proteins and protein structures

Proteins are the functional units of any organism. Their biological tasks are very diverse, for instance composing structures, enzymatic action, transportation, information transfer and regulation. They function in concert with other molecules, usually by binding. Structural proteins, for example, assemble together to make up all kinds of

1.1. Proteins and protein structures

Amino acid	3-letter code	1-letter code
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic acid	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

Table 1.1: The 20 proteinogenic amino acids together with their 3- and 1-letter code.

textures, from the cytoskeleton within the cell to tissue, hair and muscles. Enzymatic proteins, moreover, catalyze chemical reactions within the cell, often highly specific. Furthermore, the transportation of ions or molecules can be used for information transfer. Regulatory proteins, eventually, control the transcription of genes.

Proteins consist of chains of amino acid residues. There are 20 proteinogenic amino acids of different chemical properties, see Table 1.1 and Figure 1.1. An *amino acid* has a carboxyl (COOH) and an amino (NH₂) group connected to its central C_α atom, as well as a varying side chain (R). Two amino acids can be combined via a condensation reaction of the carboxyl group of one amino acid with the amino group of the other under release of a water molecule. This results in a peptide bond. The chemical reaction is displayed in Figure 1.2. A protein chain is a polypeptide in which amino acid *residues* are connected by peptide bonds. The end with a free carboxyl group is called the *C-terminus*, and the end with a free amino group the *N-terminus*. Protein chains are synthesized from C- to N-terminus, and we also write their amino acid residues in this order. The chain folds into a 3D structure which allows the protein to carry out its specific biological function. The chemical properties of amino acids (see Figure 1.1) interplay with the protein's 3D structure. For example, hydrophobic residues are usually buried in the core of the structure and hydrophilic residues are usually located at the surface with their side chains pointing towards the solvent.

There are four levels of protein structure organization: primary, secondary, tertiary

Chapter 1. Introduction

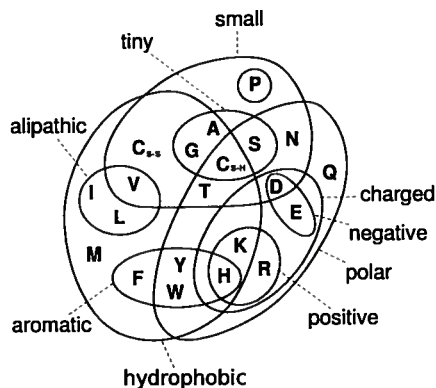


Figure 1.1: The properties of the 20 proteinogenic amino acids in a Venn diagram. Cysteine that is bonded with another cysteine through a disulphide bond (C_{S-S}) has different chemical properties than unbonded cysteine (C_{S-H}). Adapted from Wikipedia.

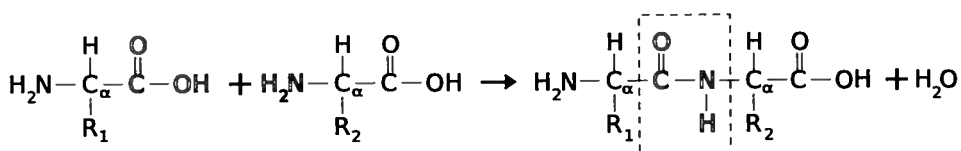


Figure 1.2: The carboxyl group of one amino acid reacts with the amino group of another amino acid and forms under release of water a peptide bond.

and quaternary structure (see Figure 1.3). The *primary structure* is the sequential arrangement of amino acid residues, referred to as *protein sequence*. The 1- or 3-letter codes that are used to denote amino acid residues in the protein sequence are given in Table 1.1. Protein length can vary between a couple of residues up to a few thousands. The largest known protein, titin, even has a size varying from 27000 up to 33000. On average, protein length is about 300 residues [127].

The *secondary structure* comprises regular elements that are stabilized by hydrogen bonds between the carboxyl ($\text{C}=\text{O}$) and amide ($\text{N}-\text{H}$) group of two peptide bonds. The most common secondary structure elements, abbreviated *SSEs*, are α -helices and β -sheets. In an α -helix, there are hydrogen bonds between $\text{C}=\text{O}$ of residue i and $\text{N}-\text{H}$ of residue $i + 4$, resulting in a right-handed spiral shape. β -sheets can be parallel or antiparallel, for a visualization see Figure 1.4. In a parallel β -sheet, the N-termini of the β strands composing the sheet are pointing in the same direction. In an antiparallel β -sheet, N-termini of strands point in opposite direction. A residue in a parallel β -strand forms one hydrogen bond with residue i in the adjacent β -strand and one with residue $i + 2$ in the adjacent strand. A residue in an antiparallel β -strand forms two hydrogen bonds with a residue in the adjacent strand. For this reason, antiparallel β -strands are more stable. Also, antiparallel strands can be adjacent in sequence, because only the chain direction needs to be reversed, whereas parallel strands must

1.1. Proteins and protein structures

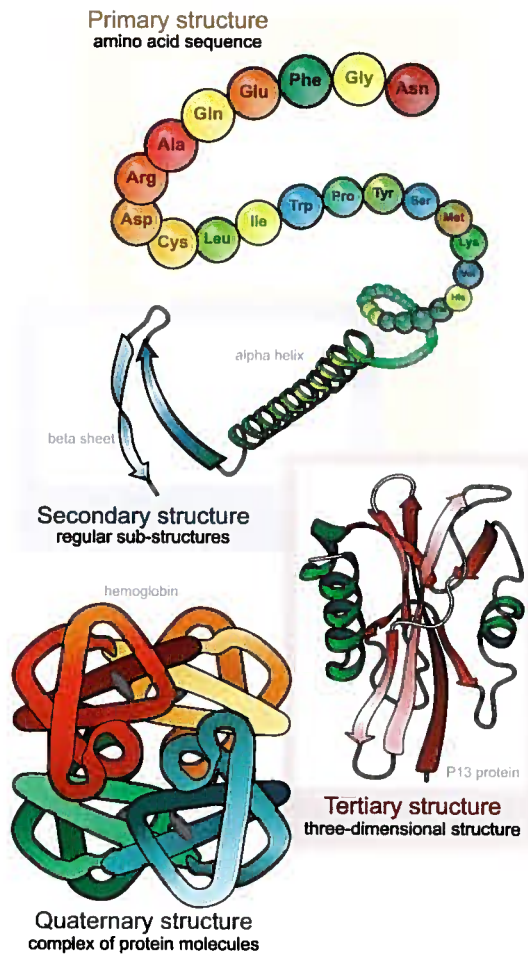


Figure 1.3: The four levels of protein structure. Taken from Wikipedia.

Chapter 1. Introduction

be distant in sequence, because the chain has to return to the C-terminus of the first strand. Rare secondary structures are the 3_{10} helix, in which there are hydrogen bonds between residues i and $i + 3$ and the π helix with hydrogen bonds between residues i and $i + 5$.

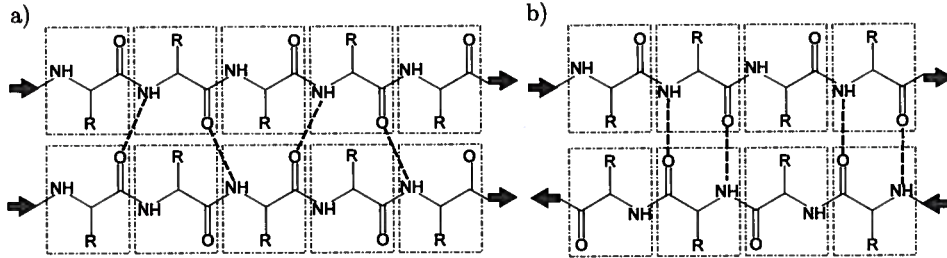


Figure 1.4: Visualization of the hydrogen bonding pattern of a) a parallel β -sheet and b) an antiparallel β -sheet.

The *tertiary structure* is described by the 3D coordinates of all atoms. Finally, a protein can consist of several amino acid chains that compose one complex; this is called the protein's *quaternary structure*.

Besides these four structure levels, there is more terminology to describe (parts of) protein structure. A *motif*, or supersecondary structure, is a certain arrangement of secondary structure elements. An example is a helical bundle, several helices that are located next to each other in parallel or antiparallel fashion. Another important term is a protein domain. A *domain* is a compact substructure of the protein that often folds independently and is autonomously stable. Domains are independent units of function and evolution. They are duplicated, inserted, deleted and mutated. The average size of a domain is about 100 residues [112] and the length of 90% of them does not exceed 200 [48].

Finally, there are various ways of grouping protein structures together based on their similarity according to different levels of hierarchical classification. The corresponding terminology stems from the two major resources for protein organization, SCOP [80] and CATH [83]. Their respective four similarity levels are introduced in the following.

SCOP (Structural Classification Of Proteins) is a manual classification of protein domains into class, fold, superfamily and family. The four main *classes* are (i) all α proteins (only helices), (ii) all β proteins (only sheets), (iii) α/β proteins (mainly parallel β sheets separated by α -helices), (iv) $\alpha + \beta$ proteins (mainly antiparallel β sheets with segregated α and β regions). Proteins with similar arrangement and orientation of secondary structure elements are combined in *folds*. On *superfamily* level, structural similarities indicate an evolutionary relationship. On *family* level, evolutionary relationship is supported by a sequence identity of more than 25%.

CATH also classifies on four levels, class (C), architecture (A), topology (T) and homologous superfamily (H), leading to the acronym. In contrast to SCOP, protein structure assignment to the CATH hierarchy is semi-automatic. CATH uses four classes, (i) mainly α proteins, (ii) mainly β proteins, (iii) mixed α and β and (iv) low secondary structure

content. The *architecture* corresponds to the protein's folds and is assigned manually. The two lowest levels in the hierarchy, *topology* and *homologous superfamily*, correspond to SCOP superfamily and family level, but they are assigned automatically.

SCOP and CATH classification are not the same. First, the domain assignments can differ, such that the entities that are classified are not the same. Further, the classification itself is different. Csaba *et al.* have generated a consensus classification that comprises only domains that have been largely defined the same by SCOP and CATH and that are consistently classified [19].

The most important resource for protein structures is the PDB (Protein Data Bank) [9]. Here, experimentally solved protein structures are uploaded and their atomic coordinates together with various related data are publicly available. The tremendous increase of the number of structures submitted to the PDB illustrates the increasing importance of automatic methods that allow to make the most of this valuable data. As of April 13th 2012, there are 80,710 protein structures in the PDB.

The experimental techniques for solving a protein structure at single atom resolution are x-ray crystallography, nuclear magnetic resonance (NMR) spectroscopy and electron microscopy. X-ray crystallography determines the density of electrons within a crystallized assay of the protein. From the electron density, a model of the positions of the protein's atoms can be deduced. Most structures are solved using this method. Compared to NMR spectroscopy, x-ray crystallography has the disadvantages that it must be possible to crystallize the protein. Further, proteins are flexible objects and the coordinates of a structure solved by x-ray crystallography just provides a snapshot of it. A protein's flexibility or conformational changes can only be captured rudimentarily by repeated crystallization.

In NMR spectroscopy, the protein is placed in a magnetic field and the distinct resonance frequencies of its atomic nuclei are measured. These resonance frequencies depend on the magnetic moment or nuclear spin of the nucleus, which in turn depends on the distribution of electrons nearby. This information is used to construct a model of the three-dimensional structure.

Currently, about 90% of structures have been solved by x-ray crystallography, and about 10% by NMR spectroscopy. Solving a protein structure with either of these experimental methods is a very expensive and time-consuming process.

1.2 Protein structure alignment

A protein *alignment* is a one-to-one mapping of evolutionary related residues in two proteins. These aligned residues are for example homologous, *i.e.*, were also present in a common ancestral protein, or they are analogous, *i.e.*, the result of a convergent evolution which independently resulted into two similar proteins. Residues that are not mapped to a residue in the other protein are assigned to a *gap*, which is denoted by the symbol “-”. Formally, we define an alignment as follows.

Definition 1 (Protein alignment). *In an alignment of two proteins A and B, gaps are inserted at the beginning, into the middle or at the end of the amino acid sequences of*

Chapter 1. Introduction

A and B. The sequences including gaps must have the same length. Residues at the same position in these padded sequences, i.e., written in the same column, are matched or aligned to each other. A residue that is not aligned to a residue in the other sequence is aligned to a gap.

According to this definition, we only consider sequential alignments. In a sequential alignment, aligned residues are *non-crossing*. This means that if residue i is matched with residue k , then residue j and l can be matched if and only if the residue pair (j, l) lies on the right or on the left of the residue pair (i, k) , i.e., either $j > i$ and $l > k$ or $j < i$ and $l < k$. An example alignment is given in Figure 1.5.



```
ASSQACQELCEKDA---KCRFFTLAS--GKCSLFA
TSVDECRKMCEESAVEPSYILQINTETNECYRNN
```

Figure 1.5: An alignment of two proteins. Matched residues are aligned in the same column, unmatched residues are aligned to gaps. In this example, gaps are only inserted in the middle of the first protein. The superposition of the two structures according to the alignment at the bottom is displayed at the top.

The number of possible alignments of two proteins of length n_A and n_B grows exponentially. It is [109]

$$\sum_{k=0}^{\min\{n_A, n_B\}} 2^k \binom{n_A}{k} \binom{n_B}{k}. \quad (1.1)$$

If n_A and n_B are both greater than 106, these are more than 10^{80} different alignments, which is more than there are particles in the universe.

If two proteins have a common ancestor, their alignment should match the evolutionary related, homologous residues. Evolutionary events are *insertions* and *deletions* (also called *indels*) as well as mutations of residues. In *sequence alignment*, these evolutionary events are scored by the use of amino acid substitution scores and gap costs. There is a single score for each pair of amino acids, which reflects the likelihood of one amino acid to be substituted with the other. Identical amino acids and common substitutions have a positive score while rare substitutions have a negative score. A gap in an alignment, i.e., assuming an insertion or deletion during evolution, is penalized. Given amino acid substitution scores and gap penalties, the score-optimal sequence alignment

of two proteins can be computed using dynamic programming based algorithms, *e.g.* Needleman-Wunsch [81] for global and Smith-Waterman [102] for local alignment.

Different gap cost models can be used: linear, concave or affine. In a *linear gap cost model*, each gap is penalized by the same gap cost, irrespective of the number of successive gaps. In a *concave gap cost model*, gap penalties are determined by a concave, *e.g.*, logarithm-like function of the gap length. Finally, in an *affine gap cost model*, there is a *gap open penalty* for starting a new stretch of gaps and a *gap extension penalty* for inserting a gap after another gap. The two latter models reflect that insertion or deletion of consecutive residues during one evolutionary event is more likely than many single indels. Score-optimal pairwise sequence alignment with affine gap costs can be computed using Gotoh's algorithm [30].

The biological function of a protein is determined by its 3D structure. Assuming that function needs to be retained during evolution, protein structure is thus more conserved than protein sequence. Therefore, "comparing protein shapes rather than protein sequences is like using a bigger telescope that looks farther into the universe, and thus farther back in time, opening the door to detecting the most remote and most fascinating evolutionary relations" [43]. For such comparisons of protein shapes, we use structure alignments. A *structure alignment* is an alignment whose residue correspondences are identified based on structural information. It can therefore only be computed for protein pairs for which the 3D structure is known. Protein structure alignment is mainly used in cases in which proteins have evolutionarily diverged such that there is no sufficient signal to compute a good alignment based on their amino acid sequences alone. This is the case if there is a low *sequence similarity*, which can be measured by the percentage of identical matched amino acids in an alignment of two proteins. The region of sequence similarity between 20 and 35% is called the *twilight zone* and the region of sequence similarity below 20% the *midnight zone* [94]. Structure alignment is especially important for protein pairs from these regions.

Together with an alignment, we usually obtain a *similarity score* for a pair of proteins. Based on all-versus-all similarity scores we can get insight into the *protein universe* [43], *i.e.*, the set of all known protein structures, and its organization. Figure 1.6 displays a recent visualization of the protein universe based on sequence comparison. Figure 1.7 shows such a visualization based on structure comparison, together with cartoons of a few of the most common protein folds.

The precise structural information that is used for a structure alignment depends on the algorithm that is used and on its focus. Following Hasegawa and Holm [36], we classify scoring schemes for structure alignment into 3-, 2-, 1- and 0-dimensional, depending on the dimensionality of the underlying protein representation.

3-dimensional scoring schemes. The 3-dimensional scoring schemes are based on *rigid body superposition* of the two protein structures. Given a set of reference 3D points of the equivalenced residues, a superposition of minimum *coordinate root mean square deviation* (RMSD_c) is sought. An example of such a superposition is given in

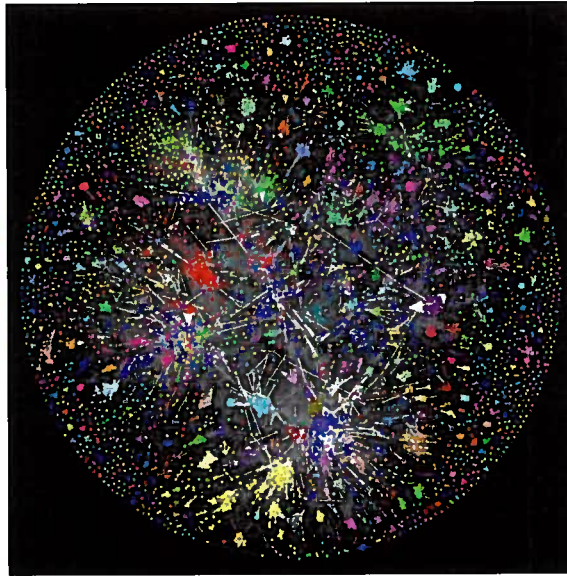


Figure 1.6: Visualization of protein similarities based on sequence comparison; each node represents a protein. Coloring according to protein fold. Figure taken from [2].

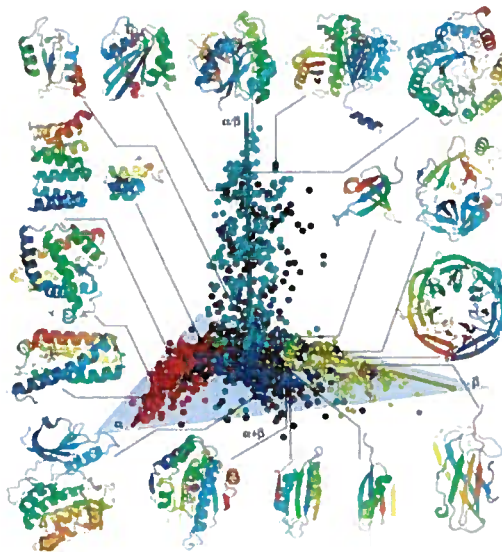


Figure 1.7: Visualization of protein similarities based on structure comparison. The axes denote the SCOP classes all α , all β and α/β structures. $\alpha+\beta$ structures are located between all α and all β structures. Also a few examples of protein folds together with their location in protein structure space are displayed. Figure taken from [45]

Figure 1.5. RMSD_c is a popular measure for structural similarity. It is defined as

$$\text{RMSD}_c = \sqrt{\frac{1}{n} \sum_{i,k \text{ aligned}} (d_{i,T(k)})^2}, \quad (1.2)$$

where $T(k)$ are the 3D coordinates of residue k after the second structure has been superpositioned on the first such that RMSD_c is minimized. The distance $d_{i,T(k)}$ is the distance between the two aligned residues after this superpositioning. The number of aligned residues is denoted by n . Given a set of residue correspondences, the superposition of minimum RMSD_c can be obtained by Kabsch' algorithm [54]. It computes a 3-dimensional translation vector and a 3×3 rotation matrix by which one structure is then translated and rotated. RMSD_c is length-dependent: the more residues are aligned, the larger it typically gets. Algorithms that use 3-dimensional scoring schemes are thus multiobjective. They minimize RMSD_c while maximizing alignment length by using scoring functions that balance these two criteria. Finding the score-optimal residue correspondences is believed to be an *NP*-hard problem for almost all 3-dimensional scoring functions.

2-dimensional scoring schemes. The 2-dimensional scoring functions integrate structural information by assigning it to pairs of residues. The most straightforward way is the use of inter-residue distances, the distances between any two residues in a protein. They can be formalized in a protein structure distance matrix. Similar structural regions are then detected by identifying similar inter-residue distance patterns. Other 2-dimensional structure information is for example the use of contacts or of 3D vectors between residues. Structure alignment using 2-dimensional scoring schemes is also in most relevant cases either proven or believed to be an *NP*-hard problem [64].

1-dimensional scoring schemes. 1-dimensional scoring schemes assign structural features to single residues, for examples secondary structure, polarity, solvent accessibility or backbone angles with neighboring residues. In this way, fast, polynomial-time sequence alignment algorithms can be used also for structure alignment. The disadvantage is that approaches which use 1-dimensional scoring schemes lose a substantial amount of the original structural information such that the corresponding alignments are less accurate.

0-dimensional scoring schemes. Finally, 0-dimensional scoring schemes represent the structure only by a fingerprint. This is achieved in such a way that two similar structures also have a similar encoding. Such a compression allows to index databases and therefore for the fastest way of identifying similar structures. Approaches that use 0-dimensional scoring schemes do not use or return an alignment.

1.2.1 Importance and applications

There are two main classes of applications for protein structure alignment. For one, the alignment is of interest and for the other the obtained similarity score. In the

Chapter 1. Introduction

first case, the precise residue correspondences in the two proteins are used. Given such a set of structurally equivalent residues, the information of a well-known protein can be transferred to a still unknown protein. In the case of a significant similarity that indicates homology or analogy, we can hypothesize that both proteins share the same function. Then, if the function of one protein is known and of the other one not, we can transfer this hypothetical function. Furthermore, we can use structure alignments for detecting functional motifs that are structurally conserved. Finally, structure alignments can also serve as gold-standard for sequence alignment algorithms.

The second class of applications for structure alignment is the organization of the protein universe. With the tremendous increase of experimentally solved structures, a manual classification into protein families, superfamilies and folds is not feasible any more and automatic methods are needed. With their help, proteins can be clustered with respect to all pairwise similarity scores. In order to demonstrate their applicability for this task, structure alignment algorithms and scoring schemes are often benchmarked based on their ability to reproduce manual classifications like **SCOP** [80].

Closely related and from the second class of applications is the problem of returning from all known protein structures those with significant similarity to a new protein structure that has just been experimentally solved. A special case is the family identification problem [70] in which a new protein structure is supposed to be added to an existing classification by detecting the target structure with largest similarity score. The new structure is then assigned to the protein family of this target structure.

These applications of structure alignment help us to learn about the relationships between structure, sequence and function and about protein evolution. They highlight and quantify structural similarities and put proteins into an evolutionary relationship within a protein universe.

1.2.2 Related work and current state of research

There are many structure alignment algorithms, each optimizing its own scoring scheme. The Wikipedia page on structure alignment software¹, for example, currently lists 95 different programs. In the following we consider approaches for 2- and 3-dimensional scoring schemes, focusing on the methods for 2-dimensional scoring, because our algorithms fall into this category.

The algorithmic approaches are manifold: iterative or double dynamic programming, superpositioning, geometric hashing, chaining fragments of rigidly superposable residues, matching distance matrices, *etc.* For comprehensive reviews see for example [25, 100]. The scoring schemes that alignment methods use are as diverse as the approaches themselves [36].

Finding an alignment that is score-optimal is assumed to be an *NP*-hard problem for nearly all 2- and 3-dimensional scoring schemes. Despite this, almost all structure alignment methods use such scoring schemes and implement sophisticated, powerful and experimentally well-tested heuristics. Nonetheless, these heuristic algorithms can

¹http://en.wikipedia.org/wiki/Structural_alignment_software

not provide any quality guarantee on the returned alignment; its score can be arbitrarily far from the optimal score. Therefore, scoring schemes can not be rigorously compared and the performance of the heuristic algorithms with respect to finding a score-optimal alignment can not be assessed. As a result, there is to date no consensus on which scoring scheme best reflects and discriminates structural similarity. The situation is further complicated by a variety of qualitatively different structural similarities and by the different focus of scoring schemes and algorithms. For example, some approaches are local, others global, some suppose that substructures must be rigidly superposable, others allow for large flexibility, some are order-preserving, others not, *etc.* Sierk and Kleywegt [100] comprehensively describe the process of choosing an appropriate structure alignment method based on clarifying which biological question the user wants to answer.

To date there has been almost no work on *practical* exact algorithms for structure alignment using 2- or 3-dimensional scoring schemes. With the practicability of an exact algorithm we denote that it can compute optimal structure alignments within time spans of minutes to maximum hours on a PC for alignment instances that are interesting for the biologist. Currently, the only practical exact algorithms for structure alignment find alignments (i) with a maximum number of common contacts, *i.e.*, of maximum contact map overlap (CMO) [4, 14, 104, 122], (ii) with a maximum number of aligned residues with all distance differences less than a threshold [69] or (iii) with a maximum score of aligned inter-residue distances [116, 113, 114]. The latter ones have been developed during the work performed for this thesis.

Nonetheless, exact and approximate algorithms for the structure alignment problem were and still are extensively studied from a theoretical point of view [64, 3, 29, 62, 123, 89, 91, 90, 68]. Many of them have been designed to establish complexity results for the structure alignment problem for particular scoring functions and under certain, often scholarly constraints. For 2-dimensional scoring schemes to the best of our knowledge no *NP*-hardness has been proven; the only exception is CMO. For CMO, the problem is not only *NP*-hard, but also *NP*-hard to approximate up to a factor $(1 - \epsilon)$, where $\epsilon > 0$ is an arbitrary, previously given error [29]. This means that no polynomial-time approximation scheme (PTAS) exists for CMO. Nonetheless, there are PTAS for special variants of the problem, for example if residues are additionally restricted to be closer than a given threshold within a superposition in 3D space [68]. Taking into consideration that CMO as a special case already renders the alignment problem *NP*-hard, we assume that the structure alignment problem for most other biologically relevant 2-dimensional scoring schemes is *NP*-hard as well. Similar, for most 3-dimensional scoring schemes no *NP*-hardness has been proven, but is assumed. For two important types of 3-dimensional scoring functions, rather recently polynomial-time approximation schemes have been proposed. The first one is for certain scorings based on the distance between aligned residues [62] and the second one for scorings based on maximizing the number of residues that can be superposed under a predefined distance threshold [89]. These algorithms guarantee that the solution's score is at most a previously given additive error $\epsilon > 0$ less than the optimal score. For the second scoring function even a polynomial-time exact algorithm is given. Still, none of these approaches is practical.

Chapter 1. Introduction

In the following, we briefly present several structure alignment algorithms. The first six compute alignments by finding residue correspondences that induce a good matching of corresponding inter-residue distances. They thus aim to optimize a 2-dimensional scoring scheme. We include these approaches to give an extensive overview of the algorithms that use 2-dimensional scoring schemes. In the following we then summarize many algorithms that do not or at least not exclusively use 2-dimensional scoring. We tried to select those methods that could be considered state-of-the-art or that frequently appear in the literature and are widely used. This selection is neither comprehensive nor meant to reflect the quality of the corresponding programs. We start by summarizing algorithms that use iterative superpositioning and alignment, aiming to maximize a 3-dimensional scoring scheme. They are followed by methods that chain pairs of structurally similar fragments. These use various different scoring functions, 2- and 3-dimensional, to evaluate similarity. Finally, we summarize approaches that aim to compute a maximum number of aligned residues that can be superposed under a pre-defined threshold, which constitutes a 3-dimensional scoring scheme.

CMO. There are various algorithms that compute the alignment of maximum contact map overlap (CMO), exact or heuristic. Implementations of exact approaches are for example A_PURVA [4], CMOS [122] and the approach of Caprara *et al.* [14] and of heuristics for example MSVNS [85], AI-EIGEN [21], SADP [51] and BIMAL [52]. A contact is an inter-residue distance of less than a given distance threshold, usually 7.5 Å. The contact map overlap, introduced in [28], is given by the number of common contacts induced by an alignment, *i.e.*, the number of contacts in the two proteins whose endpoints are aligned. Exact approaches for CMO are based on graph formulations of the problem and use techniques from combinatorial optimization. The algorithms that are devised in this thesis are related to existing exact algorithms for contact map overlap [14, 4].

DAST. DAST (Distance-based Alignment Search Tool) [69] is an algorithm that searches for the maximum number of aligned residues for which the difference between any pair of aligned distances is less than a threshold (usually 3 Å). The objective is to obtain an alignment with overall small *distance root mean square deviation*, which is defined as

$$\text{RMSD}_d = \sqrt{\frac{1}{m} \sum_{\substack{i,k \text{ aligned,} \\ j,l \text{ aligned,} \\ j>i, k>l}} (d_{ij} - d_{kl})^2}. \quad (1.3)$$

Here, m is the number of aligned inter-residue distances. In a DAST alignment, any distance between residues i and j of protein A differs therefore no more than the given threshold from the aligned distance between residues k and l of protein B . Note that, compared to coordinate RMSD according to Equation (1.2), distance RMSD considers inter-residue distances and does not involve superpositioning. Since in Equation (1.3) the sum of all distance differences is divided by their number, the overall RMSD_d of a DAST alignment is less than the previously specified threshold. Because of this condition, DAST alignments are comparably local alignments. The exact DAST algorithm searches for largest cliques in a product- or so-called alignment graph. We introduce the alignment graph in Chapter 3.

DALI. DALI (Distance mAtrix aLIgnment) [41] finds alignments with a high DALI score. This score evaluates how well the inter-residue distances induced by an alignment match. If they are similar, the score is large and if they are dissimilar, it is low or negative. The DALI score and its corresponding z-score are introduced in detail in Chapter 4. The DALI algorithm finds pairs of fragments of six residues with high DALI score and then assembles them using a Monte-Carlo method.

SSAP. SSAP (Sequential Structure Alignment Program) [108] is one of the first structure alignment methods, but its current version is still used today for automatic classification in CATH [83]. SSAP uses 3-dimensional inter-residue vectors instead of inter-residue distances. The difference between two such vectors is computed, and of this difference vector the length is taken. It is small if two inter-residue vectors have similar length and orientation and increases the more the vectors differ. The SSAP scoring function assigns large values to small difference vectors and small values to large difference vectors. The algorithm aims to detect the alignment with maximum overall score, taking into consideration also gap costs and a negative offset for aligning two residues. SSAP uses double dynamic programming. This technique keeps a pair of residues aligned and computes its score contribution by dynamic programming. This is done for all residue pairs, representing respective lower dynamic programming levels. Subsequently using a second, higher dynamic programming level, the non-crossing residue pairs with largest overall score contribution are chosen. Double dynamic programming is also used extensively for the exact algorithms that are devised in this thesis. It is described in detail in Chapter 2.

MATRAS. MATRAS (MArkovian TRAnSition of protein Structure) [58] uses three different scores, one SSE-based, another one environment-based and a third one distance-based. All three are log-odds scores: The score for pairing two elements (SSEs, environments or inter-residue distances, respectively) is estimated by the log-likelihood by which these elements are paired in gold-standard alignments. As gold-standard, protein pairs were used which are, according to sequence similarity, clearly homologous. The algorithm aligns the structures hierarchically, starting with the SSEs. Aligned SSEs are promoted by a constant value and an alignment using the 1-dimensional environment scores is computed. Then, the score of each residue pair given this initial alignment is evaluated using the distance-based score. Via dynamic programming, the alignment of maximum overall distance-based score contribution is computed. The resulting alignment is then again used for the evaluation of score contribution. This iterative dynamic programming is performed until residue correspondences do not change any more, but at most a specific number of times.

VOROLIGN. VOROLIGN [12] is a double dynamic programming-based algorithm that uses Voronoi tessellation for determining residues that are in contact. This is similar to determining close residues using Euclidean distance, as in CMO, but takes into consideration additional structural information. The contacting residues are considered in a lower level dynamic program. They are aligned using an SSE- and a structure-based amino acid substitution matrix as well as penalties for unaligned contacts. The overall

Chapter 1. Introduction

lower level dynamic programming scores for each residue pair are then used as upper level dynamic programming scores for the computation of the structure alignment.

TM-ALIGN, STRUCTAL, SSM. TM-ALIGN [129] is an algorithm that finds an alignment based on superpositioning, *i.e.*, 3-dimensional scoring. Its name traces back to the template modeling score (TM-score) [128] which TM-ALIGN aims to maximize. The algorithm finds in a first step three types of initial alignments, based on matching SSEs, based on TM-score of gapless alignment and based on a combination of both. These alignments are then refined in a procedure of iterative dynamic programming and superpositioning. Given the initial residue correspondences, the superposition that maximizes the TM-score is computed by finding the corresponding TM-score rotation matrix [128]. Then, given this superposition, a score is assigned to each pair of residue i from the first structure and k from the second structure, reflecting how well they match given the current superposition. This score is maximal if their representative atoms have the same 3D position. These scores are used for a sequence alignment, in which the residue correspondences may change. This new assignment is then again used for superpositioning and so on, iteratively adjusting the alignment until residue correspondences do not change any more. STRUCTAL [105] is another algorithm based on iterative superpositioning and aligning. It uses five different initial alignments: aligning C-termini, N-termini and chain mid points without gaps as well as an alignment maximizing sequence identity and another one maximizing the similarity of C_α torsion angles. After optimal superposition given an alignment, dynamic programming is used to find new residue correspondences which maximize the STRUCTAL score. These two steps are iteratively repeated. SSM (Secondary Structure Matching) [63] also uses iterative superpositioning and adapting of alignment correspondences. It initially matches secondary structures. Given the corresponding superposition of SSEs, it applies various rules to obtain a residue alignment. This residue alignment is then iteratively superposed and recomputed in order to aim maximizing a score that normalizes $RMSD_c$ against alignment length.

CE, FATCAT, MATT, PPM. First, these algorithms evaluate how well pairs of fragments of the two structures match. Then they chain these so-called aligned fragment pairs together in a structurally consistent way. CE (Combinatorial Extension) [99] is, together with FATCAT, the structure alignment algorithm provided by the PDB. It computes fragment similarity based on the differences of inter-residue distances. In a final step it applies iterative superposition and alignment to refine the structure alignment based on $RMSD_c$. FATCAT (Flexible structure Alignment by Chaining Aligned fragment pairs allowing Twists) permits to introduce a given number of hinges in the alignment and is therefore especially tailored for comparing flexible proteins. Same as CE, it is available via the PDB. MATT (Multiple Alignment with Translations and Twists) is another algorithm for flexible alignment. Different from FATCAT it does not allow only for a certain, limited number of hinges. Instead it allows to chain fragments even if they do not match well when superposed, but then corrects for such inconsistencies later during alignment computation. In PPM (Phenotypic Plasticity Method), aligned fragment pairs exceed an $RMSD_c$ -based score threshold. Then, any two pairs

of fragment pairs have a certain score assigned to them which reflects the cost of morphing into each other. This again RMSD_c-based score decides which fragment pairs will be chained together in the final structure alignment.

LGA, MAX-PAIRS. LGA (Local-Global Alignment) [125] is the structure alignment algorithm used to benchmark structure models in the bi-annual CASP competition for protein structure prediction [78]. It aims to maximize the number of aligned residues which lie below a given threshold from each other in a corresponding superposition. This is done by iterative 3D transformation and exclusion of residues. MAX-PAIRS is a heuristic algorithm for the same scoring scheme, following the corresponding approach for finding an optimal alignment [89]. It experimentally improves over LGA.

1.3 Contributions of this thesis

This thesis deals with practical algorithms for score-optimal protein structure alignment using 2-dimensional scoring schemes. We were inspired by the exact algorithms for contact map overlap: From them we observed that, although using algorithms with exponential worst case run time, computing CMO alignments to optimality is often feasible. This is especially the case for real-life instances in which usually structurally similar proteins are compared. Building upon the CMO ideas, the scope of this thesis is to develop algorithms which can compute score-optimal solutions for any 2-dimensional scoring function for many biologically relevant protein pairs.

To this end, we formulate for the first time general mathematical models for any 2-dimensional scoring function and show how specific, widely used scoring schemes are special cases of them. Further, we extend existing and design novel exact algorithms to compute provably optimal solutions in these models for many real-life problem instances. We investigate the practicability and performance of our algorithms for different scoring schemes and analyze what influences this performance.

Besides developing and advancing algorithms, another focus of this thesis is the application of our algorithms. For this purpose, we develop our own scoring scheme that integrates various biological knowledge and is specifically amenable to produce score-optimal alignments. We implemented and released it to the public within a structure alignment tool called PAUL. Furthermore, existing scoring functions like the ones of CMO, DALI and MATRAS are integrated into our algorithms. We provide their computation within a web server, together with a comprehensive comparison of the generated alignments and their scores. For the DALI scoring function, we benchmark on a large scale heuristic alignments generated by the popular DALI program.

Our exact algorithms that are practical for biologically interesting protein pairs are a first and important step towards the goal of comparing and ultimately improving scoring functions and heuristic algorithms for protein structure alignment. Rigorously enhanced structure alignments, in turn, detect more accurately the structural, functional and evolutionary relationships between proteins, which eventually helps to unravel the molecular machinery of life. This thesis contributes towards this goal.

Preliminaries from combinatorial optimization

In this chapter we give a brief introduction to graph theory, linear programming and combinatorial optimization. Its focus lies on mathematical terms and concepts from these fields which are used later in this thesis. For comprehensive introductions see for example text books such as [22, 11, 32].

2.1 Graph theory

A directed graph is a pair $G = (V, E)$ where *nodes*, or also called vertices, V are a finite set and *edges* $E \subseteq V \times V$. By $|V|$ and $|E|$, we denote the number of nodes and edges, respectively. An edge from node v to node w is denoted by (v, w) . Its source node is called its *tail*, and its target node is called its *head*. We will only consider directed graphs in this thesis and usually just call them graphs, omitting the word directed.

A *subgraph* $G' = (V', E')$ of $G = (V, E)$ is a graph whose nodes are a subset of V and whose edges are incident to nodes of V' and a subset of E , *i.e.*, $V' \subseteq V$ and $E' \subseteq E \cap V' \times V'$. If G' is a subgraph of G , then G is called the *supergraph* of G' . An *induced subgraph* G' consists of a subset of nodes from graph G together with all edges between pairs of these nodes that are present in the original graph G , *i.e.*, $E' = E \cap V' \times V'$.

We can further specify paths in graphs. A *path* is a sequence of nodes with an edge between any two consecutive nodes. A *cycle* is a path whose starting node and ending node are identical. Graphs that contain cycles are called *cyclic* and graphs without cycles *acyclic*. Paths may have also many other properties and there are various problem formulations and algorithms for finding such specific paths. For example, the *shortest path* problem seeks the path of minimum overall edge weight and, in the case of positive weights, can be solved using Dijkstra's algorithm [23]. In directed, acyclic graphs, with which we deal in this thesis, shortest or longest paths can be computed by traversing all nodes in topological order. Such a *topological order* is an order of nodes such that for every edge (v, w) , node v comes before node w .

We will also use a *product graph* of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a graph in which the nodes are defined by the Cartesian product $V_1 \times V_2$. We denote such a node in the product graph by $v_1.v_2$. The edges in this graph are defined based on E_1 and E_2 : There is an edge $(v_1.v_2, w_1.w_2)$ in the product graph if and only if there is an edge (v_1, w_1) in graph G_1 and an edge (v_2, w_2) in graph G_2 .

2.2 Linear optimization

In optimization, we are concerned with the problem of finding the best element of a given set. Therefore we measure the quality of an element by assigning an *objective value* to it. Such an assignment is called an *objective function*. In the following, we consider the case that a higher objective value is better. The optimization problem then asks to maximize the objective function, which is defined over the set of *feasible solutions*. This feasible set is given by *constraints* that the elements have to satisfy. An element that is best, *i.e.*, has maximum objective value, is called an *optimal solution*.

We focus here on optimization problems in which the objective as well as all constraints are described by linear functions. Such a linear programming problem, or *linear program* (LP), is given by

$$\max c^T x \tag{2.1}$$

$$\text{s.t. } Ax \leq b, \tag{2.2}$$

in which the vector $c \in \mathbb{R}^n$ gives the objective function. The feasible region is a polyhedron $\{x \in \mathbb{R}^n \mid Ax \leq b\}$, in which $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Such a polyhedron is given by constraints $a_i x \leq b_i$ for $i = 1, \dots, m$. A linear equality $a_i x = b_i$ describes a *hyperplane* $\{x \in \mathbb{R}^n \mid a_i x = b_i\}$ and a linear inequality $a_i x \leq b_i$ a *halfspace* $\{x \in \mathbb{R}^n \mid a_i x \leq b_i\}$. A polyhedron is thus the intersection of a finite number of halfspaces and convex. We denote an optimal solution by x^* , it is a feasible solution with maximal objective function value, *i.e.*, $c^T x^* \geq c^T x$ for all feasible x . An intersection of the polyhedron and n hyperplanes $a_i x = b_i$ is called a *basic feasible solution*. It represents a *vertex* of the polyhedron. For a visualization, see Figure 2.1, left. An optimal solution, if it exists, is one of the basic feasible solutions and thus attained at a vertex of the polyhedron.

To each *primal* LP there corresponds another LP, called the dual. A *dual* problem has a constraint for each primal variable and a variable for each primal constraint. Intuitively, a dual variable assigns a penalty to the violation of primal constraints. In an optimal solution of the dual problem, these penalty values are set such that the constraints that they correspond to in the primal problem are satisfied. If the primal is a maximization problem, the dual is a minimization problem and the other way round. The dual of LP (2.1)-(2.2), for example, is defined as the minimization problem

$$\min y^T b \tag{2.3}$$

$$\text{s.t. } y^T A = c^T \tag{2.4}$$

$$y \geq 0, \tag{2.5}$$

Chapter 2. Preliminaries from combinatorial optimization

where $y \in \mathbb{R}^m$. The dual of a dual problem is again the primal problem. There are important weak and a strong duality theorems which both hold for linear programs. The *weak duality theorem* states that the objective function value of any feasible solution x of the primal is less than or equal to the objective function value of any feasible solution y of the dual, *i.e.*,

$$c^T x \leq y^T b. \quad (2.6)$$

If the primal and consequently also the dual problem have finite optimal solutions x^* and y^* , respectively, then the objective function values of these optimal solutions are equal, *i.e.*,

$$c^T x^* = y^{*T} b. \quad (2.7)$$

This relationship is called the *strong duality theorem*. We will encounter linear programming duality again in the context of Lagrangian relaxation.

Linear programs can be solved in polynomial time using ellipsoid [59] or interior point methods [56]. However, the simplex method (published 1947 by G. Dantzig) with an exponential worst case run time is usually used and performs well in practice. It proceeds from one vertex in the polyhedron to the next. Starting from a basic feasible solution, a direction is determined in which the objective function increases and the algorithm progresses to the next basic feasible solution which lies in this direction.

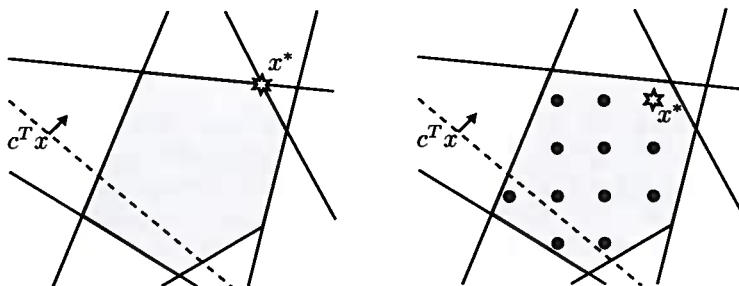


Figure 2.1: In gray, a polyhedron that is described by six constraints $a_i x \leq b_i$. The objective function $c^T x$ increases in the direction in which the arrow points. The optimal solution x^* is denoted by a star. Left: The linear program. Right: The integer linear program. Here, only the integer points within the polyhedron are feasible, which are colored black. The LP relaxation of this ILP is the LP problem that is visualized on the left side. The optimal objective function value of the LP relaxation is always an upper bound on the optimal objective function value of the ILP.

2.3 Integer linear programming

An *integer linear program* (ILP) is a linear program in which variables are restricted to have integer value. LPs with this additional constraint are *NP-hard*. Many discrete optimization problems can be cast into ILPs, for example problems from combinatorial

optimization. In combinatorial optimization, we search for an optimal object in a finite set of objects [97]. Many such problems can be cast in the following framework. Given is a basic set together with a value for each of its elements, the feasible solutions are a subset of the basic set's power set. The objective value of a feasible solution is the sum of the values of the basic elements that the solution set contains.

In an ILP for a problem from combinatorial optimization, elements are represented by binary variables which indicate whether they are in the solution. If a variable has value 1, the corresponding element is in the solution, and if it has value 0, it is not. Many problems from graph theory have respective ILP formulations, for example the traveling salesman problem, which asks for the shortest cyclic tour passing through a given set of cities.

The *LP relaxation* of an ILP is the problem in which the constraint that variables have to take integer values is relaxed: variable values may then be real numbers. Since the polyhedron described by linear constraints contains all feasible integer solutions, the optimal objective function value of an LP relaxation is an upper bound on the optimal objective function value of the ILP. For a visualization, see Figure 2.1. In the rare case that the optimal LP solution has integer values, it is also the optimal solution of the ILP.

Algorithms for solving ILPs associated with *NP*-hard problems are either exact, approximate or heuristic. *Exact algorithms* compute the optimal solution, but they have an exponential worst-case run time. They often use the divide-and-conquer paradigm: dividing the problem into smaller subproblems which can eventually be solved and combining these solutions to find the optimal solution of the original problem.

Approximation algorithms approximate the optimal solution. They are usually polynomial-time algorithms that return a solution that is at most a constant factor or a previously defined $\epsilon > 0$ worse than the optimal solution. Not all *NP*-hard problems can be approximated and approximation algorithms are not necessarily feasible in practice.

Heuristic algorithms return a solution without any theoretical quality guarantee, whose objective value can thus be arbitrarily far from the objective value of the optimal solution. They are typically fast and designed to produce solutions that, even if non-optimal, are sufficient for practical applications. Furthermore, heuristics are important to generate feasible solutions and lower bounds that can be used within exact algorithms.

2.4 Lagrangian relaxation

Lagrangian relaxation is, like LP relaxation, a technique to obtain upper bounds for an ILP. The idea behind it is based on the observation that some constraints can render a problem *NP*-hard, but that when removing these constraints, the relaxed problem can

Chapter 2. Preliminaries from combinatorial optimization

be solved efficiently, *i.e.*, in polynomial time. We consider the ILP

$$\max c^T x \tag{2.8}$$

$$\text{s.t. } Ax \leq b \tag{2.9}$$

$$Dx \leq p \tag{2.10}$$

$$x \text{ integer,} \tag{2.11}$$

in which $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, $D \in \mathbb{R}^{l \times n}$ and $p \in \mathbb{R}^l$. Constraints $Dx \leq p$ are the “bad” constraints, with which the problem can not be solved efficiently. In Lagrangian relaxation, such “bad” constraints are relaxed and moved to the objective function where they are multiplied by so-called *Lagrangian multipliers*. The Lagrangian relaxation is then

$$\text{LR}(\lambda) = \max_x c^T x + \lambda^T (p - Dx) \tag{2.12}$$

$$\text{s.t. } Ax \leq b \tag{2.13}$$

$$x \text{ integer.} \tag{2.14}$$

Here, $\lambda \geq 0$ is an l -dimensional vector with one Lagrangian multiplier for each relaxed constraint. If such a relaxed constraint is violated, $p_i - d_i x$ is less than zero and subtraction of the corresponding Lagrangian multiplier λ_i penalizes the violation. Since the solution space becomes larger when relaxing constraints, while still containing all solutions to the original problem, the optimal solution of the relaxed problem will constitute an upper bound to the original problem for any $\lambda \geq 0$.

Not only inequality constraints, also equality constraints can be relaxed. This is for example done in a technique called *variable splitting*. In this approach, not one type of constraint is relaxed, but instead each variable x is split into two variables y and z , with one variable present in one set of constraints and the other one in another set of constraints. Additional equality constraints $y = z$ ensure that after such a variable splitting both variables have the same value, resulting in the ILP

$$\max c^T y \tag{2.15}$$

$$\text{s.t. } Ay \leq b \tag{2.16}$$

$$Dz \leq p \tag{2.17}$$

$$y = z \tag{2.18}$$

$$y, z \text{ integer.} \tag{2.19}$$

The equality constraints (2.18) are then relaxed as follows

$$\text{LR}(\lambda) = \max_{y,z} c^T y + \lambda^T (y - z) \tag{2.20}$$

$$\text{s.t. } Ay \leq b \tag{2.21}$$

$$Dz \leq p \tag{2.22}$$

$$y, z \text{ integer.} \tag{2.23}$$

When relaxing equality constraints, multipliers λ are in \mathbb{R} . We can re-write the objective function as $(c^T + \lambda^T)y - \lambda^T z$ and observe that the problem decomposes into two

Algorithm 1 Solving the Lagrangian dual problem (2.24)–(2.26).

```

1:  $t \leftarrow 0$  // Iteration
2:  $UB \leftarrow \infty$  // Best upper bound
3:  $UB^t$  // Upper bound of iteration  $t$ 
4:  $\gamma^t \in \mathbb{R}_+$  // Stepwidth
5:  $g^t \in \mathbb{R}^n$ ,  $g^t \leftarrow \text{undef}$  // Subgradient
6:  $\lambda^t \in \mathbb{R}_+^n$ ,  $\lambda^t \leftarrow 0$  // Lagrangian multipliers
7: while  $g^t \neq 0$  do
8:   Compute objective function value  $\text{LR}(\lambda^t)$  of relaxation (2.12)–(2.14)
9:    $UB^t \leftarrow \text{LR}(\lambda^t)$ 
10:  if  $UB^t < UB$  then
11:     $UB \leftarrow UB^t$ 
12:  end if
13:  Compute  $g^t$ 
14:  Compute  $\gamma^t$ 
15:   $\lambda^{t+1} \leftarrow \lambda^t + \gamma^t g^t$ 
16:   $t \leftarrow t + 1$ 
17: end while
18: return  $UB$ 

```

independent problems. The first one is maximizing $(c^T + \lambda^T)y$ under the constraints $Ay \leq b$ and the second one is maximizing $-\lambda^T z$ under the constraints $Dz \leq p$.

Given a Lagrangian relaxation of a problem, we compute the tightest, *i.e.*, lowest upper bound by minimizing over the Lagrangian multipliers λ . Considering the objective function with respect to λ , we observe that each feasible solution \bar{x} corresponds to a linear function in λ . The intersection of these linear functions for all feasible solutions \bar{x} is piecewise linear and convex. We minimize the multipliers λ over this intersection, obtaining for relaxation (2.12)–(2.14) the so-called *Lagrangian dual*

$$\min_{\lambda \geq 0} \max_x c^T x + \lambda^T (p - Dx) \quad (2.24)$$

$$\text{s.t. } Ax \leq b \quad (2.25)$$

$$x \text{ integer.} \quad (2.26)$$

For ILPs, there is only weak duality between primal and dual problems. The objective function value of the optimal solution of the Lagrangian dual problem is therefore larger than or equal to the objective function value of the optimal solution of the original, primal problem.

Objective function (2.24) is convex, but only piecewise linear, and thus non-differentiable. We therefore have to use minimization methods for convex, non-differentiable functions. One such a method that is commonly used is *subgradient optimization* [37]. Subgradient optimization is an iterative method in which at each iteration a subgradient, *i.e.*, a direction in which the objective function decreases, is computed. Then, the Lagrangian multipliers are adjusted with respect to this subgradient. The stepsize by which the Lagrangian multipliers are changed depends in practice on the gap between the objective function value of the best feasible solution

Chapter 2. Preliminaries from combinatorial optimization

found so far, LB , and the current upper bound UB . The approach for solving the Lagrangian dual problem (2.24)–(2.26) is outlined in Algorithm 1.

The relationship between the objective function value of an ILP, z_{ILP} , of its Lagrangian dual, z_{LD} , and of its LP relaxation, z_{LP} , is as follows,

$$z_{ILP} \leq z_{LD} \leq z_{LP}. \quad (2.27)$$

For the lower bound on z_{LD} equality holds if and only if the polyhedron $\{x \in \mathbb{R}^n | Dx \leq p\}$ spanned by the relaxed constraints has integer vertices. Then the objective function value of ILP and Lagrangian dual are identical, *i.e.*, $z_{ILP} = z_{LD}$. For the upper bound on z_{LD} , equality $z_{LD} = z_{LP}$ holds if and only if the polyhedron $\{x \in \mathbb{R}^n | Ax \leq b\}$ spanned by the constraints $Ax \leq b$ that are not relaxed has integer vertices. Further, the objective function value obtained by Lagrangian relaxation after variable splitting can be tighter than the bound obtained when relaxing constraints directly [33].

2.5 Cutting plane method

The *cutting plane method* solves linear programs with a large, *e.g.* exponential number of constraints. It is based on iteratively computing relaxations of the original problem. A relaxation is solved to optimality. Then, we check whether the optimal solution of the relaxed problem is feasible with respect to the original problem. If this is the case, we found the optimal solution of the original problem. If the relaxed solution is not feasible for the original problem, then at least one constraint of the original problem is violated. If this constraint is added to the relaxed problem, it “cuts” off the current relaxed solution from the polyhedron of solutions to the original problem. Such a constraint is therefore called a *cutting plane*. Figure 2.2 visualizes the approach. In practice, several such cutting planes may be added to the relaxation before solving it again.

Cutting plane methods need an oracle that, given a relaxed solution, decides whether this solution satisfies all constraints of the original problem, and if not, returns a violated constraint. These two points are answered by solving a so-called *separation problem*. The cutting plane method is summarized in Algorithm 2.

If cutting plane methods are applied to integer linear programs, usually LP relaxations are used. Cutting planes then iteratively separate fractional solutions until the solution has integer value or no more cutting planes can be determined. There are various ways of determining cutting planes for integer linear programming.

The cutting plane method solves relaxations of the original problem and therefore provides upper bounds. Such upper bounds can be used within higher level algorithms like for example branch-and-bound.

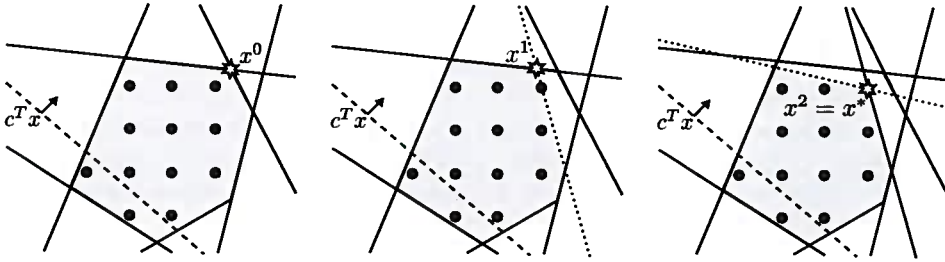


Figure 2.2: The cutting plane method solves an ILP problem. The gray area denotes the polyhedron described by the constraints of the current relaxed problem. The dashed line denotes the objective function $c^T x$ which increases in the direction in which the arrow points. The integer feasible solutions are colored black. Solving the LP relaxation, we obtain the relaxed solution x^0 . After adding a cutting plane (dotted line), we obtain a new relaxed solution x^1 . Finally, after adding a second cutting plane, we obtain solution x^2 which has integer value and is thus the optimal solution x^* of the ILP. A cutting plane method solves only the LP relaxation of an ILP, but here, since the optimal solution has integer value, the solution of the LP relaxation is also the solution of the ILP.

Algorithm 2 Solving an LP problem using the cutting plane method.

```

1:  $P$  // The original problem
2:  $P^t$  // The relaxed problem in iteration  $t$ 
3:  $t \leftarrow 0$  // Iteration
4: while True do
5:   Compute optimal solution  $x^t$  for  $P^t$ 
6:   if  $x^t$  feasible for  $P$  then
7:     return  $x^t$ 
8:   else
9:     Find a cutting plane  $a_i x \leq b_i$  that all solutions of  $P$  satisfy, but not  $x^t$ 
10:     $P^{t+1} \leftarrow P^t$  with additional constraint  $a_i x \leq b_i$ 
11:     $t \leftarrow t + 1$ 
12:   end if
13: end while

```

2.6 Branch-and-bound

Branch-and-bound is a divide-and-conquer approach. A problem is repeatedly split into subproblems until such a subproblem can be solved. For each subproblem, we have a global lower and a local upper bound. These bounds are used to *fathom*, or prune, other subproblems or to decide whether a subproblem needs to be split further. In the worst case, a branch-and-bound algorithm enumerates all solutions, and if there is an exponential number of them, it has thus exponential run time. In practice, the performance depends on the quality of the bounds and on the ability to split a problem

into non-overlapping subproblems.

A branch-and-bound algorithm generates a *branch-and-bound tree*. It starts by solving a relaxation of the entire problem, which is located at the *root* node of the tree. This is, for example, a Lagrangian relaxation or an LP relaxation. From this relaxation, we obtain a local upper bound. Further, we try to find a good feasible solution, for example by the use of heuristics. Such a feasible solution constitutes a global lower bound. If lower and upper bound in the root node are not equal, *i.e.*, the relaxed solution does not satisfy all constraints of the original problem, we split the problem into several subproblems. In the case of binary ILP variables, for example, we can split the problem into two subproblems: in one of them, we fix a specific variable to 0 and in the other one we fix it to 1. The optimal solution will be in one of these two branch-and-bound *branches*, *i.e.*, the specified variable will be either 0 or 1 in the global optimal solution. For both subproblems we then again compute lower and upper bounds, respecting the fixed variable value. If we succeed in finding a better feasible solution than in the root node, we update the global best feasible solution and set the global lower bound to its objective function value. Further, in each branch-and-bound node we obtain a local upper bound, which is an upper bound on the maximum objective value that can be reached within any child node in the respective branch.

The idea behind branch-and-bound is that the global lower bound and local upper bounds can be used for fathoming. There are three cases in which we can fathom branch-and-bound nodes and refrain from further enumerating their subsequent branch. The first situation is the one in which a local upper bound is less than the global lower bound of a feasible solution. In this case a node (or any of its children) can not improve over the best feasible solution found so far. The second situation is the one in which the subproblem in a node turns out to be infeasible. Finally, in a third situation, we might be able to find a relaxed solution that is feasible for the original problem. In this case we have solved the respective subproblem to optimality and there is no need in further splitting it. Using these three rules for fathoming, we process all nodes, one after another, and either split or fathom them. Once all branch-and-bound nodes have been enumerated in this fashion, we found the provably optimal solution to the original problem. The branch-and-bound strategy is outlined in Algorithm 3.

2.7 Dynamic programming

Dynamic programming (DP) is a way of solving a problem by assembling its optimal solution from the optimal solutions of smaller, independent subproblems. In the following, we consider dynamic programming only for the task of computing alignments, for which it is a standard method in Bioinformatics. In this context, it is used to compute the score-optimal sequential alignment of two proteins in which each aligned residue pair (i, k) with residue i from protein A and residue k from protein B contributes a score c_{ik} to the overall, additive alignment score. Gap costs can be incorporated. Depending on the way in which gap costs are applied, we can distinguish between global, semi-global and local alignment. In a global alignment every gap is penalized. In a semi-global alignment, N- and C-terminal gaps are cost-free. Finally, in a local

Algorithm 3 Branch-and-bound algorithm.

```

1:  $Q \leftarrow \{\text{root}\}$  // The subproblem list, root is the entire problem
2:  $LB \leftarrow -\infty$  // Global best lower bound
3:  $t \leftarrow 0$  // Subproblem number
4: while  $Q \neq \emptyset$  do
5:    $p \leftarrow$  a subproblem from  $Q$ 
6:   Solve  $p$ 
7:    $UB^t \leftarrow$  upper bound on optimal solution of  $p$ 
8:   if  $p$  infeasible or  $UB^t < LB$  then
9:      $Q \leftarrow Q \setminus \{p\}$  // Fathom  $p$ 
10:     $t \leftarrow t + 1$ 
11:    continue
12:   end if
13:    $LB^t \leftarrow$  lower bound on optimal solution of  $p$ 
14:    $x^t \leftarrow$  best feasible solution found for  $p$ 
15:   if  $LB^t > LB$  then
16:      $LB \leftarrow LB^t$  // Found a better global lower bound
17:      $x^* \leftarrow x^t$  // Found the currently best feasible solution
18:     for each  $m \in Q$  do
19:       if  $m$ 's local upper bound is less than  $LB$  then
20:          $Q \leftarrow Q \setminus \{m\}$  // Fathom  $m$ 
21:       end if
22:     end for
23:   end if
24:   // Split  $p$  if it was not solved to optimality
25:   if  $UB^t > LB^t$  then
26:     Split  $p$  into two distinct subproblems  $n$  and  $o$ 
27:     Initialize their local upper bounds with  $UB^t$ 
28:      $Q \leftarrow Q \cup \{n, o\}$  // Add  $n$  and  $o$  to subproblem list
29:   end if
30:    $Q \leftarrow Q \setminus \{p\}$  // Finished exploring  $p$ 
31:    $t \leftarrow t + 1$ 
32: end while

```

Chapter 2. Preliminaries from combinatorial optimization

alignment, gap costs are only applied within the aligned region, unaligned N- and C-terminal residues preceding or succeeding this aligned region are not penalized. We use in this thesis global alignment with linear or affine gap costs. The corresponding algorithm in the first case is by Needleman and Wunsch [81] and in the second case by Gotoh [30] (*cf.* Section 1.2). In the following we give the corresponding dynamic programming recursions.

We use a matrix dp of size $(n_A + 1) \times (n_B + 1)$, where n_A and n_B are the lengths of the two proteins. The score-optimal sequential alignment of the two proteins can be computed using the following DP recursion

$$dp(i, k) = \max \begin{cases} dp(i-1, k-1) + c_{ik} \\ dp(i-1, k) - \nu \\ dp(i, k-1) - \nu \end{cases} . \quad (2.28)$$

Here, $\nu \geq 0$ is the gap penalty. The dynamic programming matrix is initialized with

$$dp(0, 0) = 0, \quad (2.29)$$

$$dp(i, 0) = -i\nu, \quad i = 1, 2, \dots, n_A, \quad (2.30)$$

$$dp(0, k) = -k\nu, \quad k = 1, 2, \dots, n_B. \quad (2.31)$$

The DP cell $dp(i, k)$ contains the score of the optimal alignment of the subsequences of length i and k and $dp(n_A, n_B)$ consequently the score of the optimal alignment of the entire sequences. For each field in the matrix we encode in a separate traceback matrix, which of the three cases in (2.28) resulted in the maximum. In the first case of (2.28), residues i and k have been aligned, in the second case, residue i has been aligned to a gap in protein B and in the third case, residue k has been aligned to a gap in protein A .

Besides the linear gap cost model, there is also a more elaborate, affine gap cost model. In an affine gap cost model, different costs apply for opening a gap and extending a gap. Linear gap costs are included as a special case, since the gap open and gap extension costs may have the same value. Affine gap costs are biologically more realistic: Opening a gap that ranges over several residues can occur from one evolutionary event (one indel, *i.e.*, insertion or deletion) and is therefore more likely than several individual gaps for which we have to assume an individual indel each.

Let $\nu \geq 0$ be the gap open and $\xi \geq 0$ the gap extension penalty. Affine gap penalties are given by

$$\nu + n_g \xi, \quad (2.32)$$

where n_g is the number of gap extensions, which is the length of the gap minus 1. Usually, the gap open penalty is greater than the gap extension penalty, thus $\nu > \xi$, since opening a gap is considered worse than extending a gap.

For affine gap costs, the optimal alignment of highest score can be determined by Gotoh's algorithm. Here, the penalty of a residue aligned to a gap depends on whether the previous residue has also been aligned to a gap or not. Therefore, Gotoh's algorithm keeps track of these different possibilities. This means that for determining the maximum score of subsequences of length i and k , not only three cases need to be

considered, aligning i and k , aligning i to a gap and aligning k to a gap. We also need to consider that existing gaps in any of the two sequences can be extended. Hence, the maximum score for aligning subsequences of length i and k may be obtained by gap extension. For this reason, one dynamic programming matrix is not sufficient, instead we need three. Matrix M keeps track of the alignment of residues, matrix G_A of gaps in protein A and matrix G_B of gaps in protein B ;

$$M(i, k) = \max \begin{cases} M(i-1, k-1) + c_{ik} \\ G_A(i-1, k-1) + c_{ik} \\ G_B(i-1, k-1) + c_{ik} \end{cases}, \quad (2.33)$$

$$G_A(i, k) = \max \begin{cases} M(i, k-1) - \nu \\ G_A(i, k-1) - \xi \\ G_B(i, k-1) - \nu \end{cases}, \quad (2.34)$$

$$G_B(i, k) = \max \begin{cases} M(i-1, k) - \nu \\ G_A(i-1, k) - \nu \\ G_B(i-1, k) - \xi \end{cases}. \quad (2.35)$$

These recursions allow that a gap in protein A may be followed immediately by a gap in protein B , which constitutes an adaption of the recursions of the original algorithm that disallows this situation. We will address the motivation for this choice in the following chapter. The dynamic programming matrix M contains the maximum score for aligning the subsequence up to residue i of protein A with the subsequence up to residue k of protein B , terminating with the alignment of i with k . Three situations can occur: first, the previous residues $i-1$ and $k-1$ have also been aligned, secondly a gap which possibly ranges over several residues in protein A has been closed by residues i and k and thirdly a gap which might range over several residues in protein B has been closed. The dynamic programming matrix G_A contains the maximum score if residue k of protein B has been inserted and is therefore aligned to an either newly introduced or extended gap in protein A . Also here, the previous three situations are (i) the alignment of two residues, in which case a new gap is opened, (ii) the extension of a gap in A and (iii) a gap in B , which is now followed by a newly opened gap in A . This third case of gaps in one protein followed immediately by gaps in the other protein is in sequence alignment often not allowed, but we consider it since we want to permit entirely unaligned regions. Analogously, the dynamic programming matrix G_B contains the maximum score if residue i of protein A has been inserted and is therefore aligned to an either newly introduced or extended gap in protein B . The initialization of the three matrices is as follows;

$$M(0, 0) = G_A(0, 0) = G_B(0, 0) = 0, \quad (2.36)$$

$$M(i, 0) = G_A(i, 0) = -\infty, \quad i = 1, 2, \dots, n_A, \quad (2.37)$$

$$M(0, k) = G_B(0, k) = -\infty, \quad k = 1, 2, \dots, n_B, \quad (2.38)$$

$$G_A(0, k) = -\nu - (k-1)\xi, \quad k = 1, 2, \dots, n_B, \quad (2.39)$$

$$G_B(i, 0) = -\nu - (i-1)\xi, \quad i = 1, 2, \dots, n_A. \quad (2.40)$$

Chapter 2. Preliminaries from combinatorial optimization

Since the alignment might end with the last two residues of both proteins aligned or with a gap in protein A or a gap in protein B , the maximum overall alignment score is given by

$$\max\{M(n_A, n_B), G_A(n_A, n_B), G_B(n_A, n_B)\}. \quad (2.41)$$

The optimal alignment can again be determined via traceback.

Double dynamic programming is a technique that is applied in structure alignment. It is best described as dynamic programming on two levels. On the lower level, a residue pair (i, k) is fixed to be aligned and the corresponding contribution of other residue pairs (j, l) given (i, k) is used as dynamic programming score c_{jl}^{ik} for the alignment of residues j and l . Then residues are aligned such that they do not cross with (i, k) . The overall lower level dynamic programming score

$$c_{ik} = \sum_{\substack{j,l \text{ aligned} \\ j,l \text{ and } i,k \text{ non-crossing}}} c_{jl}^{ik} \quad (2.42)$$

is the score contribution for aligning residue i with residue k . It is used as upper level DP score of pair (i, k) . The upper level dynamic program then computes the alignment which maximizes the overall score contributions c_{ik} of aligned residues, *i.e.*, the alignment for which $\sum_{i,k \text{ aligned}} c_{ik}$ is maximal.

Mathematical models and structure alignment algorithms

In this chapter we introduce the distance matrix alignment problem, set up a mathematical model for finding an optimal distance matrix alignment and devise algorithms to solve this model or variants of it.

In the following section we start by formalizing the structure alignment problem. We then focus in Section 3.2 on the case of protein structure distance matrix alignment. In Section 3.3 we cast the distance matrix alignment problem into suitable graph problems. Using one of these graph representations, we set up a mathematical model for distance matrix alignment in Section 3.4. In Section 3.5 we then use techniques from combinatorial optimization to develop three algorithms for computing bounds on the optimal distance matrix alignment score. The corresponding approaches described in Sections 3.5.1, 3.5.2 and 3.5.3 are published in [113],[116] and [114], respectively. Sections 3.6 and 3.7 describe variable elimination as preprocessing and branch-and-bound, both are techniques that can be used for improving all three presented algorithmic approaches. Finally, Section 3.8 describes the implementation of each algorithm.

3.1 Structure alignment representations

Protein structure alignments are often given either in alignment format according to Definition 1 or by a set of pairs of structurally equivalent residues, which is called the *alignment trace*. In a trace, each residue is uniquely identifiable, for example by sequentially numbering residues from N- to C-terminus. It has been introduced in [95] and can be formally defined as follows.

Definition 2 (Alignment trace). *Let A and B be two proteins of length n_A and n_B , respectively. An alignment trace is a set of residue pairs*

$$\{(i_1, k_1), (i_2, k_2), \dots, (i_n, k_n)\}, \quad (3.1)$$

with the property that $1 \leq i_1 < i_2 < \dots < i_n \leq n_A$ and $1 \leq k_1 < k_2 < \dots < k_n \leq n_B$. Here, a pair (i_p, k_p) denotes that residue i_p in protein A is aligned to residue k_p in protein B . The number of aligned residues is $n \leq \min\{n_A, n_B\}$.

Chapter 3. Mathematical models and structure alignment algorithms

Note that the trace description implicitly contains the information which residues are not mapped and thus aligned to gaps, but that it does not contain information on stretches of gaps that directly follow each other. There is therefore a one-to-one mapping between a trace and an alignment if and only if for no consecutive residue pairs (i_p, k_p) and (i_{p+1}, k_{p+1}) in the trace both $i_{p+1} - i_p > 1$ and $k_{p+1} - k_p > 1$ holds. In the case that $i_{p+1} - i_p > 1$ and $k_{p+1} - k_p > 1$, a trace can be mapped to more than one alignment. The possibly ambiguous alignment representation of a trace is usually no problem: In the case of linear gap costs, all alignments to which a trace can be mapped have the same score. We can just pick the biologically most plausible one with the minimum number of two stretches of gaps. In the case of affine gap costs with a gap open penalty that is larger than the gap extension penalty, the same treatment of unaligned regions will then maximize the score.

Some structure alignment programs return only an alignment trace and no alignment. Others compute a trace of structurally equivalent residues and align regions without structural correspondences using other criteria. The program DALI, for example, denotes in the alignment the structural trace in capital letters and the remaining aligned or unaligned residues in lower case letters.

Figure 3.1 illustrates several different representations of a protein structure alignment. In a), the alignment format as well as the alignment trace are displayed. In alignment format, usually the amino acid type is used to identify a residue. Figure 3.1 a) uses the sequential numbers instead, in order to illustrate the relationship to the other alignment representations. The difference between a sequence and a structure alignment in the common alignment format is that a sequence alignment uses sequential information and a structure alignment structural information for the identification of aligned residues.

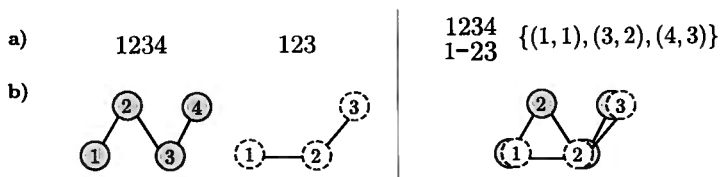


Figure 3.1: Different protein and alignment representations of protein A with $n_A = 4$ residues and protein B with $n_B = 3$ residues. a) The amino acid sequence representation. Instead of the amino acid, the corresponding sequential residue number is given. On the right the alignment format is displayed as well as the alignment trace. Both denote which residues structurally match. In alignment format, the second residue of protein A is unmatched and thus aligned to a gap. b) The corresponding superposition. Given the alignment of residue 1 with 1, 3 with 2 and 4 with 3, protein B is translated and rotated such that the distances between these aligned residues minimize RMSD_e .

Given a structure alignment or trace, another representation is the corresponding superposition of the two protein structures. It can be computed in polynomial time using Kabsch' algorithm [54]. The representative 3D coordinates of the residues of one

protein are rotated and translated such that the RMSD_c between aligned residues is minimized. This is schematically illustrated in Figure 3.1 b). An example is given in Figure 3.2.

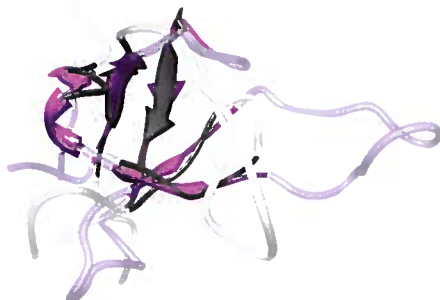


Figure 3.2: Superposition according to a structure alignment of PDB structure 1aww, chain A (gray) and PDB structure 1gxi, chain E (magenta). The trace is colored in dark tone. Unaligned residues are colored in light tone.

3.2 Distance matrix alignment

In this section we introduce protein structure inter-residue *distance matrices*. We show how these matrices can be used for pairwise structure alignment and formally introduce the corresponding distance matrix alignment problem.

Definition 3 (Distance matrix). *A protein structure distance matrix A is a symmetric $n_A \times n_A$ matrix, where n_A is the length of the protein. Rows and columns, from left to right and top to bottom, correspond to residues in their sequential order from N- to C-terminus. An entry $A_{i,j}$ denotes the distance between two representative 3D points of residues i and j , for example their C_α atom coordinates.*

Figure 3.3 visualizes the distance matrices of two structurally similar proteins whose superposition was displayed in Figure 3.2.

An alignment trace not only determines which residues are aligned, but also which distance matrix rows and columns. Let A be the distance matrix of the first and B be the distance matrix of the second protein. Alignment trace $\{(i_1, k_1), (i_2, k_2), \dots, (i_n, k_n)\}$ assigns the $n \times n$ submatrix $A_{\{i_1, i_2, \dots, i_n\}\{i_1, i_2, \dots, i_n\}}$ to the $n \times n$ submatrix $B_{\{k_1, k_2, \dots, k_n\}\{k_1, k_2, \dots, k_n\}}$. The aligned distance pairs are $(A_{i_o, i_p}, B_{k_o, k_p})$ for $o = 1, \dots, n$ and $p = 1, \dots, n$. The submatrices are symmetric, such that of the n^2 distances that are mapped, all inter-residue distance pairs are represented twice, except for diagonal entries. A distance matrix alignment is illustrated in Figure 3.4. Formally, we define it as follows.

Definition 4 (Distance matrix alignment). *An alignment of two distance matrices A and B of two proteins of length n_A and n_B assigns a subset of rows (and corresponding columns) of A to a subset of rows (and corresponding columns) of B taking into*

Chapter 3. Mathematical models and structure alignment algorithms

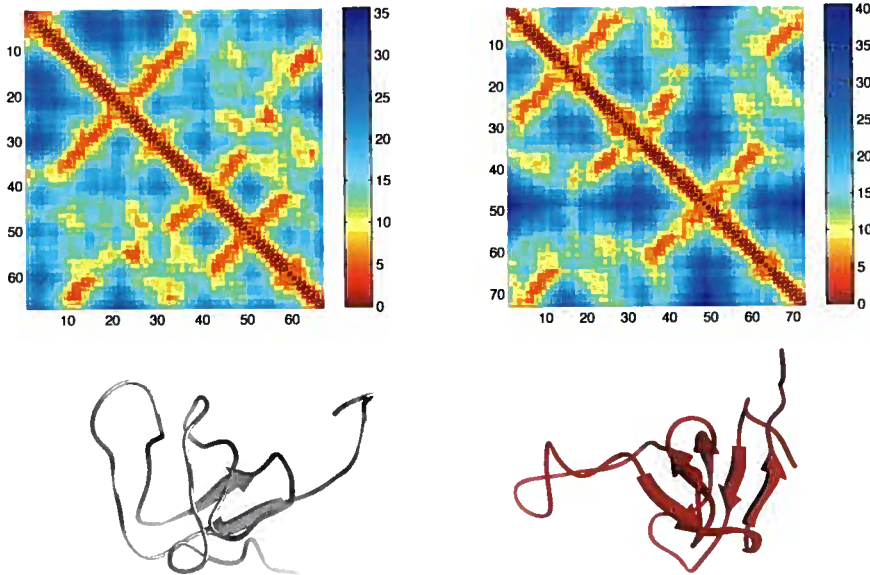


Figure 3.3: Visualization of the distance matrices of two structurally similar proteins. The axes denote the residue number. The color reflects the distance at a particular position in the matrix in ångström. Left: PDB structure 1aww, chain A, with a length of 67 residues. Right: PDB structure 1gxi, chain E, with a length of 73 residues.

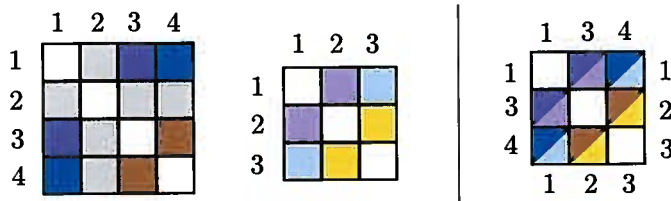


Figure 3.4: A distance matrix alignment $(I, K) = ((1, 3, 4), (1, 2, 3))$ which corresponds to the alignment trace $\{(1, 1), (3, 2)(4, 3)\}$. There are three symmetric aligned inter-residue distances, (A_{13}, B_{12}) , (A_{14}, B_{13}) and (A_{34}, B_{23}) . The aligned distances on the diagonal are (A_{11}, B_{11}) , (A_{33}, B_{22}) and (A_{44}, B_{33}) .

consideration the sequential order of the rows. It is described by a pair of sequences of matrix indices (I, K) with $I = (i_1, i_2, \dots, i_n)$ and $K = (k_1, k_2, \dots, k_n)$ satisfying $|I| = |K| = n$, $1 \leq i_1 < i_2 < \dots < i_n \leq n_A$ and $1 \leq k_1 < k_2 < \dots < k_n \leq n_B$. Indices I denote rows in matrix A and indices K rows in matrix B . They can be interpreted as aligning residue i_p to residue k_p for $p = 1, \dots, n$.

Note that there is a one-to-one correspondence between distance matrix alignments and alignment traces.

A distance matrix alignment (I, K) of length n induces an alignment of pairs of inter-residue distances and we define its score as

$$S(I, K) := \sum_{o=1}^n \sum_{p=1}^n s(A_{i_o, i_p}, B_{k_o, k_p}) + S_{\text{gap}}(I, K). \quad (3.2)$$

Here, $s : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \rightarrow \mathbb{R}$ gives the scoring term of aligning individual distances from matrix A with distances from matrix B . The function $S_{\text{gap}}(I, K)$ penalizes the gaps of the alignment (I, K) . In the case of unaligned regions, it assumes two stretches of gaps following each other, cf. Section 3.1. We denote by $\mathcal{A}(A, B)$ the set of all, exponentially many possible alignments of two distance matrices A and B .

The problem consists now in finding the best alignment between the two matrices with respect to S . It is *NP*-hard, following from [64], since variable length gaps are allowed in the alignment and interactions between amino acid residues from the sequence are admitted into the scoring function.

Problem 1 (Optimal distance matrix alignment). *Given two distance matrices A and B , find a score-optimal alignment $(I, K)^*$ of A and B with*

$$(I, K)^* = \arg \max_{(I, K) \in \mathcal{A}(A, B)} S(I, K). \quad (3.3)$$

This is the problem that we solve in this thesis.

3.3 Graph-based formalizations

In the following descriptions and illustrations of our models and algorithms, we will use two different, but equivalent graph-based representations of distance matrix alignment. They formally describe the set $\mathcal{A}(A, B)$, i.e., the set of *all* alignments that can be generated from the distance matrices A and B of two proteins. A distance matrix alignment corresponds to an induced subgraph within these graphs.

We use the first graph formulation for visualization, because its nodes and edges directly represent residues and inter-residue distances and it thus intuitively captures the distance matrix alignment problem. This representation is used in most literature on exact algorithms for structure alignment and we therefore include it for the sake of completeness. The second formulation uses a product graph in which nodes and edges

Chapter 3. Mathematical models and structure alignment algorithms

represent *pairs* of residues and inter-residue distances, respectively. This formulation is less intuitive at first, but it facilitates the mathematical formalization of the distance matrix alignment problem. We therefore usually refer to this representation in the description of our mathematical models and algorithms.

We call the first graph-based formalization the *matching graph*. It is depicted in Figure 3.5 b).

Definition 5 (Matching graph). *In the matching graph, proteins A and B are represented by graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Nodes V_1 and V_2 , respectively, are sequentially labeled and typically visualized linearly from left to right. Edges E_1 and E_2 , respectively, denote inter-residue distances. Further, alignment edges L connect any residue of the first protein with any residue of the second protein, i.e., are defined by $L = V_1 \times V_2$.*

An alignment corresponds to a subset of alignment edges L that is non-crossing, i.e., for any pair (i, k) and (j, l) either $i < j$ and $k < l$ or $i > j$ and $k > l$ holds. Such an alignment induces pairs of edges from E_1 and E_2 . For example, the alignment edges $(i, k) \in L$ and $(j, l) \in L$ induce the pairing of the graph edge $(i, j) \in G_1$ with $(k, l) \in G_2$. Such a pair of pairs represents aligned inter-residue distances, i.e., inter-residue distances whose endpoint residues are aligned, see Figure 3.5 b).

Our second graph-based formulation of the structure alignment problem is shown in Figure 3.5 c). It is called the *alignment graph*. Initially, it has been proposed for sequence alignment [87]. For structure alignment we define it as follows.

Definition 6 (Alignment graph). *For two proteins of length n_A and n_B , the alignment graph $G = (V, E)$ is a $n_A \times n_B$ product or grid graph. Rows, from bottom to top, represent the residues of the first protein and columns, from left to right, the residues of the second protein, both sequentially from N- to C-terminus. A node $i.k$ in the alignment graph indicates the alignment of residue i from the first with residue k from the second protein. There exist directed edges $(i.k, j.l)$ between any pair of nodes for which $i < j$ and $k < l$. Edges are thus south-west to north-east bound. An edge $(i.k, j.l)$ denotes the matching of distance A_{ij} with distance B_{kl} .*

The alignment graph is visualized in Figure 3.5 c) and 3.6. We say a node $j.l$ is strictly smaller than $i.k$ if and only if $j < i$ and $l < k$ and strictly larger than $i.k$ if and only if $j > i$ and $l > k$. By definition, any edge in the alignment graph has the property that its tail is strictly smaller than its head. Because of the partial ordering between nodes and because there exist only edges between pairs of ordered nodes, the alignment graph is a directed acyclic graph, see for example Figure 3.6.

Every alignment is described by a *strictly increasing path*, which is defined as follows.

Definition 7 (Strictly increasing path). *A strictly increasing path is a subset $\{i_1.k_1, i_2.k_2, \dots, i_n.k_n\}$ of alignment graph nodes that can be ordered such that each node is strictly larger than the previous one, i.e., $i_1 < i_2 < \dots < i_n$ and $k_1 < k_2 < \dots < k_n$.*

There is a one-to-one correspondence between distance matrix alignments and strictly increasing paths. Every distance matrix alignment can be represented by nodes of a

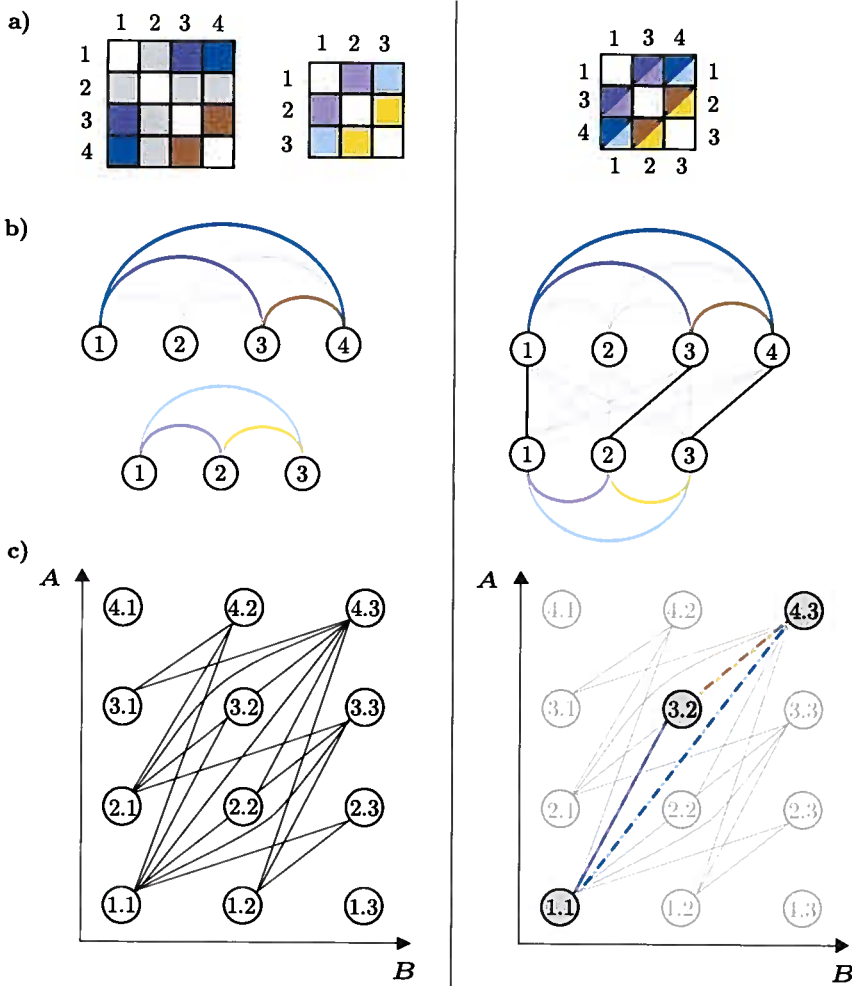


Figure 3.5: Different graph-based formalizations of the distance matrix alignment problem for protein A with $n_A = 4$ residues and protein B with $n_B = 3$ residues. The highlighted distance matrix alignment is $(I, K) = ((1, 3, 4), (1, 2, 3))$. a) The superimposed and collapsed distance matrices highlight three pairs of symmetric, non-diagonal aligned distances. b) The matching graph. On the right, a subset of alignment edges, colored in black, denotes which residues are aligned. They induce three pairs of aligned inter-residue distances, *i.e.*, edges of the graph of the first protein that are matched to edges of the graph of the second protein. c) The alignment graph. On the right, the activated nodes and edges that correspond to the given distance matrix alignment are shown.

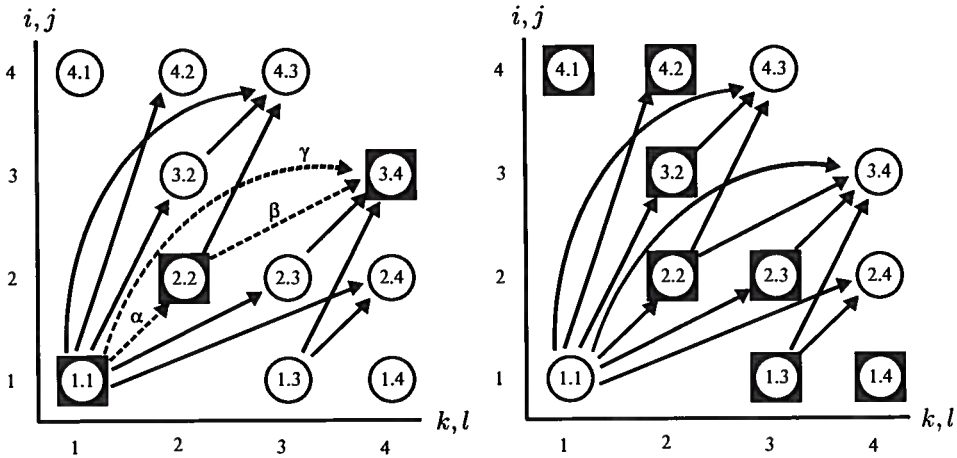


Figure 3.6: Alignment graph for a subset of nodes. Left: the set of nodes $\{1.1, 2.2, 3.4\}$ forms a strictly increasing path that induces a structure alignment with edge score $\alpha + \beta + \gamma$ plus the score of the nodes. Here, the structural score is $\alpha = 2s(A_{12}, B_{12})$, $\beta = 2s(A_{23}, B_{24})$ and $\gamma = 2s(A_{13}, B_{14})$, and the sequence score is $s(A_{11}, B_{11}) + s(A_{22}, B_{22}) + s(A_{33}, B_{44})$. Right: Nodes on the decreasing path $C = \{4.1, 4.2, 3.2, 2.2, 2.3, 1.3, 1.4\}$ mutually contradict.

strictly increasing path together with all induced edges. Its score is the sum of node and edge weights (cf. Figure 3.6, left). Determining an optimal distance matrix alignment is equivalent to finding an induced subgraph of maximum score in the alignment graph.

Nodes in the alignment graph contradict if they do not lie on a strictly increasing path. A set of *mutually* contradicting nodes is called a *decreasing path* and defined as follows.

Definition 8 (Decreasing path). *A decreasing path is a set $C = \{i_1.k_1, i_2.k_2, \dots\}$ of alignment graph nodes for which $i_1 \geq i_2 \geq \dots$ and $k_1 \leq k_2 \leq \dots$ holds.*

See Figure 3.6, right, for an illustration. Decreasing paths of maximum cardinality have the shape of a staircase in which the number of the stairs may vary and their height may be irregular. We denote the set of all decreasing paths by \mathcal{C} ; there is an exponential number of them. Note that we call $C \in \mathcal{C}$ a decreasing path in analogy to strictly increasing paths, but that in fact C is not a path in the alignment graph since its nodes are not connected by edges. Note also that strictly smaller and strictly larger nodes exist only in strictly increasing paths and not in decreasing paths.

We introduce some further notation describing the neighborhoods of a node $i.k$, see Figure 3.7. By $V^-(i.k)$ we denote the set of nodes that are strictly smaller than $i.k$, i.e., the left neighborhood, and by $V^+(i.k)$ the set of nodes that are strictly larger than $i.k$, i.e., the right neighborhood. The set $\mathcal{C}_{i.k}^-$ contains all decreasing paths in $V^-(i.k)$ and the set $\mathcal{C}_{i.k}^+$ all decreasing paths in $V^+(i.k)$.

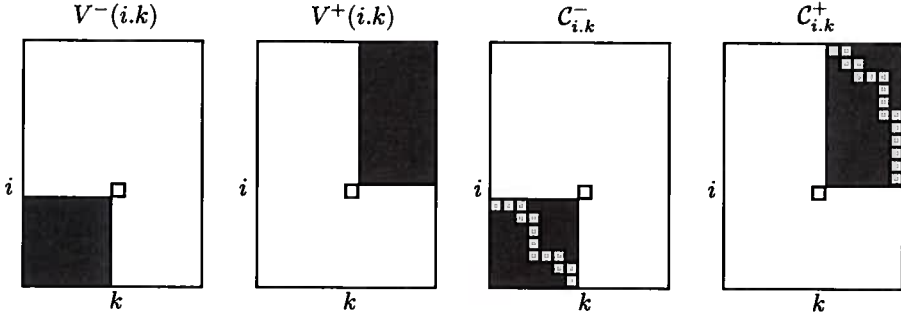


Figure 3.7: Visualization of the left neighborhood $V^-(i,k)$ and right neighborhood $V^+(i,k)$ of node i,k in black. In gray, two examples of decreasing paths C , from $C_{i,k}^-$ in the left neighborhood and $C_{i,k}^+$ in the right neighborhood.

3.4 Mathematical model

In this section we introduce our mathematical model for computing score-optimal alignments of inter-residue distance matrices. It is the first model for protein structure alignment in the general setting of optimizing any distance-matrix based scoring function (3.2) and was published in [113]. We use the alignment graph representation according to Definition 6 and give an integer linear programming model.

The ILP formulation uses two types of variables. Binary variables x_{ik} represent nodes in the alignment graph and indicate whether residues i and k are aligned, in which case $x_{ik} = 1$. Variables y_{ikjl} denote whether the alignment graph edge (i,k,j,l) is present in the solution, *i.e.* whether distance A_{ij} is aligned with distance B_{kl} . In this case $y_{ikjl} = 1$. It is not necessary to restrict variables y_{ikjl} to be binary, only to be greater than zero, because they are implicitly constraint to be either 0 or 1. The ILP is then

$$\max \sum_{i=1}^{n_A-1} \sum_{j=i+1}^{n_A} \sum_{k=1}^{n_B-1} \sum_{l=k+1}^{n_B} 2s(A_{ij}, B_{kl})y_{ikjl} + \sum_{i=1}^{n_A} \sum_{k=1}^{n_B} s(A_{ii}, B_{kk})x_{ik} \quad (3.4)$$

$$\text{s.t. } x_{ik} \geq \sum_{j,l \in C} y_{ikjl} \quad \forall C \in \mathcal{C}_{i,k}^+, i \in [1, n_A - 1], k \in [1, n_B - 1] \quad (3.5)$$

$$x_{ik} \geq \sum_{j,l \in C} y_{jlik} \quad \forall C \in \mathcal{C}_{i,k}^-, i \in [2, n_A], k \in [2, n_B] \quad (3.6)$$

$$x_{ik} \leq 1 + \sum_{\substack{j,l \in C \\ s(A_{ij}, B_{kl}) \leq 0}} (y_{ikjl} - x_{jl}) \quad \forall C \in \mathcal{C}_{i,k}^+, i \in [1, n_A - 1], k \in [1, n_B - 1] \quad (3.7)$$

$$\sum_{i,k \in C} x_{ik} \leq 1 \quad \forall C \in \mathcal{C} \quad (3.8)$$

$$\mathbf{y} \geq 0 \quad (3.9)$$

$$\mathbf{x} \text{ binary.} \quad (3.10)$$

Note that the objective function (3.4) of this ILP only models the structural part of

Chapter 3. Mathematical models and structure alignment algorithms

objective (3.2), and not $S_{\text{gap}}(\cdot, \cdot)$. It is possible to integrate also linear or affine gap costs. For the sake of simplicity, we do not describe this here but refer to [1] where this is done for multiple sequence alignment. Further, note that in (3.4) symmetric, identical distance pairs are combined, resulting in the factor 2 in the first sum of the objective function.

Constraints (3.8) guarantee that the solution is a proper alignment, *i.e.*, that the nodes represented by x -variables form a strictly increasing path in the alignment graph. For any set of mutually contradicting alignment graph nodes C , there is a constraint that denotes that at most one of these nodes may be in the solution.

Constraints (3.5), (3.6) and (3.7) link x and y variables. Inequalities (3.5) and (3.6) prevent activating edges for which tail or head node are not activated as well. If for example tail $i.k$ is not in the solution and thus $x_{ik} = 0$, no edge $(i.k, j.l)$ can be in the solution; the right hand side of inequality (3.5) is forced to zero. Inequalities (3.6) describe the analogous situation for head nodes $i.k$ of edges $(j.l, i.k)$. Then again, if the tail node $i.k$ is in the solution and thus $x_{ik} = 1$, the right hand sides of (3.5) denotes that the heads $j.l$ of edges $(i.k, j.l)$ can not contradict. The same holds for inequalities (3.6): the tails $j.l$ of edges $(j.l, i.k)$ with common head $i.k$ can not contradict.

Inequalities (3.7) force the activation of edges whose endpoints are activated. If $x_{ik} = x_{jl} = 1$, then also $y_{ikjl} = 1$ must hold. The corresponding simple constraints $x_{ik} + x_{jl} - y_{ikjl} \leq 1$ can be lifted to constraints (3.7), since each tail $j.l$ has, according to constraints (3.5) no edges with contradicting heads $i.k$. Constraints (3.7) are necessary because edges $(i.k, j.l)$ with score $s(A_{ij}, B_{kl}) < 0$ would otherwise never be part of an optimal solution.

Note that all inequality classes have exponential size. Every feasible solution of model (3.4)–(3.10) is a strictly increasing path in the alignment graph together with its induced edges and therefore constitutes a structure alignment.

3.5 Computing lower and upper bounds

In the following we devise three algorithms for computing lower and upper bounds for the distance matrix alignment problem (3.4)–(3.10) of the previous section: a cutting plane method and two different Lagrangian approaches.

3.5.1 Cutting planes

In our cutting plane algorithm [96] (*cf.* Section 2.5), we first solve the LP relaxation of an initial ILP with a reduced number of constraints (3.5)–(3.8). In the case of a fractional solution, we solve a separation problem that either yields a violated inequality cutting off this solution or reports that no such violated inequality exists. In practice, several cutting planes are generated at once. The new constraints are added to the LP relaxation which is solved again, and so on. If no more cutting planes can be found and the solution is still fractional, the formulation comprising the new constraints is

solved within a branch-and-bound algorithm (*cf.* Section 2.6).

We show in the following how cutting planes for model (3.4)–(3.10) can be generated efficiently using a shortest path algorithm in a directed acyclic graph. Shortest path computation for separation of constraints (3.8) which denote that the solution must be a sequential alignment has first been described by Lenhof *et al.* in the context of multiple sequence alignment [66] and RNA sequence structure alignment [65]. Caprara *et al.* separate the same inequalities in the context of protein structure alignment [14]. They further strengthen their formulation by various types of constraints which can be separated efficiently and forbid mutually exclusive alignment graph edges, *i.e.*, y -variables. They report that in practice these cuts for y -variables are very weak.

Here we present for the first time a cutting plane approach for the LP relaxation of the entire protein structure alignment model (3.4)–(3.10). All constraints can be separated using a shortest path algorithm in a suitable directed, acyclic graph $G' = (V', E')$ that is constructed as follows. The node set V' is identical to the the node set V of the alignment graph and contains an additional artificial source node s , *i.e.*, $V' = V \cup \{s\}$. The edge set E' is constructed as follows. We generate edges between any node $i.k$ and $i - 1.k$ as well as between $i.k$ and $i.k + 1$ (if node $i - 1.k$ and $i.k + 1$ exist). Further, the source node s is connected to exactly one node. For a visualization, see Figure 3.8, center. The graph constructed in this way has at most $n_A n_B = O(n_A n_B)$ nodes and $(n_A - 1)n_B + (n_B - 1)n_A + 1 = O(n_A n_B)$ edges. We assign weights to the edges of G'

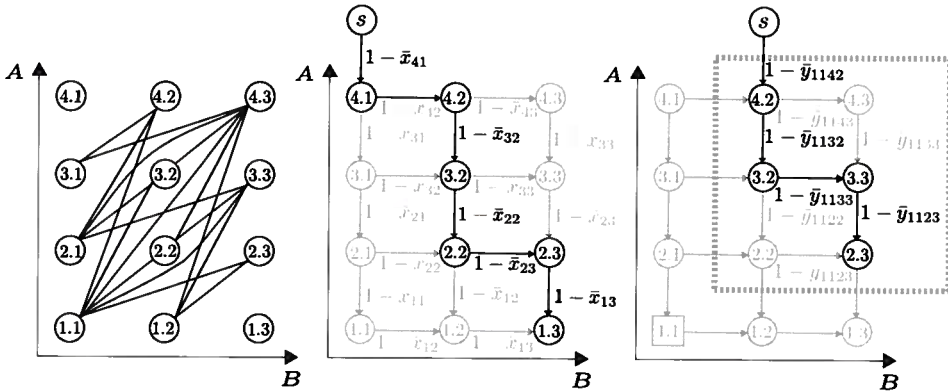


Figure 3.8: Example of the graphs in which we use shortest path computations to detect violated constraints. Left: The alignment graph $G = (V, E)$. Center: The graph $G' = (V', E')$ in which we identify a violated constraint (3.8). The shortest decreasing path is colored black, it is $C = \{4.1, 4.2, 3.2, 2.2, 2.3, 1.3\}$. If for this path $\sum_{i,k \in C} \bar{x}_{ik} > 1$ holds, we identified a violated constraint (3.8). Right: The graph $G' = (V', E')$ for the identification of a violated constraint (3.6) for node $i.k = 1.1$. The right neighborhood $V'^+(i.k)$ is framed by a box. The shortest path $C = \{4.2, 3.2, 3.3, 2.3\}$ is colored black. If $\bar{x}_{ik} < \sum_{j,l \in C} \bar{y}_{ikjl}$ holds, the corresponding inequality $x_{ik} \geq \sum_{j,l \in C} y_{ikjl}$ is a cutting plane.

which correspond to the node weight of their head in the alignment graph G . In the following, let \bar{x} and \bar{y} be the solution of the LP relaxation.

Lemma 1. *The separation problem for constraints (3.8) can be solved in time $O(n_A n_B)$ using a shortest path algorithm in a directed acyclic graph.*

Proof. Cutting planes for inequalities (3.8) can be generated efficiently by assigning the value \bar{x}_{ik} obtained by the linear programming relaxation as weight to each alignment graph node. We can determine the most violated constraint by computing the maximum weight decreasing path C in the alignment graph nodes V . If this weight is greater than one, the nodes of the decreasing path violate constraint (3.8). Since node weights are between zero and one, we can always lift a constraint of type (3.8) by adding x -variables that refer to a corresponding decreasing path of maximum cardinality. Hence, we search in the alignment graph for the maximum weight decreasing path among all decreasing paths of length $(n_A \times n_B) - 1$.

We reformulate this problem to a shortest path problem in graph G' . We connect the source node to $n_A \cdot 1$ and assign the length $(1 - \bar{x}_{ik})$ to edges with head $i.k$. This is visualized in Figure 3.8, center. We can compute the shortest decreasing path in V' by computing the shortest path from the source node to node $1.n_B$ in G' . This can be done in time $O(n_A n_B)$ using a shortest path algorithm in a directed, acyclic graph. If for the path C found in this way $\sum_{i.k \in C} \bar{x}_{ik} > 1$ holds, we identified a violated constraint (3.8). If this is not the case, no such violated constraint exists. \square

Lemma 2. *The separation problem for constraints (3.5) and (3.6) for an alignment graph node $i.k$ can be solved in time $O(n_A n_B)$ using a shortest path algorithm in a directed, acyclic graph.*

Proof. Cutting planes for constraints (3.5) and (3.6) are generated analogous to cutting planes (3.8). For each node $i.k$, a cutting plane (3.5) considers edges with tail $i.k$ and a cutting plane (3.6) edges with head $i.k$. In the first case, we assign for node $i.k$ the weights \bar{y}_{ikjl} to each node $j.l \in V^+(i.k)$, and in the second case we assign \bar{y}_{jlik} to each node $j.l \in V^-(i.k)$. Then we compute the maximum weight decreasing path in $V^+(i.k)$ and $V^-(i.k)$, respectively. If this path has weight greater than \bar{x}_{ik} , we identified a cutting plane.

We again reformulate the maximum weight decreasing path problems to a shortest path problem in G' . In the following we consider constraints (3.5), constraints (3.6) are similar. In G' , we connect the source node s to node $n_A.k + 1$. We further assign length $1 - \bar{y}_{ikjl}$ to edges E' with head $j.l \in V^+(i.k)$. This is displayed in Figure 3.8, right. We then again in $O(n_A n_B)$ compute the shortest path C from the source node s to node $i + 1.n_B$ using a shortest path algorithm in the directed, acyclic graph G' . If for this path it holds that $\bar{x}_{ik} < \sum_{j.l \in C} \bar{y}_{ikjl}$, a cutting plane $x_{ik} \geq \sum_{j.l \in C} y_{ikjl}$ has been identified. Otherwise, no constraint (3.5) is violated. \square

Lemma 3. *The separation problem for constraints (3.7) for an alignment graph node $i.k$ can be solved in time $O(n_A n_B)$ using a shortest path algorithm in a directed, acyclic graph.*

Proof. Similar to constraints (3.5) and (3.6), we identify violated activation constraints (3.7). For a node $i.k$, we assign the weight $\bar{y}_{ikjl} - \bar{x}_{jl}$ to each node $j.l \in V^+(i.k)$ with objective function coefficient $s(A_{ij}, B_{kl}) \leq 0$. Note that, because we consider only edges with negative objective function score, in this case we do not know the cardinality of the path C beforehand. If the weight of the lightest decreasing path C plus one is smaller than \bar{x}_{ik} , we identified a violated inequality (3.7). Otherwise, no such violated inequality exists.

We transform the problem of finding the minimum weight decreasing path C in the alignment graph to a shortest path problem in G' . We assign to edges E' with head $j.l \in V^+(i.k)$ and $s(A_{ij}, B_{kl}) \leq 0$ the weight $\bar{y}_{ikjl} - \bar{x}_{jl}$. Nodes with $s(A_{ij}, B_{kl}) > 0$ and their corresponding edges are ignored and do not contribute to the length of the path. Edge weights $\bar{y}_{ikjl} - \bar{x}_{jl}$ are therefore less than or equal to zero. For computing the shortest path we traverse all nodes in topological order in the directed, acyclic graph G' . This can be done in $O(n_A n_B)$. If for the shortest path C constraint (3.7) is violated, we found a cutting plane. Otherwise, no cutting plane (3.7) exists. \square

We can now obtain upper bounds for the distance matrix alignment model by applying a cutting plane method according to Algorithm 2. It successively solves an LP relaxation of model (3.4)–(3.10) with only a subset of its constraints and adds cutting planes (3.5), (3.6), (3.7), and/or (3.8).

Concluding, we establish the time complexity of solving the separation problem for the entire distance matrix alignment model (3.4)–(3.10) given a solution to the model with only a subset of constraints.

Theorem 1 (Complexity of solving the separation problem for model (3.4)–(3.10)).
The separation problem for the LP relaxation of (3.4)–(3.10) can be solved in time $O(n_A^2 n_B^2)$.

Proof. According to the previous three lemmas, we can detect in $O(n_A n_B)$ whether a constraint (3.5), (3.6), (3.7) or (3.8) of model (3.4)–(3.10) is violated. In order to solve the separation problem, we have to check at most $n_A n_B$ such constraints for every inequality type, one for every alignment graph node, *i.e.*, residue pair. The overall complexity is therefore $O(n_A^2 n_B^2)$. \square

3.5.2 Lagrangian relaxation using variable splitting

In this section we apply Lagrangian relaxation after variable splitting to the distance matrix alignment model (3.4)–(3.10). We then describe an algorithm that solves the relaxation and provides both lower and upper bounds on the optimal distance matrix alignment score.

Both mathematical model and algorithm presented in this section are based on the work of Caprara *et al.* [14]. They compute a pairwise alignment of two protein structures that maximizes the number of common contacts. Two residues are in contact if they are in some sort of chemical interaction, *e.g.* by hydrogen bonding. Their model uses a simple distance criterion: whenever the distance between two residues is below a predefined

Chapter 3. Mathematical models and structure alignment algorithms

distance threshold d_t , the residues are considered to be in contact. The *contact map overlap* (CMO) problem is then the problem of finding the maximum number of common contacts in two proteins. CMO as scoring scheme has first been introduced in [28].

In the following we generalize the integer linear program and Lagrangian approach of Caprara *et al.* to protein structure distance matrix alignment.

Mathematical model after variable splitting. This generalization from the CMO model to a model for distance matrix alignment is straightforward. First, we use a variable y_{ikjl} for any pair of inter-residue distances, not only for pairs of contacts. Further, we generalize the objective function. Finally, we have to add an additional type of constraint that forces distance pairs with negative score into the solution if their endpoints are aligned. This is a novel type of constraint that is not necessary for the contact map overlap problem in which all scores are equal to 1 and hence positive.

The resulting generalization of the ILP of Caprara *et al.* is similar to our model (3.4)–(3.10) of Section 3.4. Binary variables x_{ik} indicate whether residue i is aligned with residue k and variables \tilde{y}_{ikjl} denote the mapping of distance A_{ij} with distance B_{kl} . The main difference to model (3.4)–(3.10) from Section 3.4 is that Caprara *et al.* apply variable splitting. By using this technique, they can formulate a relaxed problem that can be solved efficiently. In our general model (3.4)–(3.10), it holds for y_{ikjl} -variables that $i < j$ and $k < l$. In the model of Caprara *et al.*, a variable y_{ikjl} with $i < j$ and $k < l$ is split into two variables, \tilde{y}_{ikjl} and \tilde{y}_{jlik} . This variable splitting corresponds to adding for each edge in the alignment graph a reverted edge that is north-east to south-west bound. When splitting the variables, we also need to split the structural score between them. The score of each variable \tilde{y}_{ikjl} is now $s(A_{ij}, B_{kl})$, instead of previously $2s(A_{ij}, B_{kl})$. The corresponding integer linear programming formulation is then

$$\max_{\mathbf{x}, \mathbf{y}} \sum_{i=1}^{n_A} \sum_{\substack{j=1 \\ j \neq i}}^{n_A} \sum_{k=1}^{n_B} \sum_{\substack{l=1 \\ l \neq k}}^{n_B} s(A_{ij}, B_{kl}) \tilde{y}_{ikjl} + \sum_{i=1}^{n_A} \sum_{k=1}^{n_B} s(A_{ii}, B_{kk}) x_{ik} \quad (3.11)$$

$$\text{s.t. } x_{ik} \geq \sum_{j,l \in C} \tilde{y}_{ikjl} \quad \forall C \in \mathcal{C}, i \in [1, n_A], k \in [1, n_B] \quad (3.12)$$

$$\sum_{i,k \in C} x_{ik} \leq 1 + \sum_{\substack{j,l \in C, i,k < j,l \\ s(A_{ij}, B_{kl}) \leq 0}} (\tilde{y}_{ikjl} - x_{jl}) \quad \forall C \in \mathcal{C}_{i,k}^+, i \in [1, n_A - 1], k \in [1, n_B - 1] \quad (3.13)$$

$$\tilde{y}_{ikjl} = \tilde{y}_{jlik} \quad \forall i \in [1, n_A - 1], k \in [1, n_B - 1], j \in [i + 1, n_A], l \in [k + 1, n_B] \quad (3.14)$$

$$\sum_{i,k \in C} x_{ik} \leq 1 \quad \forall C \in \mathcal{C} \quad (3.15)$$

$$\tilde{\mathbf{y}} \geq 0 \quad (3.16)$$

$$\mathbf{x} \text{ binary.} \quad (3.17)$$

Objective function (3.11) maximizes structural scores $s(A_{ij}, B_{kl})$ for inter-residue distance pairs as well as sequence scores $s(A_{ii}, B_{kk})$ for aligned residues. Throughout the model, a set C denotes mutually contradicting alignment graph nodes, *i.e.*, residue pairs that can not be aligned at the same time because they are not sequential or do

not represent a one-to-one mapping. The set \mathcal{C} contains all such sets C . It represents all decreasing paths in the alignment graph (*cf.* Section 3.3). Constraints (3.15) then denote that the solution must be a sequential one-to-one mapping by disallowing any set C of aligned residues that contradict, *i.e.*, any decreasing path. These constraints are identical to constraints (3.8) of our general model (3.4)–(3.10) from Section 3.4.

Constraints (3.12) link directed distance pairs to their respective aligned residues, *i.e.*, alignment graph edges to alignment graph nodes. If i is not aligned with k and x_{ik} is thus zero, no distance pairs that align i with k , represented by \vec{y}_{ijkl} , can be in the solution; The right-hand side of (3.12) is forced to zero. Then again, if i is aligned with k , only distances that do not contradict can be in the solution at the same time. Constraints (3.12) summarize constraints (3.5) and (3.6) of model (3.4)–(3.10) from Section 3.4. These two previous types of constraints do not need to be stated individually anymore since we no longer impose on alignment graph edges ($i.k, j.l$) and their corresponding variables y_{ijkl} that tail $i.k$ needs to be smaller than head $j.l$.

Constraints (3.13) are the so-called activation constraints for distance pairs with negative score. They are described in detail in the previous and in the next section. These constraints denote that if two alignment graph nodes $i.k$ and $j.l$ are activated in the solution, then also the alignment graph edge between them must be in the solution. If the edge score is positive, this will be imposed by the objective function, since taking the edge will increase the overall score. If the edge score is negative, activation of the respective edge needs to be enforced explicitly. Constraints (3.13) are identical to constraints (3.7) in model (3.4)–(3.10) from Section 3.4.

Equality constraints (3.14) result from the variable splitting and link two symmetric distance pairs, represented by two alignment graph edges.

Lagrangian relaxation. There are various scoring schemes for structure alignment for which structural scores are exclusively positive, for example CMO or PAUL scorings; see Chapter 4 for details. If the scoring function also assigns negative scores to distance pairs, constraints (3.13) are in our model. In the following Section 3.5.3 we will devise a Lagrangian relaxation for this case. In this section, we assume that structural scores $s(\cdot, \cdot)$ are greater than or equal to zero. We can thus omit the activation constraints (3.13) from model (3.11)–(3.17).

Caprara *et al.* relax equality constraints (3.14) in a Lagrangian fashion. For each equality constraint and hence for each symmetric distance pair (A_{ij}, B_{kl}) and (A_{ji}, B_{lk}) we have one Lagrangian multiplier $\lambda_{ijkl} \in \mathbb{R}$ with $i < j$ and $k < l$. For convenience, we use additional multipliers λ_{jlik} , which are simply defined as $-\lambda_{ijkl}$. The Lagrangian relaxation is then given by

$$\text{LR}(\lambda) = \max_{\mathbf{x}, \vec{y}} \sum_{i=1}^{n_A} \sum_{\substack{j=1 \\ j \neq i}}^{n_A} \sum_{k=1}^{n_B} \sum_{\substack{l=1 \\ l \neq k}}^{n_B} [s(A_{ij}, B_{kl}) + \lambda_{ijkl}] \vec{y}_{ijkl} + \sum_{i=1}^{n_A} \sum_{k=1}^{n_B} s(A_{ii}, B_{kk}) x_{ik} \quad (3.18)$$

s.t. (3.12), (3.15), (3.16), (3.17). (3.19)

For any λ , the score of the optimal solution of the relaxed problem (3.18)–(3.19) is an upper bound on the optimal score of the original problem (3.11)–(3.17). We aim to

Chapter 3. Mathematical models and structure alignment algorithms

find the values of the Lagrangian multipliers λ which correspond to the lowest, *i.e.*, tightest upper bound by minimizing over λ ,

$$\min_{\lambda} \text{LR}(\lambda) \quad (3.20)$$

$$\text{s.t. (3.12), (3.15), (3.16), (3.17).} \quad (3.21)$$

This can be achieved by subgradient optimization: The relaxed problem (3.18)–(3.19) is solved for particular values of multipliers λ and the multipliers are adjusted based on the obtained solution. This is described in the following paragraphs.

Solving the relaxed problem. The relaxed problem (3.18)–(3.19) can be solved in time $O(n_A^2 n_B^2)$ if all structural scores $s(A_{ij} B_{kl})$ are positive. In this section, we will devise a double dynamic programming approach for this task, see Figure 3.9. Such double dynamic programming has been used before for structure alignment, for example in SSAP [108] (see Section 1.2.2). It has been summarized in Section 2.7.

When the equality constraints are relaxed, the problem can be decomposed into two individual problems. We call these problems local and global, respectively. The so-called *global problem* is given by

$$\text{LR}(\lambda) = \max_{\mathbf{x}} \sum_{i=1}^{n_A} \sum_{k=1}^{n_B} [s(A_{ii}, B_{kk}) + p_{ik}(\lambda)] x_{ik} \quad (3.22)$$

$$\text{s.t. } \sum_{i,k \in C} x_{ik} \leq 1 \quad \forall C \in \mathcal{C} \quad (3.23)$$

$$\mathbf{x} \text{ binary.} \quad (3.24)$$

Its solution is the sequence alignment of maximum score based on sequence scores $s(A_{ii}, B_{kk})$ and structural profits p_{ik} for aligning residue i with residue k (see Figure 3.9 b)). The upper level dynamic programming score for residues i and k is then given by

$$c_{ik}(\lambda) = s(A_{ii}, B_{kk}) + p_{ik}(\lambda). \quad (3.25)$$

The optimal alignment can be computed by dynamic programming using the Needleman-Wunsch algorithm [81]. Gap costs can be used if needed. For affine gap costs, dynamic programming according to Gotoh's algorithm can be used [30]. Both algorithms are described in Section 2.7.

The profits p_{ik} for aligning residue i with residue k are obtained by solving a so-called *local problem* for alignment graph node $i.k$. In this local problem, only alignment graph edges with tail $i.k$ are considered. The profit is the solution of

$$p_{ik}(\lambda) = \max_{\vec{y}} \sum_{\substack{j=1 \\ j \neq i}}^{n_A} \sum_{\substack{l=1 \\ l \neq k}}^{n_B} [s(A_{ij}, B_{kl}) + \lambda_{ikjl}] \vec{y}_{ikjl} \quad (3.26)$$

$$\text{s.t. } 1 \geq \sum_{j,l \in C} \vec{y}_{ikjl} \quad \forall C \in \mathcal{C} \quad (3.27)$$

$$\vec{y} \geq 0. \quad (3.28)$$

3.5. Computing lower and upper bounds

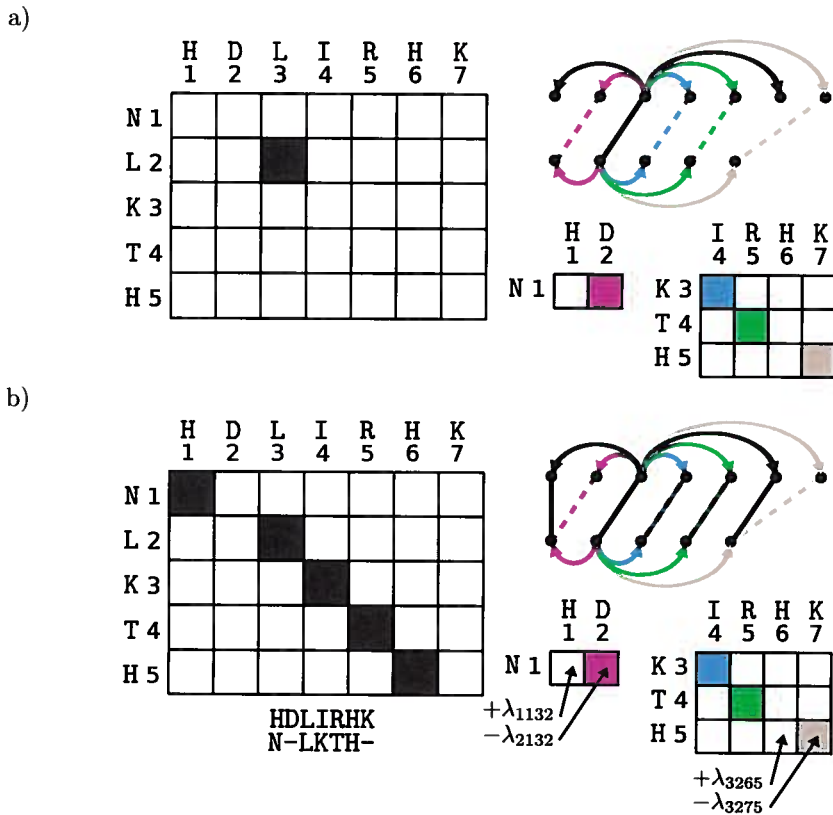


Figure 3.9: Double dynamic programming. a) Left, the upper level DP matrix. Right, profit computation via two lower level DP matrices, demonstrated for the upper level DP cell that aligns residues 3 and 2. One profit DP matrix aligns on the left, the other on the right of residue pair (3, 2). Scores of lower level DP cells are given by a scoring function, *e.g.*, PAUL scoring (4.18). Each colored cell and its score corresponds to a pair of inter-residue distances that is realized in the profit computation. The score of an upper level DP cell consists of this profit (computed via the two lower level DPs) and a score for the corresponding residue pair. b) An alignment, visualized by black DP cells and black lines that denote which residues are aligned. It contains two gaps. Choices that have been made in the lower level DPs during profit computation do not agree with the upper level DP and are penalized by Lagrangian multipliers λ . Steps a) and b) are repeated iteratively.

Chapter 3. Mathematical models and structure alignment algorithms

A local problem (3.26)–(3.28) can again be solved simply by a sequence alignment algorithm, *i.e.* by dynamic programming. We assign to each alignment graph node $j.l$ with $j < i$ and $k < l$ or $j > i$ and $k > l$, respectively, the score of alignment graph edge \vec{y}_{ikjl} , *i.e.*, $s(A_{ij}, B_{kl}) + \lambda_{ikjl}$. The lower level dynamic programming score for aligning residue j with residue l given the alignment of i with k is therefore given by

$$c_{jl}^{ik}(\lambda) = s(A_{ij}, B_{kl}) + \lambda_{ikjl}. \quad (3.29)$$

Then, we compute the best set of nodes in the left neighborhood $V^-(i.k)$ and the right neighborhood $V^+(i.k)$ of $i.k$ using dynamic programming as described in Section 2.7. The sum of the maximum profit obtained in $V^-(i.k)$ and $V^+(i.k)$ is the profit p_{ik} . The profit computation in terms of the lower level dynamic program is displayed in Figure 3.9 a) and in terms of the alignment graph in Figure 3.10.

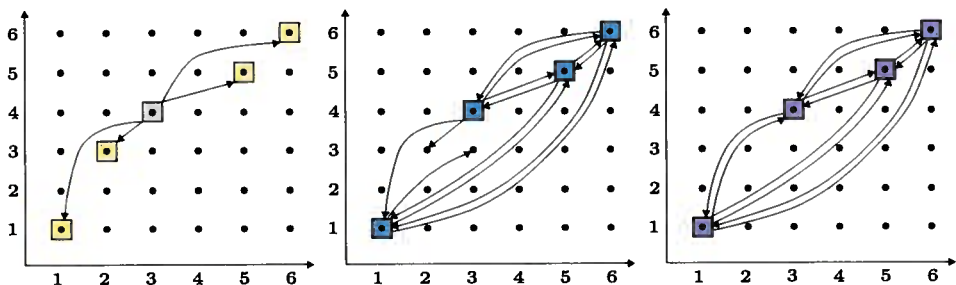


Figure 3.10: Assume each edge has a score of 1 and node scores are zero. Left: Profit computation p_{43} for alignment graph node 4.3; $p_{43} = 4$. Center: Solution of the relaxed problem with score $LR(\lambda) = 13$, which is an upper bound UB on the optimal score. Three equality constraints are violated: $\vec{y}_{1143} \neq \vec{y}_{4311}$, $\vec{y}_{3243} \neq \vec{y}_{4332}$ and $\vec{y}_{1133} \neq \vec{y}_{3311}$. The Lagrangian multipliers of the missing edges (1.1, 4.3), (3.2, 4.3) and (3.3, 1.1) will be increased, and those of their reverted partners (4.3, 1.1), (4.3, 3.2) and (1.1, 3.3) decreased. Right: The feasible solution that can be deduced from the relaxed solution comprises the alignment graph nodes from the relaxed solution together with all induced alignment graph edges. Its score is 12 and constitutes a lower bound.

We denote the solutions of the local problems (3.26)–(3.28) by \hat{y}_{ikjl} for $i = 1, \dots, n_A$ and $k = 1, \dots, n_B$ and the solution of the global problem (3.22)–(3.24) by \bar{x}_{ik} . The solution \bar{x}, \bar{y} of the relaxation (3.18)–(3.19) is then the alignment graph nodes described by \bar{x}_{ik} together with their outgoing edges described by $\bar{y}_{ikjl} = \bar{x}_{ik} \hat{y}_{ikjl}$.

In general, a solution of the relaxed problem is not a feasible structure alignment (see Figure 3.9 b) and Figure 3.10, center). Nonetheless, from a solution of the relaxed problem a feasible structure alignment can be deduced by simply taking the global solution \bar{x}_{ik} together with all its induced alignment graph edges, *i.e.*, induced distance pairs (see Figure 3.10, right). The score of this structure alignment provides a lower bound on the optimal score of the original problem. If lower and upper bound coincide, an optimal structure alignment has been found.

3.5. Computing lower and upper bounds

Dynamic programming is used for solving the local problems (3.26)–(3.28) and the global problem (3.22)–(3.24). Section 2.7 gives the precise DP recursions for linear and affine gap costs. Based on these recursions, we establish the complexity of solving the relaxed problem.

Theorem 2 (Complexity of solving the Lagrangian relaxed problem). *The Lagrangian relaxed problem (3.18)–(3.19) can be solved in time $O(n_A^2 n_B^2)$.*

Proof. Problem (3.18)–(3.19) can be solved by first solving the local problem (3.26)–(3.28) for any residue pair i from protein A and k from protein B and subsequently solving the global problem (3.22)–(3.24). We therefore need to compute $n_A n_B$ local problems and one global problem. Local as well as global problem can be solved by dynamic programming in time $O(n_A n_B)$ as described in Section 2.7. Therefore, we use scores c_{ik} according to (3.25) for the global, upper level dynamic program and scores c_{ji}^{ik} according to (3.29) for the local, lower level dynamic programs. Since $n_A n_B + 1$ DP matrices have to be computed, the complexity of $O(n_A^2 n_B^2)$ follows. \square

Note that the complexity can be reduced by omitting residues or inter-residue distances. In the case of omitting residues, the global problem can be solved in $O(|V|)$. Further, each alignment graph edge corresponds to exactly one lower level DP cell of one local problem. In the case of omitting inter-residue distances, all lower level DPs together are thus solved in $O(|E|)$. The overall time complexity is then $O(|V| + |E|)$. This is for example the case for contact map overlap [4].

The situation becomes more difficult if we do not omit residues or inter-residue distances entirely, but only certain pairs of residues or pairs of distances. As long as there is at least one distance pair or residue pair left for a given distance or residue, the lower and upper level dynamic programming matrices keep the same dimension and computation time does not change. In such a case we can improve time complexity by switching from dynamic programming to algorithms for sparse dynamic programming. One such algorithm is based on computing the heaviest increasing subsequence [50] after a problem reformulation as described in [69]. We can then solve the local problems in $O(|E| \log |E|)$. If many residue pairs or distance pairs are omitted, but little or no residues or distances are entirely disregarded, this can be faster than conventional dynamic programming. We find that for our practical problem instances this is not the case.

Subgradient optimization. In practice, instead of finding optimal Lagrangian multipliers λ^* , we solve the Lagrangian dual (3.20)–(3.21) only approximately using subgradient optimization. Let $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ be the solution of the relaxed problem for given multipliers λ . In each iteration, new values for λ are determined, the Lagrangian relaxed problem is solved for them and the solution $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ is used to update the multipliers based on their previous value. Initially multipliers are set to zero, *i.e.*, $\lambda^0 = 0$. The

Lagrangian multipliers in iteration $t + 1$ are then defined by

$$\lambda_{ikjl}^{t+1} := \begin{cases} \lambda_{ikjl}^t & \text{if } \bar{y}_{ikjl} - \bar{y}_{jlik} = 0 \\ \max(\lambda_{ikjl}^t - \gamma, -s(A_{ij}, B_{kl})) & \text{if } \bar{y}_{ikjl} - \bar{y}_{jlik} = 1 \\ \min(\lambda_{ikjl}^t + \gamma, s(A_{ij}, B_{kl})) & \text{if } \bar{y}_{ikjl} - \bar{y}_{jlik} = -1 \end{cases}, \quad (3.30)$$

where $i < j$ and $k < l$. Note that in scheme (3.30) only such multipliers λ_{ikjl} are updated for which the equality constraint $\bar{y}_{ikjl} = \bar{y}_{jlik}$ is violated. The effect of the Lagrangian multipliers is then the redistribution of the overall structural score between an alignment graph edge and its reverted partner, while keeping the sum always at $2s(A_{ij}, B_{kl})$. By this redistribution, the violation of equality constraints (3.14) is minimized. The most extreme redistribution which the Lagrangian multipliers can provoke is that the alignment graph edge in one direction has a profit of $2s(A_{ij}, B_{kl})$ while the reverted alignment graph edge has a profit of zero. The multipliers cannot be changed furthermore, which is guaranteed by update scheme (3.30).

The stepsize depends on the best lower and upper bound found so far, LB and UB , and is defined by

$$\gamma = \mu \frac{UB - LB}{\sum_{ikjl} (\bar{y}_{ikjl} - \bar{y}_{jlik})}. \quad (3.31)$$

Parameter μ is initially set to 1 and halved if no improved upper bound has been found within a certain number of iterations.

3.5.3 An alternative Lagrangian relaxation

In this section we specify certain classes of decreasing paths C in the alignment graph. We use these decreasing paths in order to construct a model for distance matrix alignment that is a special case of the general model (3.4)–(3.10) from Section 3.4. We then devise a Lagrangian relaxation for this model which differs from the relaxation of the previous section, but results in a similar relaxed problem which is again solved via dynamic programming.

Both, model and Lagrangian relaxation, are generalizations of a model and relaxation of Andonov *et al.* for contact map alignment [4]. The advantage of the model of Andonov *et al.* over the model of Caprara *et al.* of the last section is the smaller number of Lagrangian multipliers. If all inter-residue distances are considered, variable splitting leads to $O(n_A^2 n_B^2)$ multipliers, whereas the relaxation presented in this section only has $O(n_A^2 n_B)$ multipliers ($n_A \geq n_B$). The alternative model and relaxation allows to fit medium-size proteins into 24 Gb of memory, whereas this is currently infeasible for the relaxation presented in the last section. For this reason the formulation of Andonov *et al.* is clearly preferable for those cases of distance matrix alignment that take into consideration all inter-residue distances, *e.g.*, when using scoring schemes like those of DALI or MATRAS, see Section 4.

We extend the model and relaxation of Andonov *et al.* [4] by a new type of constraint that is needed for pairs of distances with negative score. These constraints are then relaxed in a Lagrangian fashion, additional to constraints that were already relaxed

previously. Note that precisely this extension allows the approach to be applied for the most general case of distance matrix alignment which may use any distance matrix-based scoring function.

Mathematical model. We specify the following sets of nodes that mutually contradict because they form a decreasing path: edge_{ik} , $\text{row}_{ik}(j)$ and $\text{col}_{ik}(l)$. Sets $\text{row}_{ik}(j)$ and $\text{col}_{ik}(l)$ are defined for $j \neq i$ and $l \neq k$. If $j < i$ or $l < k$, they represent tails of edges with head at $i.k$ and if $j > i$ or $l > k$ they represent heads of edges with tail at $i.k$. There are many ways of constructing special types of decreasing paths, the one we use is introduced in [4] and illustrated in Figure 3.11. Formally

$$\text{edge}_{ik} = \{i.0, i.1, \dots, i.k\} \cup \{0.k, 1.k, \dots, i-1.k\}, \quad (3.32)$$

$$\text{row}_{ik}(j) = \begin{cases} \{j.0, j.1, \dots, j.k-1\} \cup \\ \{j+1.0, j+2.0, \dots, i-1, 0\} \cup \\ \{0.k-1, 1.k-1, \dots, j-1.k-1\} & j < i \\ \{j.k+1, j.k+2, \dots, j.n_B\} \cup \\ \{j+1.k+1, j+2.k+1, \dots, n_A.k+1\} \cup \\ \{i+1.n_B, i+2.n_B, \dots, j-1.n+B\} & \text{otherwise} \end{cases}, \quad (3.33)$$

$$\text{col}_{ik}(l) = \begin{cases} \{0.l, 1.l, \dots, i-1.l\} \cup \\ \{i-1.0, i-1.1, \dots, i-1.l-1\} \cup \\ \{0.l+1, 0.l+2, \dots, 0.k-1\} & l < k \\ \{i+1.l, i+2.l, \dots, n_A.l\} \cup \\ \{n_A.k+1, n_A.k+2, \dots, n_A.l-1\} \cup \\ \{i+1.l+1, i+1.l+2, \dots, i+1.n_B\} & \text{otherwise} \end{cases}. \quad (3.34)$$

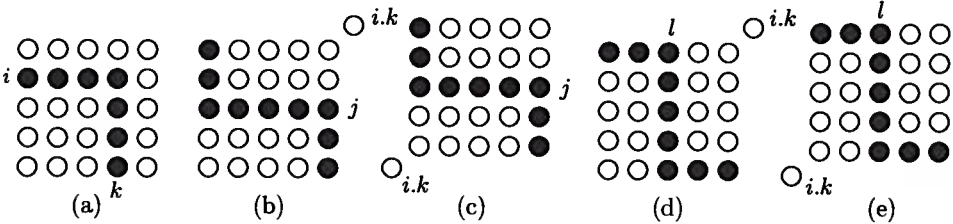


Figure 3.11: The black nodes are an illustration of edge_{ik} , $\text{row}_{ik}(j)$ and $\text{col}_{ik}(l)$ in the alignment graph. For sets $\text{row}_{ik}(j)$ and $\text{col}_{ik}(l)$, if $j < i$ or $l < k$, the colored nodes are sets of mutually exclusive tails of contradicting edges with common head $i.k$. If $j > i$ or $l > k$, the colored nodes are sets of mutually exclusive heads of contradicting edges with common tail $i.k$. a) edge_{ik} b) $\text{row}_{ik}(j)$ for $j < i$. c) $\text{row}_{ik}(j)$ for $j > i$. d) $\text{col}_{ik}(l)$ for $l < k$. e) $\text{col}_{ik}(l)$ for $l > k$.

In the following model we specify constraints for every node $i.k$ by using decreasing paths edge_{ik} , $\text{row}_{ik}(j)$ for $j = 1, \dots, n_A$ and $\text{col}_{ik}(l)$ for $l = 1, \dots, n_B$. This results in

Chapter 3. Mathematical models and structure alignment algorithms

a polynomial number of inequalities. The model uses then these constraints instead of the exponential number of constraints for *all* decreasing paths that were used in the general model (3.4)–(3.10) of Section 3.4. This is sufficient because the constraints enforce that no pair of contradicting nodes can be in the solution at the same time.

We assign binary variables x_{ik} to alignment graph nodes. They indicate whether residue i is aligned with residue k , in which case $x_{ik} = 1$. An alignment graph edge between nodes $i.k$ and $j.l$ is described by a binary variable y_{ikjl} with $i < j$ and $k < l$. It denotes whether distance A_{ij} from the first protein is aligned with distance B_{kl} in the second protein, in which case $y_{ikjl} = 1$. Andonov *et al.* do not apply variable splitting, as described in the last section. Instead, they relax certain types of constraints directly. Therefore, as in the general model (3.4)–(3.10) of Section 3.4, two symmetric distance pairs (A_{ij}, B_{kl}) and (A_{ji}, B_{lk}) are described by one variable y_{ikjl} and the respective score of such a variable is $2s(A_{ij}, B_{kl})$.

The ILP with a polynomial number of constraints is given by

$$\max_{\mathbf{x}, \mathbf{y}} \sum_{i=1}^{n_A-1} \sum_{j=i+1}^{n_A} \sum_{k=1}^{n_B-1} \sum_{l=k+1}^{n_B} 2s(A_{ij}, B_{kl})y_{ikjl} + \sum_{i=1}^{n_A} \sum_{k=1}^{n_B} s(A_{ii}, B_{kk})x_{ik} \quad (3.35)$$

$$\text{s.t. } x_{ik} \geq \sum_{r.s \in \text{row}_{ik}(j)} y_{ikrs} \quad j \in [i+1, n_A], i \in [1, n_A-1], k \in [1, n_B-1] \quad (3.36)$$

$$x_{ik} \geq \sum_{r.s \in \text{col}_{ik}(l)} y_{ikrs} \quad l \in [k+1, n_B], i \in [1, n_A-1], k \in [1, n_B-1] \quad (3.37)$$

$$x_{ik} \geq \sum_{r.s \in \text{row}_{ik}(j)} y_{rsik} \quad j \in [1, i-1], i \in [2, n_A], k \in [2, n_B] \quad (3.38)$$

$$x_{ik} \geq \sum_{r.s \in \text{col}_{ik}(l)} y_{rsik} \quad l \in [1, k-1], i \in [2, n_A], k \in [2, n_B] \quad (3.39)$$

$$x_{ik} \leq \sum_{\substack{r.s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} (y_{rsik} - x_{rs}) + 1 \quad j \in [1, i-1], i \in [2, n_A], k \in [2, n_B] \quad (3.40)$$

$$\sum_{j.l \in \text{edge}_{ik}} x_{jl} \leq 1 \quad i \in [1, n_A], k \in [1, n_B] \quad (3.41)$$

$$\mathbf{y} \geq 0 \quad (3.42)$$

$$\mathbf{x} \text{ binary.} \quad (3.43)$$

A solution of ILP (3.35)–(3.43) is a structure alignment of maximum overall distance matrix alignment score. All constraints except (3.40) are generalizations of the constraints established by Andonov *et al.* for the CMO model.

Constraints (3.36) and (3.37) denote that an edge can only be in the solution if its tail node is activated and if the heads of edges with common tail $i.k$ do not contradict. They are special cases of constraints (3.5) of the general distance matrix alignment model (3.4)–(3.10) of Section 3.4. Constraints (3.38) and (3.39) denote the reverse situation: an edge can only be in the solution if its head is activated and the tails of edges with common head $i.k$ do not contradict. They refer to constraints (3.6) in model (3.4)–(3.10).

3.5. Computing lower and upper bounds

Different from the model for CMO, the distance matrix alignment model has additional constraints (3.40). These describe that an edge has to be activated if its head and tail are activated. This is important since for various scoring functions (*e.g.* DALI or MATRAS scoring, see Chapter 4), edge scores can be negative. In such a case the remaining constraints allow to omit these edges. Constraints (3.40) are derived from the simple constraints $x_{ik} + x_{jl} - y_{jlik} \leq 1$ for all y_{jlik} with score less than or equal to zero. In these simple constraints, the term $x_{jl} - y_{jlik}$ can be lifted to $\sum_{r,s \in \text{row}_{ik}(j)} (x_{rs} - y_{rsik})$, since each tail $r.s$ has, according to constraints (3.36) and (3.37), no outgoing edges with contradicting heads $i.k$. Constraints (3.40) are a special case of constraints (3.7) in the general model of Section 3.4.

Constraints (3.41) correspond to constraints (3.8) in the general distance matrix alignment model of Section 3.4. Instead of having in our model a constraint for every decreasing path $C \in \mathcal{C}$, we use only constraints for the decreasing paths edge_{ik} for every node $i.k$. Still, any pair of contradicting nodes is disallowed by at least one constraint, and the solution must therefore represent a strictly increasing path in the alignment graph.

Note that in model (3.35)–(3.43), we have for every class of constraints a polynomial number of inequalities, whereas in the general model (3.4)–(3.10) of Section 3.4, every constraint class comprises an exponential number of inequalities. The constraints of model (3.35)–(3.43) represent a subset of the constraints of model (3.4)–(3.10).

Lagrangian relaxation. We relax constraints (3.38), (3.39) and (3.40). This means that now an edge can be in the solution even if its head is not activated (constraints (3.38) and (3.39)) as well as that an edge can be omitted even if its tail and head are activated (constraints (3.40)). Since constraints (3.36) and (3.37) are not relaxed, still any edge needs to have an activated tail, and since constraints (3.41) dictate that activated nodes lie on a strictly increasing path, the tails of edges can not contradict in spite of relaxing constraints (3.38) and (3.39). The solution of the relaxed problem is the following: A strictly increasing path of activated nodes, in which each activated node picks outgoing edges of maximum overall score. The heads of these outgoing edges are not necessarily activated. See Figure 3.12, center, for a visualization. The relaxation can then be strengthened by constraints

$$\sum_{\substack{r,s \in \text{edge}_{jl} \\ r > i, s > k}} y_{ikrs} \leq 1, \quad i \in [1, n_A], k \in [1, n_B], j \in [i + 1, n_A], l \in [k + 1, n_B], \quad (3.44)$$

which denote that the heads of outgoing edges picked by each node $i.k$ must (although still not necessarily activated) form a strictly increasing path. Note that constraints (3.44) were implied by constraints (3.41) together with inequalities (3.38) and (3.39), but are not implied anymore now that we relax the two latter constraints.

The relaxed problem is then given by

$$\begin{aligned}
 \text{LR}(\lambda) = & \max_{\mathbf{x}, \mathbf{y}} \sum_{i=1}^{n_A-1} \sum_{j=i+1}^{n_A} \sum_{k=1}^{n_B-1} \sum_{l=k+1}^{n_B} 2s(A_{ij}, B_{kl})y_{ikjl} + \sum_{i=1}^{n_A} \sum_{k=1}^{n_B} s(A_{ii}, B_{kk})x_{ik} \\
 & + \sum_{\substack{i,k \\ j \in [1, i-1]}} \lambda_{ikj}^h \left(x_{ik} - \sum_{r,s \in \text{row}_{ik}(j)} y_{rsik} \right) + \sum_{\substack{i,k \\ l \in [1, k-1]}} \lambda_{ikl}^v \left(x_{ik} - \sum_{r,s \in \text{col}_{ik}(l)} y_{rsik} \right) \\
 & + \sum_{\substack{i,k \\ j \in [1, i-1]}} \lambda_{ikj}^a \left(1 - x_{ik} + \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} (y_{rsik} - x_{rs}) \right) \tag{3.45} \\
 \text{s.t. } & \text{(3.36), (3.37), (3.41), (3.42), (3.43), (3.44)}. \tag{3.46}
 \end{aligned}$$

All λ are greater than or equal to 0. In objective function (3.45), λ_{ikj}^h denotes the multipliers for constraints (3.38), λ_{ikl}^v for constraints (3.39) and λ_{ikj}^a for constraints (3.40). The number of Lagrangian multipliers for constraints (3.38) and (3.40) are each $\frac{1}{2}n_A(n_A-1)(n_B-1)$ and for constraints (3.39) are $\frac{1}{2}n_B(n_B-1)(n_A-1)$. Let $n_A \geq n_B$, then this can be summarized to a cubic number $O(n_A^2 n_B)$ of multipliers. Note that the relaxation of the previous section used more Lagrangian multipliers, $O(n_A^2 n_B^2)$.

The Lagrangian dual problem determines the multipliers such that the upper bound $\text{LR}(\lambda)$ is minimized. It is given by

$$\begin{aligned}
 \min_{\lambda} \text{LR}(\lambda) & \tag{3.47} \\
 \text{s.t. } & \text{(3.36), (3.37), (3.41), (3.42), (3.43), (3.44)}. \tag{3.48}
 \end{aligned}$$

Solving the relaxation by double dynamic programming. The relaxed problem (3.45)–(3.46) can be solved by double dynamic programming, *i.e.*, dynamic programming on two levels, as described in Section 2.7. Similar to the Lagrangian approach described in the last section, we solve for this purpose for each alignment graph node a local problem and afterwards one global problem. In the local problems, we compute for each alignment graph node $i.k$ the best set of outgoing edges with their heads on a strictly increasing path. We call the sum of the corresponding edge weights the structural profit p_{ik} . Its computation via a lower level dynamic program is analogous to the profit computation for the right neighborhood of $i.k$ that has been described in the last section. In the global problem, we assign to each node its profit plus the node score and compute the strictly increasing path of maximum overall weight. This again can be done by dynamic programming, now on an upper level. The relaxed problem is therefore solved in the same way as the relaxed problem of the previous section. The difference lies in the scores that are used during double dynamic programming.

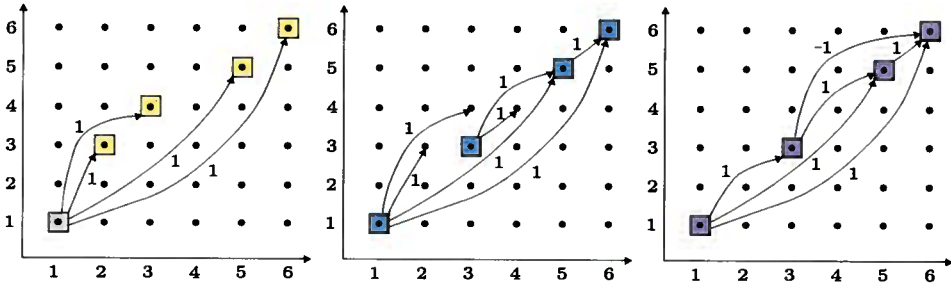


Figure 3.12: Visualization of local profit computation, the solution of the relaxed problem and the feasible solution. The edge scores are given next to the edges. In this example, they are 1 or -1 and the node scores are zero. Left: Node 1.1 picks its best set of outgoing edges, which are the edges maximizing the profit of this node. Here, $p_{11} = 4$. The corresponding strictly increasing path is colored yellow. Center: The solution of the relaxed problem. It is composed of the strictly increasing path that is the solution of the global problem, colored in blue, together with the outgoing edges that these nodes picked in their respective local problem. The relaxed solution maximizes the sum of profits. Its score is $LR(\lambda) = 7$ and an upper bound UB on the optimal score. For a few edges in the solution of the relaxed problem, the heads are not activated, *e.g.* for edge (1.1, 3.2). Also, for nodes in the solution, the induced edge is missing, *e.g.* in the relaxed solution, there is no edge between nodes 3.3 and 6.6. Right: The feasible solution that can be deduced from the relaxed solution. It is composed of the nodes that are activated in the relaxed solution together with all induced edges. Its score $LB = 4$ is a lower bound on the optimal score.

The *global problem* is given by

$$\text{LR}(\lambda) = \max_{\mathbf{x}} \sum_{i=1}^{n_A} \sum_{k=1}^{n_B} \left[p_{ik}(\lambda) + s(A_{ii}, B_{kk}) + \sum_{j \in [1, i-1]} \lambda_{ikj}^h + \sum_{l \in [1, k-1]} \lambda_{ikl}^v \right. \\ \left. - \sum_{j \in [1, i-1]} \lambda_{ikj}^a - \sum_{\substack{r, s, j \\ i, k \in \text{row}_{rs}(j) \\ r > i, s > k, r > j \\ s(A_{ir}, B_{ks}) \leq 0}} \lambda_{rsj}^a \right] x_{ik} \quad (3.49)$$

$$\text{s.t.} \quad \sum_{j, l \in \text{edge}_{ik}} x_{jl} \leq 1 \quad i \in [1, n_A], k \in [1, n_B] \quad (3.50)$$

$$\mathbf{x} \text{ binary.} \quad (3.51)$$

The constraints (3.50) express that a solution $\bar{\mathbf{x}}$ to the global problem should form a strictly increasing path. In the model of the previous section, this was formulated with an exponential number of constraints, here, alternatively only a polynomial number of constraints (3.50) is used. The profit p_{ik} for aligning residues i and k in objective function (3.49) depends on the multipliers λ . It is the solution of the corresponding *local problem*

$$p_{ik}(\lambda) = \max_{\mathbf{y}} \sum_{j=i+1}^{n_A} \sum_{l=k+1}^{n_B} [2s(A_{ij}, B_{kl}) - \lambda_{jli}^h - \lambda_{jlk}^v + \lambda_{jli}^a] y_{ikjl} \quad (3.52)$$

$$\text{s.t.} \quad \sum_{\substack{r, s \in \text{edge}_{jl} \\ r > i, s > k}} y_{ikrs} \leq 1, \quad j \in [i+1, n_A], l \in [k+1, n_B] \quad (3.53)$$

$$1 \geq \sum_{r, s \in \text{row}_{ik}(j)} y_{ikrs} \quad j \in [i+1, n_A] \quad (3.54)$$

$$1 \geq \sum_{r, s \in \text{col}_{ik}(l)} y_{ikrs} \quad l \in [k+1, n_B] \quad (3.55)$$

$$\mathbf{y} \geq 0. \quad (3.56)$$

Global and local problem can be solved by dynamic programming, analogous to the approach presented in the last section. The differences are that now we have Lagrangian multipliers on the edges as well as on the nodes, *i.e.*, in the local and global problem. Further, we need to compute dynamic programs only for the right neighborhood of node $i.k$, not, as in the relaxation of the previous section, for the left and right neighborhood.

The edge and node scores, *i.e.*, scores in the lower and upper level dynamic programs, correspond to the coefficients of the y_{ikjl} and x_{ik} variables in the local and global problem. Figure 3.13 visualizes how the Lagrangian multipliers redistribute score between nodes and incident edges in the case of a violated constraint. In the local problems, we associate to each edge $(i.k, j.l)$ and corresponding lower level DP cell (j, l) the weight

$$c_{jl}^{ik}(\lambda) = \begin{cases} 2s(A_{ij}, B_{kl}) - \lambda_{jli}^h - \lambda_{jlk}^v + \lambda_{jli}^a & \text{if } s(A_{ij}, B_{kl}) \leq 0 \\ 2s(A_{ij}, B_{kl}) - \lambda_{jli}^h - \lambda_{jlk}^v & \text{otherwise} \end{cases} \quad (3.57)$$

3.5. Computing lower and upper bounds

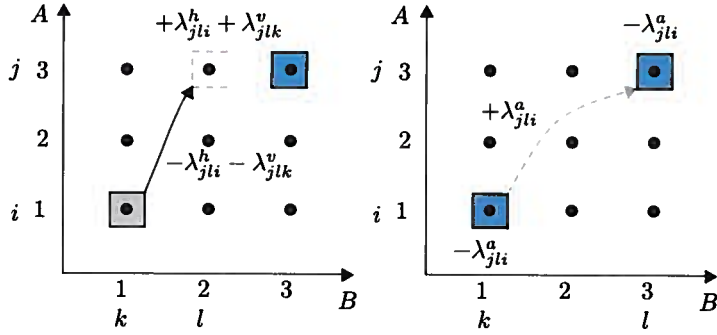


Figure 3.13: An example of the redistribution of score between nodes and edges in the case of violated constraints. Left: Constraints (3.38) and (3.39) are violated, since the head of edge (1.1, 3.2) is not activated and thus $x_{32} \not\geq y_{1132}$. Lagrangian multipliers will increase the weight of node 3.2 and decrease the weight of edge (1.1, 3.2). Right: Constraint (3.40) is violated, since the edge between the activated nodes is not activated and thus $x_{11} + x_{33} - y_{1133} \not\leq 1$. The multipliers for the activation constraints will decrease the weights of the nodes 1.1 and 3.3 and increase the weight of the incident edge.

Once profits $p_{ik}(\lambda)$ have been computed by solving the local problem for each node $i.k$, we solve the global problem. In the global problem, the node weights of $i.k$ and corresponding upper level DP scores of cell (i, k) are given by

$$c_{ik}(\lambda) = p_{ik}(\lambda) + s(A_{ii}, B_{kk}) + \sum_{j \in [1, i-1]} \lambda_{ikj}^h + \sum_{l \in [1, k-1]} \lambda_{ikl}^v - \sum_{j \in [1, i-1]} \lambda_{ikj}^a - \sum_{\substack{r, s, j \\ i.k \in \text{row}_{rs}(j) \\ r > i, s > k, r > j \\ s(A_{ir}, B_{ks}) \leq 0}} \lambda_{rsj}^a. \quad (3.58)$$

The solution of relaxation (3.45)–(3.46) has objective function value $LR(\lambda)$, which is an upper bound for the objective value of the original problem. Let \hat{y}_{ikjl} be the solutions of the local problems (3.52)–(3.56) for $i = 1, \dots, n_A$ and $k = 1, \dots, n_B$ and \bar{x}_{ik} the solution of the global problem (3.49)–(3.51). The solution \bar{x}, \bar{y} of the relaxation (3.45)–(3.46) then comprises the set of nodes that solve the global problem, \bar{x}_{ik} , together with the set of edges composing the solutions of the local problems for those tail nodes that are in the solution of the global problem, thus $\bar{y}_{ikjl} = \bar{x}_{ik} \hat{y}_{ikjl}$.

Nodes corresponding to \bar{x} together with their induced edges represent a feasible solution for the original problem, *i.e.*, a structure alignment. The objective function value of this feasible solution constitutes a lower bound LB . This is visualized in Figure 3.12, right.

Theorem 3 (Complexity of solving the Lagrangian relaxed problem). *The relaxed problem (3.45)–(3.46) can be solved in time $O(n_A^2 n_B^2)$, where n_A and n_B are the protein lengths.*

Proof. The proof is analogous to the proof of complexity for solving the relaxed problem from the previous section. The Lagrangian relaxed problem (3.45)–(3.46) can be solved

Chapter 3. Mathematical models and structure alignment algorithms

by computing a local problem (3.52)–(3.56) for each pair of residues i from protein A and k from protein B . The obtained profits p_{ik} are then used to compute one global problem (3.49)–(3.51). For each local problem we have to compute a dynamic program of the size of the right neighborhood of node $i.k$, which is at most $(n_A - 1)(n_B - 1)$. For solving the global problem we also compute one dynamic program in $O(n_A n_B)$. Therefore, from $n_A n_B + 1$ such calculations, we obtain an overall complexity of $O(n_A^2 n_B^2)$. \square

Note that the relaxation of this section can be solved exactly twice as fast as the relaxation of the previous section. The reason is that for profit computation it only considers the right neighborhood of a node, whereas the relaxation of the previous section considers right and left neighborhood. Further, in analogy to the relaxed problem of the previous section the time complexity can be reduced to $O(|V| + |E|)$ when omitting residues or distances. In the case of omitting only *pairs* of residues and distances, the complexity can be reduced by using methods for sparse dynamic programming, as described in the last section.

Updating Lagrangian multipliers. In order to find the tightest upper bound we need to solve the Lagrangian dual (3.47)–(3.48) in which $\text{LR}(\lambda)$ is the the objective function value of the optimal solution $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ of the relaxed problem for a given λ . For any multipliers λ , $\text{LR}(\lambda)$ constitutes an upper bound for the original problem (3.35)–(3.43). The solution of the Lagrangian dual is then the vector λ^* for which this upper bound is minimized. For finding good multipliers, we use the subgradient descent method (*cf.* Section 2.4). It is an iterative method in which at iteration t , given the current multiplier vector λ^t , a step is taken along a subgradient of $\text{LR}(\lambda^t)$; then, if necessary, the resulting point is projected onto the non-negative orthant. Formally, the Lagrangian multipliers are adjusted as follows,

$$\lambda^{t+1} = \max\{0, \lambda^t - \theta^t g^t\}, \quad (3.59)$$

with stepsize

$$\theta^t = \frac{\alpha(\text{LR}(\lambda^t) - LB)}{\sum[(g^h)^t]^2 + \sum[(g^v)^t]^2 + \sum[(g^a)^t]^2}. \quad (3.60)$$

Here, $\text{LR}(\lambda^t)$ is the objective function value of the Lagrangian relaxation for multipliers λ^t and LB is the best lower bound found so far. Further, g^h, g^v and g^a are the gradients for the multipliers of constraints (3.38), (3.39) and (3.40), respectively. Let $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ be the solution of the relaxed problem (3.45)–(3.46) for λ^t . For updating the multipliers, the corresponding gradients are then computed as follows

$$g_{ikj}^h = \bar{x}_{ik} - \sum_{r,s \in \text{row}_{ik}(j)} \bar{y}_{rsik} \in \{-1, 0, 1\}, \quad (3.61)$$

$$g_{ikl}^v = \bar{x}_{ik} - \sum_{r,s \in \text{col}_{ik}(l)} \bar{y}_{rsik} \in \{-1, 0, 1\}, \quad (3.62)$$

$$g_{ikj}^a = 1 - \bar{x}_{ik} + \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} (\bar{y}_{rsik} - \bar{x}_{rs}) \in \{-1, 0, 1\}. \quad (3.63)$$

There are only three situations in which gradient g_{ikj}^a for the activation constraints is non-zero. The first situation is $g_{ikj}^a = 1$, if

$$\bar{x}_{ik} = 0, \quad \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} \bar{y}_{rsik} = 0, \quad \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} \bar{x}_{rs} = 0. \quad (3.64)$$

In this case λ_{ikj}^a is decreased. Since $\lambda_{ikj}^a \geq 0$, we have to check for this situation only for indices i , k and j with non-zero multiplier λ_{ikj}^a in the current iteration.

The gradient g_{ikj}^a is also equal to 1, if

$$\bar{x}_{ik} = 0, \quad \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} \bar{y}_{rsik} = 1, \quad \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} \bar{x}_{rs} = 1. \quad (3.65)$$

This second situation is analogous to the situation detected by identifying violated constraints (3.38): The tail node of an edge is activated, but its head is not. Multiplier λ_{ikj}^a is decreased, as in the first situation, which is only possible if it is currently greater than zero.

Furthermore, in the third situation $g_{ikj}^a = -1$, if

$$\bar{x}_{ik} = 1, \quad \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} \bar{y}_{rsik} = 0, \quad \sum_{\substack{r,s \in \text{row}_{ik}(j) \\ s(A_{ri}, B_{sk}) \leq 0}} \bar{x}_{rs} = 1. \quad (3.66)$$

Here, we have to identify pairs of nodes that are both activated and are connected by an edge which is not activated. Since we have to check only all pairs of the n nodes in the solution, this can be done quickly in time $O(n^2)$.

3.6 Variable elimination

Independent of the mathematical model and algorithm that is used, the challenge for any exact distance matrix alignment approach is the large number of $\binom{n_A}{2} \binom{n_B}{2}$ pairs of inter-residue distances. Because all of them need to be considered explicitly, large instances currently do not fit into memory. This problem can be tackled by using a scoring function that disregards many distances, *e.g.*, CMO or PAUL scoring (see Chapter 4). Then the variables that correspond to the respective distance pairs can be eliminated from the mathematical model and do not need to be considered in the corresponding algorithms.

Usually, scoring functions assign non-zero scores to all distance pairs, *cf.* Chapter 4. We therefore developed a preprocessing which eliminates x and y variables from the mathematical models which provably can not be part of the optimal solution. The algorithms that were introduced in the last sections then have to consider less residue- and distance pairs, *i.e.*, alignment graph nodes and edges.

For an efficient variable elimination, a good feasible solution is needed. It is provided by a heuristic algorithm. For structure alignment, often powerful heuristics are available. Then, for each alignment graph node, an upper bound on the optimal structural

Chapter 3. Mathematical models and structure alignment algorithms

alignment score *including* this node is computed. If this upper bound is less than the score of a known feasible solution, the node cannot be part of the optimal solution and will be discarded. The corresponding variable will be eliminated from the ILP.

The benefit of preprocessing is twofold: first, the computation time decreases if only a subset of nodes in the alignment graph needs to be considered. Second, less memory is needed, because together with a node also all its incoming and outgoing edges can be discarded. Since each alignment graph node can be handled one after another, there is no memory issue during preprocessing.

The efficiency of variable elimination depends on the quality of the lower bound that is used, which in turn depends on the performance of the heuristic algorithm that computes it. Further, efficiency depends on the structural similarity of the compared proteins. For very similar proteins, we can eliminate almost all nodes that do not belong to the optimal solution, whereas for remotely similar proteins sometimes almost no nodes are discarded. Successful preprocessing allows to compute alignments of protein pairs that otherwise would not fit into memory.

In order to compute overestimations of the structural alignment score for a particular alignment graph node, we use double dynamic programming as described in Section 2.7. In the first, local dynamic program, we compute a profit p_{ik} for any node $i.k \in V$, which corresponds to the largest value that this node can add to the overall score. To compute p_{ik} we focus only on edges with head or tail $i.k$, and consider only those with score $s(A_{ij}, B_{kl}) > 0$. These scores are then assigned to the corresponding nodes $j.l \in V^-(i.k)$ and $V^+(i.k)$, respectively. The weight p_{ik}^- of the heaviest strictly increasing path in $V^-(i.k)$ (p_{ik}^+ for $V^+(i.k)$, respectively) can be computed in time proportional to the size of the rectangle $V^-(i.k)$ ($V^+(i.k)$, respectively) using dynamic programming. Finally, we set $p_{ik} = p_{ik}^- + p_{ik}^+$.

The second level, global dynamic program consists in overestimating the score of any solution containing a given node $m.n$, *i.e.*, in which $x_{mn} = 1$. For this purpose we associate profits p_{ik} to all nodes, and using them as weights we compute the heaviest strictly increasing path in $V^-(m.n)$ and $V^+(m.n)$, respectively. Adding the weights of these two paths, as well as p_{mn} , we obtain an overestimated score of the best structure alignment including node $m.n$. If this overestimation falls below the score of a known feasible solution, node $m.n$ and the corresponding variable x_{mn} together with all incoming and outgoing edges and their variables are eliminated from the problem description.

The efficiency of this variable elimination depends on the precision of the profit computation. The coarsest but also fastest method to potentially eliminate a node $m.n$ is the use of the previously computed profits p_{ik} , see Figure 3.14 a). We call it *rough node elimination*. The advantage is that the profits can be computed once for all nodes in the beginning.

Better upper bounds can be obtained by using profits $p_{ik}^{m.n}$ that are computed analogous to profits p_{ik} , but consider only nodes in $V^-(m.n) \cup V^+(m.n)$. This *accurate node elimination* gives tighter overestimations, but is more time-consuming. It is illustrated in Figure 3.14 b).

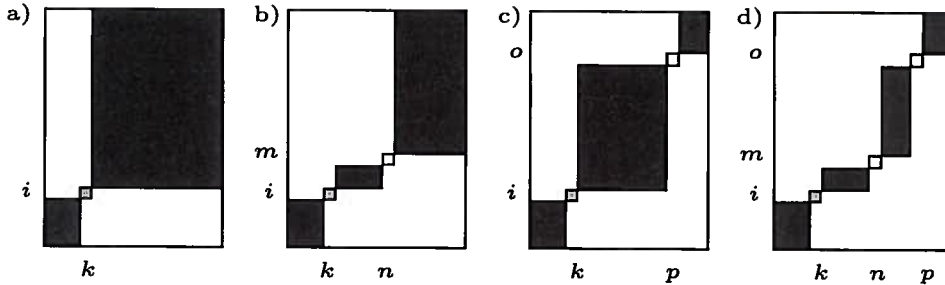


Figure 3.14: Different computations for profit p_{ik} of node $i.k$. In black the feasible nodes. a) Rough profit computation. In black $V^-(i.k)$ and $V^+(i.k)$. b) Accurate profit computation for eliminating node $m.n$ when $i.k < m.n$: p_{ik}^- stays unchanged, but p_{ik}^+ is recomputed by considering only nodes in $V^+(i.k) \cap (V^-(m.n) \cup V^+(m.n))$. c) The same accurate profit computation for eliminating node $o.p$. For eliminating edge $(m.n, o.p)$, we can take the minimum of the two cases b) and c). d) Most accurate profit computation for node $i.k$ for eliminating edge $(m.n, o.p)$.

Besides eliminating nodes, we can also eliminate edges. We implement two ways of accomplishing this. One way, the so-called *rough edge elimination* $(m.n, o.p)$, uses the same profits as for rough node elimination, see Figure 3.14 a). We can do better by re-using the profits $p_{i.k}^{m.n}$ and $p_{i.k}^{o.p}$ computed during the accurate elimination of nodes $(m.n)$ and $(o.p)$. Instead of p_{ik} , we use in this case updated profits \hat{p}_{ik} defined as $\hat{p}_{ik} = \min(p_{i.k}^{m.n}, p_{i.k}^{o.p})$. This is the minimum of the profits visualized in Figure 3.14 b) and c). If $p_{i.k}^{m.n}$ and $p_{i.k}^{o.p}$ are stored during the accurate node elimination, such an *accurate edge elimination* is then not more expensive than rough edge elimination. The most precise profits are obtained if we take the intersection of the neighborhoods of $i.k$, $m.n$ and $o.p$, see Figure 3.14 d). Because there is a large number of edges, recomputing profits for every single edge is in practice too time-consuming.

3.7 Branch-and-bound.

The ILP of Section 3.4 and its variants model the problem of finding a score-optimal distance matrix alignment. The corresponding algorithms from Sections 3.5.1, 3.5.2 and 3.5.3 for solving respective relaxations of the ILP provide lower and upper bounds on the optimal score. In some cases these bounds converge such that they coincide. Then a provably optimal solution has been found which solves both the relaxation and the original ILP. However, this does not necessarily happen. From theory we know that, even if we solve the Lagrangian dual of the ILPs to optimality, the optimal solution provides only an upper bound, since only weak duality applies for ILPs (*cf.* Section 2.4). Similar holds for the cutting plane approach which provides the optimal solution of the LP relaxation, but not of the ILP.

In practice we find that in the case of structurally very similar proteins, the gap between

Chapter 3. Mathematical models and structure alignment algorithms

lower and upper bound is closed. The less similar two proteins are, the larger becomes the gap. In our approaches from Sections 3.5.1, 3.5.2 and 3.5.3, bounds tend to converge quickly at the beginning of the computation and then more and more slowly. Therefore these algorithms for bound computation can be embedded into a branch-and-bound framework. In fact, only by doing so we obtain exact algorithms, since solving the relaxations alone does not necessarily return the optimal solution.

We apply a general branch-and-bound approach as introduced in Section 2.6 and summarized in Algorithm 3. Therefore, the only thing left to do is to specify how we split a problem into two or more distinct subproblems. The splitting rule that is summarized in this section is taken from [4].

We consider again the alignment graph and recall that a solution to the distance matrix alignment problem corresponds to a strictly increasing path of nodes. We can then specify a subproblem by tuples

$$(l_1, u_1), (l_2, u_2), \dots, (l_{n_B}, u_{n_B}) \text{ with } 1 \leq l_k, u_k \leq n_A, \quad (3.67)$$

which define the set of feasible alignment graph nodes. Here, (l_k, u_k) denotes that residue k in the second protein can be aligned with residues i in the first protein for $l_k \leq i \leq u_k$. We define F as the set of feasible nodes described by (3.67). We can now select one feasible alignment graph node $i.k$ and split the problem into two subproblems. In one, node $i.k$ may be included in the solution and in the other it is excluded from the solution. In order to accomplish this, the set F of feasible nodes is split into sets F_1 and F_2 with

$$F_1(j, l) = \{j.l \in F \mid j < i \text{ or } l > k \text{ or } j.l = i.k\} \quad (3.68)$$

$$F_2(j, l) = \{j.l \in F \mid j > i \text{ or } l < k\}. \quad (3.69)$$

For a visualization, refer to Figure 3.15. Note that the two subproblems overlap. If the best solution in F consists for example only of nodes $j.l$ with $j.l < i.k$ and/or nodes

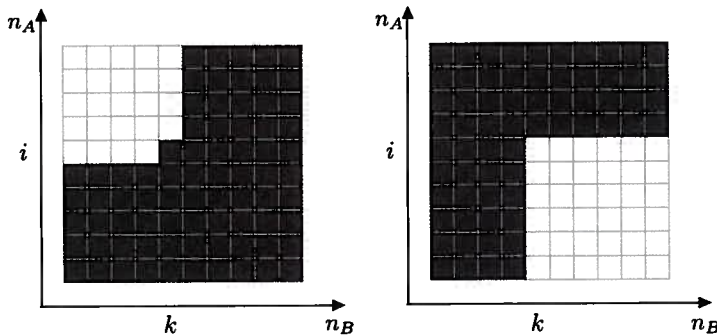


Figure 3.15: Splitting of a problem with $n_A = 10$ and $n_B = 10$ into two subproblems by fixing node $i.k = 6.5$ within the branch-and-bound algorithm. Set $F_1 = \{(1, 5), (1, 5), (1, 5), (1, 5), (1, 6), (1, 10), (1, 10), (1, 10), (1, 10), (1, 10)\}$ is displayed on the left, set $F_2 = \{(1, 10), (1, 10), (1, 10), (1, 10), (7, 10), (7, 10), (7, 10), (7, 10), (7, 10), (7, 10)\}$ on the right, both in black.

with $j.l > i.k$, then this solution will be present in both F_1 and F_2 . This property of overlapping subproblems also impedes an efficient divide-and-conquer approach. We experimented for example with dividing the problem into a large number of subproblems, but the more subproblems, the more unfavorable scales their number with the decrease of the size of the subproblems.

We pick a node $i.k$ for branching such that the number of feasible nodes in both subproblems is about the same, *i.e.*,

$$i.k = \operatorname{argmax}_{j,l} \min(|F_1(j,l)|, |F_2(j,l)|). \quad (3.70)$$

Andonov *et al.* found experimentally that splitting a problem into four subproblems instead of two speeds up the branch-and-bound algorithm significantly [4]. Therefore they apply the described splitting rule once to divide a problem into two subproblems and then apply it again to split the two subproblems.

3.8 Implementations

In this section we describe the implementations of the three structure alignment algorithms that were introduced in the previous sections: a cutting plane method and two Lagrangian approaches, one using variable splitting and another one dualizing constraints directly. The cutting plane method is implemented within the branch-and-cut framework of the software package CPLEX. The Lagrangian approach of Section 3.5.3 is implemented within a branch-and-bound framework. Both can use variable elimination according to Section 3.6 as preprocessing.

3.8.1 Branch-and-cut

When working on the cutting plane method of Section 3.5.1, we tried out several variants, including solving a less tight but polynomially-sized model, developing a cutting plane approach for the full model, combinations of the two models, several preprocessing techniques, and a divide-and-conquer approach. The most successful approach was a coarse variable elimination according to Section 3.6 followed by a cutting plane approach for constraints (3.5), (3.6) and (3.7) of model (3.4)–(3.10). Instead of generating cutting planes for inequalities (3.8), we restrict x -variables to be binary and use a polynomial number of constraints as initial ILP,

$$\sum_{j,l \in \text{edge}_{i,k}} x_{jl} \leq 1, \quad i \in [1, n_A], k \in [1, n_B]. \quad (3.71)$$

These constraints from [4] that have been introduced in Section 3.5.3 describe the set of strictly increasing paths. Cutting planes for activation constraints (3.7) are added only for nodes $i.k$ with $x_{ik} = 1$. The implementation of the cutting plane approach is summarized in Algorithm 4.

Algorithm 4 Cutting plane algorithm for solving the LP relaxation of model (3.4)–(3.10)

```

1:  $P \leftarrow$  (3.4) s.t. (3.9), (3.10), (3.71) //  $P$  is the initial problem
2: while True do
3:   Compute optimal solution  $\bar{x}, \bar{y}$  for  $P$ 
4:   if  $\bar{x}$  and  $\bar{y}$  integer then
5:     return  $\bar{x}, \bar{y}$  // Solution to ILP (3.4)–(3.10) found
6:   else
7:     for Residue pairs  $i$  and  $k$  do
8:       // Identify a cutting plane (3.5) for edges with tail  $i.k$ 
9:       Assign  $\bar{y}_{ikjl}$  as weight to all nodes  $j.l \in V^+(i.k)$ 
10:      Compute maximum weight decreasing path  $C$  in  $V^+(i.k)$ 
11:      if  $\sum_{j.l \in C} \bar{y}_{ikjl} > \bar{x}_{ik}$  then
12:         $P \leftarrow P \cup$  constraint  $\sum_{j.l \in C} y_{ikjl} \leq x_{ik}$ 
13:      end if
14:      // Identify a cutting plane (3.6) for edges with head  $i.k$ 
15:      Assign  $\bar{y}_{ikjl}$  as weight to all nodes  $j.l \in V^-(i.k)$ 
16:      Compute maximum weight decreasing path  $C$  in  $V^-(i.k)$ 
17:      if  $\sum_{j.l \in C} \bar{y}_{jlik} > \bar{x}_{ik}$  then
18:         $P \leftarrow P \cup$  constraint  $\sum_{j.l \in C} y_{jlik} \leq x_{ik}$ 
19:      end if
20:      if  $\bar{x}_{ik} = 1$  then
21:        // Identify a cutting plane (3.7) for edges with tail  $i.k$ 
22:        Assign  $\bar{y}_{ikjl} - \bar{x}_{jl}$  as weight to all nodes  $j.l \in V^+(i.k)$  with  $s(A_{ij}, B_{kl}) \leq 0$ 
23:        Compute maximum weight decreasing path  $C$  in  $V^+(i.k)$ 
24:        if  $\sum_{j.l \in C} (\bar{y}_{ikjl} - \bar{x}_{jl}) > \bar{x}_{ik}$  then
25:           $P \leftarrow P \cup$  constraint  $\sum_{j.l \in C} (y_{ikjl} - x_{jl}) \leq x_{ik}$ 
26:        end if
27:      end if
28:    end for
29:  end if
30:  if No cutting planes added then
31:    return  $\bar{x}, \bar{y}$  // Solution to LP relaxation of (3.4)–(3.10) found
32:  end if
33: end while

```

We use the optimization software package CPLEX (version 12.1) for a branch-and-cut algorithm. In order to set up and solve a CPLEX problem in C++, we use the CPLEX CONCERT technology as interface. CPLEX solves the LP relaxations of the problem and provides a branch-and-bound framework. Our cutting plane method according to Algorithm 4 is implemented in C++ within a CPLEX callback function. It adds constraints to the problem that are violated by the current solution of the LP relaxation.

We apply variable elimination according to Section 3.6 and generate only those variables that are not excluded by this preprocessing. The variable elimination is implemented in a separate, dedicated C++ class. The resulting structure alignment program uses the DALI scoring function.

3.8.2 PAUL: a Lagrangian approach using variable splitting

Our iterative double dynamic programming algorithm for computing bounds using the Lagrangian relaxation of Section 3.5.2 is summarized in Algorithm 5. We implemented it in the freely available software package PAUL [119] within the C++ software library PLANET LISA (<http://planet-lisa.net/>). PAUL computes many instances to optimality [116]. Nonetheless, it is not exact because it is not embedded into a branch-and-bound framework.

Algorithm 5 Iterative double dynamic programming for solving Lagrangian dual (3.20)–(3.21)

```

1:  $UB \leftarrow +\infty$  // Lowest upper bound found so far
2:  $LB \leftarrow -\infty$  // Highest lower bound found so far
3:  $\lambda \leftarrow 0$  // Lagrangian multipliers
4:  $\mathbf{x}^* \leftarrow \text{undef}$  // Best feasible solution found so far
5: while  $UB \neq LB$  and iteration or time limit not reached do
6:   // Solve Lagrangian relaxed problem (3.18)–(3.19) for current  $\lambda$ 
7:   for Residue pairs  $i$  and  $k$  do
8:     Compute solution  $\hat{y}_{ikjl}$  and optimal value  $p_{ik}$  of local problem (3.26)–(3.28)
9:   end for
10:  Compute solution  $\bar{\mathbf{x}}$  of global problem (3.22)–(3.24)
11:   $\bar{y}_{ikjl} \leftarrow \bar{x}_{ik} \hat{y}_{ikjl}$ 
12:  Determine the score of the feasible solution induced by  $\bar{\mathbf{x}}$ 
13:  // Update bounds
14:  if Objective value of  $\bar{\mathbf{x}}, \bar{\mathbf{y}}$  smaller than  $UB$  then
15:     $UB \leftarrow$  objective value of  $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ 
16:  end if
17:  if Feasible solution has higher score than the best feasible solution so far then
18:     $LB \leftarrow$  score of current feasible solution
19:     $\mathbf{x}^* \leftarrow \bar{\mathbf{x}}$ 
20:  end if
21:  Update Lagrangian multipliers  $\lambda$  according to (3.30)
22:  Update stepsize according to (3.31)
23: end while

```

Chapter 3. Mathematical models and structure alignment algorithms

PAUL uses a dedicated, carefully tuned scoring scheme that is presented in Section 4.1.3. Besides carefully evaluated default parameters (see Table 4.1), all scoring function parameters, gap penalties, amino acid substitution matrices, and the proportion of sequence to structural score can be chosen. The use of secondary structure and log-odds scores for filtering alignment graph nodes and edges is an additional, optional feature. By keeping the scoring adaptable, PAUL allows to incorporate additional information about the input proteins, which makes it a very flexible software tool.

PAUL supports different input formats, *e.g.* PDB files, lists of pre-selected distances or complete distance matrices, as well as different types of inter-residue distances (C_α , C_β , and all-atom, which is the minimum distance between any atoms of two residues). If a global alignment of two proteins is needed, we resort to TCOFFEE [82]. TCOFFEE computes a sequence-based alignment for those residues for which no structural correspondences could be determined.

3.8.3 Alternative Lagrangian approach and branch-and-bound

Our approach for solving the alternative Lagrangian relaxation of Section 3.5.3 is given in Algorithm 6. It is similar to the Lagrangian approach of Section 3.5.2 summarized in Algorithm 5. The difference lies in (i) considering only the right neighborhood in local problems (ii) node and edge scores in local and global problems and (iii) the computation of the gradients.

We implemented this Lagrangian approach within the branch-and-bound framework that is presented in Section 3.7. The implementation is in C++ and largely based on A_PURVA, a program for contact map alignment [4]. For contact maps, the efficiency of the solver of A_PURVA has been proven previously by extensive numerical computations [73, 70]. We extend this implementation to support negative structural scores. To this end, constraints (3.40) are relaxed in a Lagrangian fashion as described in Section 3.5.3 and the corresponding changes and extensions integrated into the A_PURVA code. Further, we implemented sequence scores, *i.e.*, scores on the alignment graph nodes, as well as affine gap costs using the Gotoh [30] algorithm, which is described in Section 2.7. In order to speed up the implementation for scoring schemes that use a large number of inter-residue distances, we parallelized the computation of the local problems as well as the identification of violated constraints using OpenMP. Optionally, the solver can thus compute on several processors in parallel.

Together with these changes that are needed for general distance matrix alignment, now any inter-residue distance-based scoring function can be passed to the Lagrangian solver. We implemented, besides the already available CMO scoring, the scoring schemes of PAUL, DALI, and MATRAS. The next chapter introduces them in detail. The re-implementation of PAUL is, because of the branch-and-bound and the smaller number of Lagrangian multipliers, experimentally more powerful than our initial implementation using the Lagrangian relaxation of Section 3.5.2. It is therefore the current, preferred version of the PAUL program. For the DALI scoring function, variable elimination as described in Section 3.6 can be used.

The implementations using DALI and MATRAS scoring are the first implementations

Algorithm 6 Iterative double dynamic programming for solving Lagrangian dual (3.47)–(3.48)

```

1:  $UB \leftarrow +\infty$  // Lowest upper bound found so far
2:  $LB \leftarrow -\infty$  // Highest lower bound found so far
3:  $\lambda \leftarrow 0$  // Lagrangian multipliers
4:  $\mathbf{x}^* \leftarrow \text{undef}$  // Best feasible solution found so far
5: while  $UB \neq LB$  and iteration or time limit not reached do
6:   // Solve Lagrangian relaxed problem (3.45)–(3.46) for current  $\lambda$ 
7:   for Residue pairs  $i$  and  $k$  do
8:     Compute solution  $\hat{y}_{ikjl}$  and optimal value  $p_{ik}$  of local problem (3.52)–(3.56)
9:   end for
10:  Compute solution  $\bar{\mathbf{x}}$  of global problem (3.49)–(3.51)
11:   $\bar{y}_{ikjl} \leftarrow \bar{x}_{ik}\hat{y}_{ikjl}$ 
12:  Determine the score of the feasible solution induced by  $\bar{\mathbf{x}}$ 
13:  // Update bounds
14:  if Objective value of  $\bar{\mathbf{x}}, \bar{\mathbf{y}}$  smaller than  $UB$  then
15:     $UB \leftarrow$  objective value of  $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ 
16:  end if
17:  if Feasible solution has higher score than the best feasible solution so far then
18:     $LB \leftarrow$  score of current feasible solution
19:     $\mathbf{x}^* \leftarrow \bar{\mathbf{x}}$ 
20:  end if
21:  Compute gradients (3.61), (3.62), (3.63)
22:  Update Lagrangian multipliers  $\lambda$  according to (3.59)
23:  Update stepsize according to (3.60)
24: end while

```

of exact algorithms for computing DALI and MATRAS alignments. We call them, in order to distinguish them from the respective heuristic programs, DALIX and MATRASX. CMO, PAUL, DALIX and MATRASX alignments can be computed online via our web server CSA at <http://csa.project.cwi.nl>. The binaries of all four programs are also available from this web site.

Scoring schemes and quality criteria for structure alignment

There are about as many scoring schemes as there are algorithms for protein structure alignment. Each method is particularly tailored towards optimizing its own scoring. According to our classification in Section 1.2, 3-dimensional scoring schemes are based on the superposition of the 3D coordinates of aligned residues. Scoring schemes that are 2-dimensional match pairwise structural information such as inter-residue distances. Finding a score-optimal structure alignment is in both cases for nearly all scoring schemes believed to be an *NP*-hard problem (*cf.* Section 1.2.2). Therefore, almost all structure alignment algorithms are heuristic. They confine the search space, *e.g.*, by considering fragments of several residues or by applying hierarchical approaches that, for example, align secondary structure elements in a first step. Several such heuristics are outlined in Section 1.2.2.

Programs that utilize 3D coordinate superposition are for example MAX-PAIRS [89], PROTDEFORM [93], MATT [75], CE [99], STRUCTAL [105], FATCAT [124], SHEBA [53], C-ALPHA MATCH (CA) [6], PPM [18], TM-ALIGN [129], LGA [125] and RASH [103]. Programs using inter-residue distances are for example DALI [41], MATRAS [57], SSAP [108] and VOROLIGN [12]. From comparative studies [101, 61, 74, 8] it follows that many of these algorithms are considerably accurate and competitive, and some are even quite fast. Nonetheless they are not capable to report whether the computed alignment is optimal according to the optimized scoring function. As a consequence, there is no way to compare different scoring schemes for structure alignment, because weak performance can be attributed either to the scoring function that is optimized or to the algorithm that is used.

In the following section we present several method-specific 2- and 3-dimensional scoring schemes for protein structure alignment, including our own, dedicated paul scoring scheme [116], as well as a few quality measures like RMSD that are generally applied. We stress here that it is only fair to assess structure alignment algorithms with respect to the native score that they optimize. Nonetheless, such a native score should be validated, for example, by assessing whether it reflects protein similarities as provided by gold-standard classifications or by evaluating whether it finds correct residue correspondences with respect to gold-standard structure alignments. These two types of

carefully validated references are introduced in Section 4.2. In this section, we also specify the corresponding data sets that were used for our computational results. The last section of this chapter deals with comparative structure alignment. Here, the comparison of structure alignments and scoring schemes is motivated and ways and means for such a comparison are demonstrated. The presented concepts have been implemented in our web server CSA (Comparative Structural Alignment) [117].

4.1 Scoring schemes

In this section we present several established scoring schemes for protein structure alignment. Some of them are generally used for assessing the quality of structure alignments, others are mainly used within the corresponding alignment program. We divide the scoring schemes into superposition-based, *i.e.*, 3-dimensional, and inter-residue distance-based, *i.e.*, 2-dimensional. Our focus lies on the inter-residue distance-based scorings, because these can be optimized by our exact structure alignment algorithms. For a comprehensive, complementary review on scoring schemes for structure alignment, refer to [36].

4.1.1 Superposition-based scoring schemes

Superposition-based scoring schemes use the distance between aligned residues after 3D superposition. They are often related to RMSD_c .

RMSD_c. The most widely used quality measure for structure alignment is the coordinate root-mean-square-deviation (RMSD_c), see also Section 1.2. It is defined as follows,

$$\text{RMSD}_c = \sqrt{\frac{1}{n} \sum_{i,k \text{ aligned}} (d_{i,T(k)})^2}, \quad (4.1)$$

where $T(k)$ are the 3D coordinates of residue k after superpositioning and n is the number of aligned residues ($n \geq 1$). The term $d_{i,T(k)}$ refers to the distance between the two aligned residues after rotation and translation. The smaller the RMSD_c is, the better the alignment. Shorter alignments typically have a smaller RMSD_c , with minimum zero if a single residue is aligned. Therefore, RMSD_c needs to be counter-balanced by alignment length and is thus not reasonably optimized alone. An alignment is better than another alignment with respect to RMSD_c only if the alignment length n is the same or larger and it has a smaller RMSD_c . If these two conditions are not met simultaneously, the two alignments cannot be compared in a sound way.

TM-score. The TM-score [128] is an additive score based on the distance between aligned residues and defined as

$$\text{TM-score} = \frac{1}{n_B} \sum_{i,k \text{ aligned}} \frac{1}{1 + \left(\frac{d_{i,T(k)}}{d_0(n_B)}\right)^2} \quad (4.2)$$

Chapter 4. Scoring schemes and quality criteria for structure alignment

with

$$d_0(n_B) = 1.24(n_B - 15)^{1/3} - 1.8. \quad (4.3)$$

Here, n_B is the number of residues in the target protein and $d_{i,T(k)}$ the distance between residue i and k after optimal superposition. The TM-score lies in the interval $(0, 1]$ and the larger it is, the better the structure alignment. According to TM-ALIGN [129], a program that aims at maximizing the TM-score, a score of less than 0.2 indicates that there is no similarity. A TM-score that exceeds 0.5 denotes that the structures share the same fold.

STRUCTAL. The STRUCTAL score is in spirit similar to the TM-score and also based on the distance $d_{i,T(k)}$ between aligned residues after optimal superposition T . It is defined as

$$\text{STRUCTAL} = \sum_{i,k \text{ aligned}} \frac{20}{1 + \frac{(d_{i,T(k)})^2}{5}} - 10n_g, \quad (4.4)$$

where n_g is the number of gaps. The STRUCTAL program [105] aims to maximize it using a heuristic algorithm. The optimal STRUCTAL score can be approximated up to any previously given, additive error $\epsilon > 0$ in time polynomial in the alignment length [62].

GDT_TS. The GDT_TS score [126] is mainly used for the comparison of protein structure predictions with experimentally solved structures, for example in the biannual CASP (Critical Assessment of techniques for protein Structure Prediction) competition [78]. GDT is defined as the maximum number of residues that, given a superposition, are closer than a given threshold. GDT_TS is then called the total GDT score, which is the average of the number of residues that can be superposed under a threshold of 1, 2, 4 and 8 Å. In protein structure prediction, no superposition has to be determined to evaluate GDT_TS, because the coordinates of the experimentally solved structure and of its model are both known. In structure alignment, in contrast, the alignment and corresponding superposition has to be determined. The LGA [125] and MAX-PAIRS [89] structure alignment programs aim to find an alignment that maximizes GDT.

RMSD100. RMSD100 is a normalization of RMSD_c with respect to the alignment length n ,

$$\text{RMSD100} = \frac{\text{RMSD}_c}{1 + \ln(\sqrt{\frac{n}{100}})}. \quad (4.5)$$

Other parameter values than 100 are also possible, putting more or less weight on RMSD_c versus alignment length. The smaller an alignment's RMSD100, the better it is.

SAS and GSAS. The SAS score and its global version GSAS are used by Kolodny *et al.* in their comparative study of structure alignment methods [61]. They also balance

RMSD_c versus alignment length n . SAS is defined as

$$\text{SAS} = 100 \frac{\text{RMSD}_c}{n}, \quad (4.6)$$

and GSAS as

$$\text{GSAS} = \begin{cases} 100 \frac{\text{RMSD}_c}{n-n_g} & \text{if } n_g < n \\ 99.9 & \text{otherwise} \end{cases}. \quad (4.7)$$

Here, n is the number of aligned residues and n_g the number of gaps in the alignment. Both scores should be minimized.

4.1.2 Inter-residue distance-based scoring schemes

Inter-residue distance-based scoring schemes are used to score distance matrix alignments (I, K) according to

$$S(I, K) = \sum_{o=1}^n \sum_{p=1}^n s(A_{i_o, i_p}, B_{k_o, k_p}) + S_{\text{gap}}(I, K), \quad (4.8)$$

see also Section 3.2. They use a scoring function $s(\cdot, \cdot)$ that assigns a structural and sequence score to distance pairs. Further, a function $S_{\text{gap}}(I, K)$ can penalize gaps in the alignment. Many algorithms exist that address the distance matrix alignment problem using a variety of scoring schemes. All of these scorings can be used in the mathematical models and algorithms that were presented in the previous chapter. In the following, we introduce several of them.

For distance-based scoring schemes the same caveat applies as mentioned previously for superposition-based scoring schemes: Alignments should not be judged using criteria other than the criterion that they have been optimized for. Although no qualitative conclusions should be drawn from comparisons using non-native scoring schemes, such comparisons are still useful to get insight into the characteristics of structure alignments computed under different scoring schemes, see for example Section 4.3.

RMSD_d. Analogous to RMSD_c, there is also an inter-residue distance-based root-mean-square-deviation, which we call RMSD_d. Just as RMSD_c, RMSD_d is not a score that can be optimized alone, but rather a quality criterion that is applied to compare alignments generated by optimizing other scoring schemes. It is defined as

$$\text{RMSD}_d = \sqrt{\frac{1}{m} \sum_{\substack{i,k \text{ aligned} \\ j,l \text{ aligned} \\ j>i, k>l}} (A_{ij} - B_{kl})^2}. \quad (4.9)$$

Here, A_{ij} and B_{kl} are the aligned distances and $m = \binom{n}{2}$ is the number of aligned distance pairs. RMSD_d is defined for $n \geq 2$ and should be minimized. It is meaningful only in conjunction with the number of aligned residues n , because if n decreases,

Chapter 4. Scoring schemes and quality criteria for structure alignment

RMSD_d typically also decreases. The minimum RMSD_d , for example, will usually be reached if a single distance pair is aligned. An alignment is better than another alignment with respect to RMSD_d if it aligns the same number or even more residues with a smaller RMSD_d (*cf.* RMSD_c).

CMO. Two residues are defined to be in contact if their Euclidean distance is less than a distance threshold d_t . The contact map overlap (CMO) is then the number of common contacts that are induced by an alignment, *i.e.*, the number of contacts whose endpoints are aligned. Many algorithms have been presented that aim at finding an alignment of maximum contact map overlap, *e.g.* A_PURVA [4], CMOS [122], MSVNS [85], AI-EIGEN [21], SADP [51] and BIMAL [52]. Formally, the scoring function is given by

$$s(A_{ij}, B_{kl}) = \begin{cases} 1 & A_{ij} \leq d_t \text{ and } B_{kl} \leq d_t \\ 0 & \text{otherwise} \end{cases}, \quad (4.10)$$

for $1 \leq i < j \leq n_A$ and $1 \leq k < l \leq n_B$ and some $d_t \in \mathbb{R}^+$. A typical value for the distance threshold d_t is 7.5 Å. The CMO scoring function is visualized in Figure 4.1.

DAST. Motivated by the fact that maximizing shared contacts often generates alignments with large RMSD, the DAST algorithm [69] focuses on local alignments with low RMSD_d . This corresponds to aligning cliques of distances with low mutual distance differences and can be expressed as

$$s(A_{ij}, B_{kl}) = \begin{cases} 1 & |A_{ij} - B_{kl}| \leq \tau \\ -\infty & \text{otherwise} \end{cases}, \quad (4.11)$$

for $1 \leq i < j \leq n_A$ and $1 \leq k < l \leq n_B$ and some $\tau \in \mathbb{R}^+$. The distance difference threshold τ lies usually between 2 and 4 Å. The DAST scoring function is visualized in Figure 4.1.

DALI. DALI [41] is a popular heuristic for protein structure alignment. Its scoring function [36, 41] corresponds to

$$s(A_{ij}, B_{kl}) = \begin{cases} \left(0.2 - \frac{|A_{ij} - B_{kl}|}{\frac{1}{2}(A_{ij} + B_{kl})}\right) e^{-\left(\frac{1}{2}(A_{ij} + B_{kl})/20\right)^2} & i \neq j \text{ and } k \neq l \\ 0.2 & \text{otherwise} \end{cases} \quad (4.12)$$

and is visualized in Figure 4.1. Based on the overall DALI score $S(A, B)$, the DALI z-score $Z(A, B)$ is computed according to the formula given in [44],

$$Z(A, B) = \frac{S(A, B) - m(L)}{0.5 \cdot m(L)}. \quad (4.13)$$

The term $m(L)$ for $L = \sqrt{n_A n_B}$ is the approximate mean score with

$$m(L) = 7.95 + 0.71L + 2.59 \cdot 10^{-4}L^2 - 1.92 \cdot 10^{-6}L^3 \quad (4.14)$$

and the denominator $0.5 \cdot m(L)$ estimates the average standard deviation. The z-score thus measures the significance of the detected structural similarity based on an experimentally determined background distribution of DALI scores.

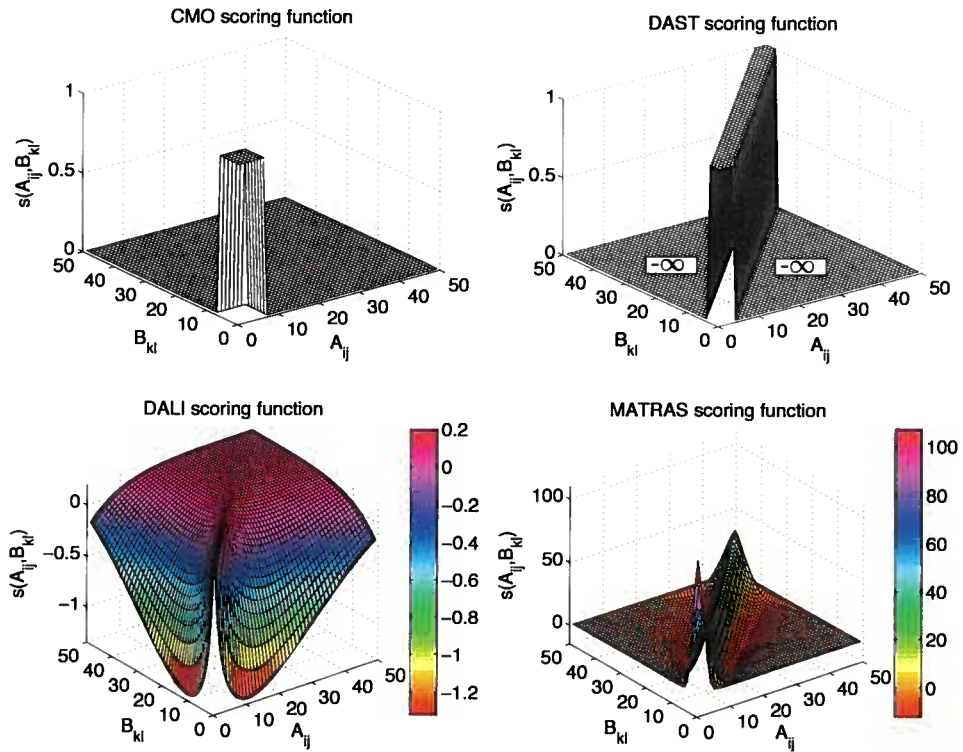


Figure 4.1: Different scoring functions that align inter-residue distance A_{ij} with inter-residue distance B_{kl} . Upper left: CMO scoring. Only distance pairs with both distances less than $d_t = 7.5 \text{ \AA}$ obtain a score of 1. Upper right: DAST scoring for distance difference threshold $\tau = 3.5 \text{ \AA}$. Distance pairs with distance difference $|A_{ij} - B_{kl}| \leq \tau$ obtain a score of 1. All other distance pairs are disallowed and thus obtain a score of $-\infty$. Lower left: DALI scoring. Lower right: MATRAS scoring for a sequence separation greater than 20.

MATRAS. Analogous to sequence substitution matrices, the structure alignment algorithm MATRAS [58] uses log-odds value matrices M as structural scores. A value $M_{[A_{ij}], [B_{kl}]}$ indicates the log-likelihood that distance A_{ij} is aligned to distance B_{kl} . A positive log-likelihood denotes that the corresponding distances are aligned more often than expected by chance. For MATRAS we therefore have

$$s(A_{ij}, B_{kl}) = M_{[A_{ij}], [B_{kl}]} \quad (4.15)$$

and function $S_{\text{gap}}(I, K)$ for affine gap costs. Log-odds matrices M are prepared for different sequence separations, where the sequence separation of a distance A_{ij} is defined as $|i - j|$. In order to assign a score to a distance pair we use the matrix that corresponds to the larger of the two sequence separations, *i.e.*, the one of $\max(|i - j|, |k - l|)$. The MATRAS scoring function for a sequence separation of more than 20 is visualized in Figure 4.1.

SSAP. SSAP [108] is an early, but still popular method for structure alignment. It uses a structural scoring function that can be expressed as

$$s(A_{ij}, B_{kl}) = \begin{cases} \frac{50}{(V_{ij}^A - V_{kl}^B)^2 + 2} & i \neq j \text{ and } k \neq l \\ -\sqrt{200 \min(n_A, n_B)} & \text{otherwise} \end{cases} \quad (4.16)$$

Here V_{ij}^A and V_{kl}^B are vectors between residue i and j in protein A and k and l in protein B , respectively. Using the difference of vectors for scoring, SSAP can account for the directionality of the compared inter-residue distances. Function $S_{\text{gap}}(I, K)$ assigns a penalty of -5 to each gap, independent of the gap's length.

VOROLIGN. VOROLIGN uses the Voronoi tessellation for determining close residues in 3D space. For these close residues that can be considered in contact, it uses two substitution matrices for structural scoring. These are a SSE substitution matrix M^S by Wallqvist *et al.* [111] and an amino acid substitution matrix M^A by Dosztányi and Torda [24], which is based on the energy contribution of amino acid mutations. Let $\text{sse}(\cdot)$ denote the SSE of a residue and $\text{aa}(\cdot)$ its amino acid type. The scoring function can then be cast into our framework by using

$$s(A_{ij}, B_{kl}) = \begin{cases} 0.71 \left(M_{\text{sse}(i), \text{sse}(k)}^S + M_{\text{sse}(j), \text{sse}(l)}^S \right) + \\ 0.58 \left(M_{\text{aa}(i), \text{aa}(k)}^A + M_{\text{aa}(j), \text{aa}(l)}^A \right) & i, j \text{ and } k, l \text{ in contact} \\ 0 & \text{otherwise} \end{cases} \quad (4.17)$$

Unaligned contacts are penalized by a cost $p_u = 5.04$ in the lower level DP. Function $S_{\text{gap}}(I, K)$ assigns a penalty of -22.5 for opening a gap and -6.5 for extending a gap in the upper level DP.

4.1.3 Dedicated scoring scheme: PAUL

Any function that scores pairs of inter-residue distances can be used to compute scores $s(A_{ij}, B_{kl})$ in objective functions (3.4), (3.11) or (3.35) of the mathematical models

from Section 3.4, 3.5.2 and 3.5.3. Examples are the 2-dimensional scoring functions presented earlier in this chapter. Schemes that assign only positive structural scores have the additional benefit that the so-called activation constraints (3.7) (or (3.13), (3.40), respectively) can be omitted, because aligning two distances will always increase the score and is therefore implied by the objective function. If the scheme assigns also negative structural scores, activation constraints are necessary. These constraints need to be considered explicitly, for example as described in Section 3.5.1 or Section 3.5.3.

The number of pairs of inter-residue distances is $\binom{n_A}{2}\binom{n_B}{2}$, where n_A and n_B are the lengths of the proteins. Because of this large number of distances, it is currently infeasible for many pairs of medium-size to large proteins to hold the corresponding instances in memory, let alone to solve them. Exceptions are similar proteins for which preprocessing can be applied successfully. Therefore, algorithms that align inter-residue distance matrices resort to techniques like aligning fragments of several residues or hierarchical alignment. In scoring schemes dedicated towards our algorithms such as CMO or PAUL scoring, we use a distance threshold d_t ; only distances smaller than the threshold are used. We hereby omit long inter-residue distances that are less significant for structural similarity than short distances. Trying to omit activation constraints that would need to be relaxed, a dedicated scoring function can be limited to assign only positive scores. Additionally, we try to omit all pairs of inter-residue distances that are rarely observed in biologically correct alignments.

A scoring function may use different types of inter-residue distances. We consider three: C_α , C_β and all-atom distances. All-atom distances are the minimum distance between any pair of atoms of two residues. Each type of inter-residue distance places a slightly different focus on the alignment. An alignment of C_α distances is based on similar protein backbone conformations, an alignment based on C_β distances takes into account side chain placement, and an alignment of all-atom distances highlights similar residue interactions in the two proteins. Because of the different nature and range of the three inter-residue distance types, individual scoring function parameters need to be determined.

The most effective technique to improve performance and speed of our algorithms is to keep the number of variables in our ILPs low. For each distance pair represented by a variable y_{ikjl} we store its score and adapt its Lagrangian multiplier. For each alignment graph node $i.k$ represented by a variable x_{ik} , we compute its profit, which has to be entirely recomputed in each Lagrangian iteration in which at least one of the multipliers of an outgoing alignment graph edge has been changed (*cf.* Figure 3.9 b)). Accordingly, there are two ways of filtering: Filtering distance pairs affects the lower level dynamic program, and filtering residue pairs the upper level dynamic program. If we forbid aligning i with k and therefore set variable x_{ik} to zero, we never have to compute the corresponding profit, *i.e.*, the lower level dynamic program.

We filter on both levels. On the upper level we use secondary structure information. Using DSSP [55], we assign to each residue its secondary structure type and forbid aligning α -helical residues (DSSP state “H”) with β -sheet residues (DSSP state “E”). On the lower level, we filter in two different ways. We omit pairs of distances for which at least one distance exceeds the distance threshold and distances that differ more than a distance difference threshold.

Chapter 4. Scoring schemes and quality criteria for structure alignment

Using only positive scores for pairs of distances leads to the problem that even aligning structurally dissimilar regions will increase the overall score. A poor global alignment can then yield a higher score than a shorter alignment of structurally highly conserved regions. We deal with this problem by using a global penalty $c \leq 0$ for aligning two residues. This penalty serves as a threshold on the structural score, *i.e.*, on the profit of an alignment graph node. Only if the profit exceeds this threshold, the overall score increases when the corresponding residues are aligned.

We use affine gap costs with gap open penalty ν and gap extension penalty $\xi = 0.25 \cdot \nu$ in our dedicated scoring scheme. Further, an optional sequence score can be added to each residue pair. This sequence score may be provided by any external amino acid substitution matrix. A scaling factor can be tuned in order to balance structure against sequence score. By default, sequence scores are not used.

We developed our own, dedicated scoring scheme based on a large scale parameter training. Since we implemented it in a software tool called PAUL, we call it PAUL scoring. Inspired by the rigid similarity introduced by Holm and Sander in their paper on DALI [41], we choose the following scoring function,

$$s(A_{ij}, B_{kl}) = \begin{cases} \max\{0, \theta - |A_{ij} - B_{kl}|\} & |A_{ij} - B_{kl}| \leq \Delta_t \text{ and } A_{ij} < d_t \text{ and } B_{kl} < d_t \\ 0 & \text{otherwise} \end{cases} \quad (4.18)$$

Here θ , the distance threshold d_t and the distance difference threshold $\Delta_t < \theta$ are constant, non-negative parameters. Scoring function (4.18) is visualized in Figure 4.2. PAUL scoring considers only distances less than the distance threshold d_t . If two distances differ more than Δ_t , aligning them does not increase the score. Since we use rather small inter-residue distances, a common maximum distance difference threshold Δ_t may be applied. A differential scoring of pairs of distances via θ ensures that the score of pairs of distances decreases the more two distances differ from each other. For $\theta \rightarrow \infty$, scoring function (4.18) assigns the same score to all pairs of distances and is therefore identical to contact map alignment of all distances that differ less than Δ_t .

We optimized the parameters of our PAUL scoring scheme for C_α , C_β and all-atom distances. For each distance type, the parameter values which led to the largest alignment accuracy on our training set are listed in Table 4.1. These are the default values for the PAUL program. Since largest accuracy was reached for C_β distances, PAUL by default uses C_β distances. The corresponding scoring function is displayed in Figure 4.2.

Parameter	C_α	C_β	All-atom
d_t	9.5	8.5	6
θ	7.1	9.8	5
δ_t	4	7	3.5
c	7.1	19.6	12.5
ν	9	23	15
ξ	2.25	5.75	3.75

Table 4.1: PAUL default parameters for C_α , C_β and all-atom distances.

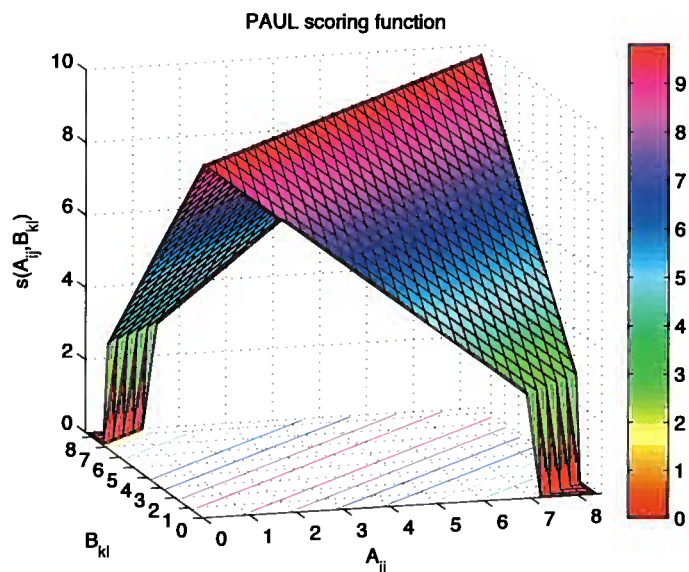


Figure 4.2: PAUL scoring function (4.18) for C_β inter-residue distances ($d_t = 8.5$, $\theta = 9.8$ and $\Delta_t = 7$).

4.2 Gold standards

There are gold standards for (i) hierarchical classification of protein structures based on pairwise similarities as well as for (ii) residue correspondences in single, pairwise alignments. Such gold-standards rely on careful manual inspection which takes into account all available biological information, for example sequence, structure, common functional motifs, *etc.*

4.2.1 Protein classification

The gold standards for protein classification are **SCOP** [80] and **CATH** [83]. In **SCOP** (Structural Classification Of Proteins), protein domains are manually classified into class, fold, superfamily and family. Main classes are α , β , α/β and $\alpha + \beta$ proteins, *cf.* Section 1.1. Proteins of the same fold share the same overall structural arrangement. Proteins of the same superfamily show structural similarities that indicate an evolutionary relationship. Proteins of the same family have additionally a sequence identity of more than 25%.

CATH classification is semi-automatic based on structure alignments computed by, among others, **CATHEDRAL** [92] and **SSAP** [108]. It divides proteins into four hierarchical levels: class (**C**), architecture (**A**), topology (**T**) and homologous superfamily (**H**). Protein classes are α -, β -, and mixed α and β proteins as well as a class for

Chapter 4. Scoring schemes and quality criteria for structure alignment

few secondary structures. Architecture describes the overall structural arrangement and is assigned manually. Superfamily and family are assigned automatically based on structural similarity score and sequence identity.

SCOP and **CATH** classifications are not the same. First, the domain assignments are different, such that the entities that are classified are not necessarily identical. Further, the hierarchic organization itself is different. Csaba *et al.* [19] have generated a gold standard classification, that comprises only domains that have been largely defined the same by **SCOP** and **CATH** and that are consistently classified. For details on this integrated data set, see Section 4.2.3.

Manual classification by experienced structural biologists like in **SCOP** and parts of **CATH** can be considered the state-of-truth: Such a classification takes into account diverse information and is largely independent of possible algorithmic errors or limitations. The disadvantage is that it can not keep up with the large number of experimentally solved protein structures.

SCOP classification is often used to validate scoring schemes for structure alignment. Either the raw structural alignment score, its normalization or its z-score is used to assess the similarity between protein pairs. These pairwise similarities should then reflect the **SCOP** gold-standard classification. For example, protein pairs whose similarity score exceeds a given threshold should belong to the same family (or superfamily or fold) whereas proteins whose similarity falls below the threshold should be from different families (or superfamilies or folds).

4.2.2 Reference alignments

Another way to assess scoring schemes or algorithms for structure alignment is the use of reference alignments. A reference alignment is a manually constructed or curated structure alignment. The residue correspondences of such a reference are considered the gold standard. Structure alignments can then be evaluated with respect to their agreement with the reference alignment. For such a comparison, the alignment accuracy is used. It is defined as the percentage of correctly aligned residue pairs.

4.2.3 Data sets

In this section we briefly introduce gold-standard data sets for classification and reference alignments which we use in our computational results.

SCOPCATH. **SCOPCATH** [19] is a benchmark of domains that have largely been defined the same in **SCOP** [80] version 1.75 and **CATH** [31, 83] version 3.2.0 and that are consistently classified in both hierarchies. A **SCOP** domain is mapped to a **CATH** domain if the intersection of the residue positions which they contain comprises at least 80% of the union of the domains' residue positions. Further, domains that share more than 50% sequence identity are clustered together and only one representative structure is retained in the data set. For the resulting 6759 domains, the **SCOPCATH** data set con-

tains all domain pairs for which **SCOP** and **CATH** agree on the classification. A domain pair is consistently classified if, according to the mapping of the two hierarchies, the structures are similar or dissimilar, respectively, on the same classification levels. The **SCOPCATH** data set can be downloaded from <http://www.bio.ifi.lmu.de/SCOPCath>.

SKOLNICK. **SKOLNICK** is a benchmark for classifying proteins into families. It consists of 40 protein chains of length between 97 and 255 residues that belong to five different **SCOP** families, see Table 4.2. This data set has been extensively used for the evaluation of CMO algorithms, *e.g.*, in [4, 14, 51, 85, 122, 69, 70]. All *vs.* all comparison of the 40 protein chains results in 780 alignments. There are 164 **SKOLNICK** structure alignment instances of protein pairs from the same family.

SCOP Family	Length	PDB IDs
CheY-related	120-130	1b00A, 1dbwA, 1natA, 1ntrA, 3chyA 1qmp(A,B,C,D), 4tmy(A,B)
Ferritin	158-191	1b71A, 1bcfA, 1dpsA, 1fhaA, 1ierA 1rcdA
Fungal ribonucleases	104	1rn1(A,B,C)
Plastocyanin/azurin-like	97-105	1bawA, 1byo(A,B), 1kdiA, 1ninA 1plaA, 2b3iA, 2pcyA, 2pltA
Triosephosphate isomerase (TIM)	243-256	1amkA, 1aw2A, 1b9bA, 1btmA, 1htiA 1tmhA, 1treA 1triA, 1ydvA, 3ypiA, 8timA

Table 4.2: The **SCOP** families, lengths and **PDB** ids of the protein structures from the **SKOLNICK** data set. The fifth character of the **PDB** ID refers to the protein chain.

HOMSTRAD. The **HOM**ologous **STR**ucture **Al**ignment **D**atabase [76] (**HOMSTRAD**) contains multiple structure alignments of families of homologous proteins. It has been established as gold-standard reference data set for sequence and structure alignment. **HOMSTRAD** alignments are initially constructed by structure alignment methods and subsequently manually inspected and annotated. Proteins of each family are clearly homologous, with either a sequence identity of more than 30% or a pronounced structural similarity. They are usually globally aligned. Therefore, by construction, **HOMSTRAD** consists of comparably easy structure alignment instances. The current version consists of 1032 families with 2 up to 41 members. The average protein length per family varies from 17 up to 885 residues, and the average sequence identity from 8 to 94%. In order to focus on the difficult pairwise structure alignment instances, we select from the 630 **HOMSTRAD** families with two members those with a low sequence identity.

SISY and RIPC. **SISY** and **RIPC** are two data sets of manually curated reference alignments. They have been constructed by Mayr *et al.* [74] and updated by Berbalk *et*

Chapter 4. Scoring schemes and quality criteria for structure alignment

Domain 1	Domain 2	Length 1	Length 2	SeqId	AlRes
d1an9a1	d1npxa1	247	198	19.19	11
d1ay9b_	d1b12a_	108	239	14.81	10
d1b5ta_	d1k87a2	275	351	15.64	8
d1cr1a_	d1edea_	534	310	17.1	3
d1d5fa_	d1nd7a_	350	374	34.57	6
d1dlia1	d1mv8a1	98	98	25.51	4
d1gbga_	d1ovwa_	214	398	16.82	3
d1ggga_	d1wdna_	220	223	99.55	220
d1gsaa1	d2hgsa1	122	102	15.69	5
d1hava_	d1kxfa_	216	159	21.38	4
d1hcyb2	d1ln1b1	263	307	11.03	4
d1jj7a_	d1lvga_	251	190	22.63	8
d1jwyb_	d1puja_	281	261	18.77	12
d1jwyb_	d1u01a2	281	212	15.09	11
d1kiaa_	d1nw5a_	275	270	19.26	12
d1l5ba_	d1l5ea_	101	101	100	101
d1nlsa_	d2bqpa_	237	228	21.05	6
d1nw5a_	d2adma1	270	215	24.19	13
d1qasa2	d1rsya_	126	135	21.43	75
d1qq5a_	d3chya_	245	128	17.97	3
d2adma1	d2hmyb_	215	327	16.74	12
d2bbma_	d4clna_	148	148	100	148

Table 4.4: The SCOP domains, lengths, sequence identity (SeqId) and number of aligned residues in the reference (AlRes) for the 22 structure alignment instances in the consolidated RIPC data set. Taken from [8].

corresponding programs and its alignments, besides CMO and PAUL, DALIX and MATRAS.

Numerous web servers are available that offer individual methods for computing structure alignments, *e.g.* [57, 77, 71, 40]. Comparative studies find that alignments produced by such different methods can differ considerably [74]. Therefore there are first attempts to integrate information from alignments generated by different structure alignment algorithms, although they are so far restricted to the context of protein classification [13, 7].

Optimality of structure alignments comes at the prize of higher run time, but is especially important when comparing alignments. A top-scoring, but biologically implausible alignment implies that the scoring scheme used is inadequate to detect the given structural relationship and a different scoring might be more advisable. In the case of pairwise structure alignment, in which primarily residue correspondences are of interest, and only secondarily the obtained similarity score, comparing alignments optimized with respect to different criteria thus brings additional insight.

In CSA, computed or uploaded alignments can be explored in terms of many inter-residue distance-, RMSD- and sequence-based scores and quality measures and with intuitive visualizations. This facilitates evaluating the agreement between alignments that maximize different established scoring schemes and helps detecting their strengths and weaknesses. Clear-case alignments on which various scoring schemes agree can be differentiated from ambiguous alignments for which it depends on the application which one is preferable. The user can then make educated decisions about the structural similarity of two proteins and, if necessary, post-process alignments by hand. In summary, comparisons thus bring new insight into the structural relationship of the protein pairs under investigation.

4.3.1 Implementation

Structure alignment algorithm. The exact solver used in CSA is an implementation of the Lagrangian approach of Section 3.5.3, for details see Section 3.8.3. The corresponding distance matrix alignment model (3.35)–(3.43) supports a generic scoring scheme with positive and negative structural scores, sequence scores and affine gap costs. Many different scoring functions are special cases of this general scheme, *cf.* Section 4.1.2. Currently, CSA supports the scoring schemes of CMO, PAUL, DALI and MATRAS. We choose CMO and PAUL scoring since they are tailored to our algorithm and DALI and MATRAS scoring because they are established and their programs and web servers [57, 40] are widely used. The performance of the Lagrangian approach strongly depends on the number of considered inter-residue distances [116]. It has been extensively evaluated for CMO [4], PAUL [116] and DALI scoring [114].

Web server implementation. The architecture of the web server is divided in a processing layer that computes (C++) and evaluates (Python) alignments and an output layer, which generates W3C-validated HTML websites, interacts with the user and displays all information (PHP and Javascript). The interface between the two layers is

Chapter 4. Scoring schemes and quality criteria for structure alignment

a MySQL database. The alignment engine for all our four currently supported scoring schemes is identical and implemented in C++ as a stand-alone program. The user may adjust the time limit of the computation. Furthermore, each scoring scheme has different parameters, for example, the use of C_α or C_β inter-residue distances.

Computed or user-uploaded (in FASTA, CLUSTALW or DALI format) alignments are read into a Python class and subsequently written to the MySQL data base. A second Python class handles the computation of different scores. It obtains the required structural information from the PDB files with the help of the Biopython package Bio.PDB [34]. Tasks related to superpositioning are also handled by this package. Visualizations of distance and distance difference matrices are generated using the Python Imaging Library.

The website functions have been implemented in separate modules, which makes it easy to integrate additional structure alignment methods. The modularity is illustrated by the use of a tab menu. All web server functions are extensively documented, which is denoted by a question mark next to the respective section titles or table headers. Additionally, a documentation puts instructions and explanations into context. Notably, we documented all structure alignment scoring schemes that are used within CSA and we provide the corresponding formulas and references. In the output layer, structures and their superpositions are visualized in Jmol (<http://www.jmol.org>) and images are generated using the PHP package pChart (<http://www.pchart.net/>).

We illustrate the functionality of CSA using two case studies which are accessible from its main page via the links “Example 1” and “Example 2”.

4.3.2 Case studies


Benefits of visualization and comparison. The first case study deals with two proteins from the SISY data set [74, 8], ubiquitin-binding protein CUE2 (PDB ID 1otr, chain A, 49 residues) and the CUE domain of activating signal cointegrator 1 complex subunit 2 (PDB ID 2di0, chain A, 71 residues). The proteins belong to the SISYPHUS [5] alignment AL00088995 of homologous proteins containing a CUE domain. The CUE domain is composed of a three helical bundle and it consists of 41 residues. It binds ubiquitin and is involved in protein degradation.

After specifying PDB IDs and chains on the main page of CSA, the user is redirected to the CSA evaluation environment. It is organized in tabs for the following tasks: overview on the protein structures, computing alignments using CMO, PAUL, DALI or MATRAS scoring, upload of external alignments, and the comparison of alignments.

The *Structures* tab lists PDB IDs, PDB file names, selected chains and their lengths and amino acid sequences. Links to the PDB [9] and to iHOP [38] are access points for additional information concerning the proteins and their function. Protein structures are visualized in Jmol. Their C_α and C_β distance matrices and contact maps are visualized. We compute CMO, PAUL, DALIX and MATRASX alignments using the default options, *i.e.*, with a time limit of 30 CPU s. The setup of all four result pages is identical. Exemplary, we consider the CMO alignment page; parts of it are displayed in Figure 4.3.

4.3. Web server for comparative structural alignment

Home
Documentation



Comparative Structural Alignment

CWI *lostra*

Structures
CMO ✓
PAUL ✓
DALX ✓
MATRASY ✓
SISY ✓
DALI ✓
CUSTOM3
CUSTOM4
Comparison

Optimized score (CMO) ? [back to top]

	Lower bound ?	Upper bound ?	Gap ?	Relative gap [%] ?
Score ?	125	125.000	0	0
Similarity ?	0.751	0.751	0	0

CMO alignment ? [back to top]

Show color-coded according to SSE
 Show color-coded according to residue pair contribution

```

1         2         3         4         5         6
-----|-----|-----|-----|-----|-----|-----|
  N  I  -  H  E  S  K  I  L  M  D  M  P  P  I  S  R  S  K  L  Q  V  H  L  E  N  N  D  L  L  T  G  L  L  K  E  N  N  D  -  -  N  I
  G  S  G  S  S  G  G  E  L  D  S  L  I  S  Q  V  K  D  L  L  P  L  L  E  G  E  F  L  L  A  C  E  Y  Y  H  Y  D  P  E  Q  V  I  N  N  I  L  E  E  R  L  P  T
-----|-----|-----|-----|-----|-----|-----|
              7
-----|-----|-----|-----|-----|-----|
Q  L  D  R  N  L  D  R  E  M  N

```

Low score (0.5) (5.5) High score

Score overview and related measures ? [back to top]

CMO score ?	125	DALI z-score ?	3.624
PAUL score ?	886.251	TM-score ?	0.451
DALI score ?	141.287	Aligned residues [#] ?	49
MATRAS score ?	5883.472	Aligned residues [%] ?	81.667
CMO norm. similarity ?	0.751	Coordinate RMSD ?	4.321
PAUL norm. similarity ?	0.532	Distance RMSD ?	3.616
DALI norm. similarity ?	0.326	RMSD100 ?	6.716
MATRAS norm. similarity ?	0.173	Sequence identity [%] ?	16.667

CMO alignment segments ? [back to top]

```

[0 - 1] <-> [7 - 8] ( [6 - 7] <-> [8 - 9] )
[2 - 2] <-> [10 - 10] ( [8 - 8] <-> [11 - 11] )
[3 - 46] <-> [12 - 55] ( [9 - 52] <-> [13 - 56] )
[47 - 48] <-> [58 - 59] ( [53 - 54] <-> [59 - 60] )

```

Figure 4.3: CSA website with different tabs and parts of the information displayed for the CMO alignment of 1otrA and 2di0A.

Chapter 4. Scoring schemes and quality criteria for structure alignment

Bounds on alignment score and similarity. Section *Optimized score* lists the resulting scores: the raw score S_{AB} of proteins A and B according to Equation (3.2) (here, the number of common contacts), and a similarity score that normalizes S_{AB} with respect to the self-similarity of the two proteins, computed as $2S_{AB}/(S_{AA} + S_{BB})$. Our CSA solver is an exact branch-and-bound search as described in Section 3.7. At each step of the solving process, it borders the optimal solution using two values: a lower bound LB , which is the score of the best feasible solution found so far, and an upper bound UB that results from solving the Lagrangian relaxation of Section 3.5.3. When an instance is optimally solved, then the relation $LB = S_{AB} = UB$ holds. Otherwise, $LB \leq S_{AB} \leq UB$, and the so-called relative gap $(UB - LB)/|LB|$ quantifies the precision of the returned score LB . Such a quality guarantee is very useful in the context of large-scale database comparisons where the execution time is usually bounded. It helps to quickly determine the progress of the computation as well as the similarity of the two proteins. If two proteins are dissimilar, the relative gap tends to be large, but the upper bound on the similarity score tends to be low from computation start on. In the considered example aligning 1otrA with 2di0A with respect to CMO yields 125 common contacts, and the corresponding similarity score on a scale from 0 to 1 is 0.751. The relative gap is 0%, indicating that the top-scoring alignment has been found.

Structural conservation and variation. The *Alignment* section displays the computed structure alignment. Residues are color-coded according to either SSE (helix, sheet, coil) as assigned by DSSP [55] or to residue pair score contribution. The second color-coding denotes for each residue pair how structurally prominent it is given the current alignment, *i.e.* how much it adds to the alignment score in comparison to other residue pairs. For an example, see Figure 4.3. For the two proteins containing the CUE domain, this indicates that the first identically aligned leucines are structurally preserved, and in fact this position is part of a motif for binding ubiquitin that consists of an invariant proline and two highly conserved leucines [98]. Pairs of aligned residues with low score contribution highlight structural variations. In the CMO alignment, the N- and C-terminal regions are of little structural importance, as well as the residues in the region of the invariant proline within the CUE domain, because the proline is located in a turn. Such a visualization of residue score contribution can hint towards a manual modification of the alignment by removing aligned residues with low score. In fact, this is what happens in the top-scoring DALIX alignment of 1otrA and 2di0A, in which the four C-terminal residues with low CMO score are excluded from the alignment.

Comprehensive alignment-related data. Additional to the alignment, CSA displays the aligned segments, both using sequential and PDB residue numbering, see Figure 4.3. Numerous raw alignment- and similarity scores are listed, for example the number of aligned residues, sequence identity and RMSD. Furthermore, some statistics concerning the alignment computation are given. These are the number of residues and inter-residue distances considered during computation. They greatly influence the memory consumption of the algorithm: the more inter-residue distances are considered, the

4.3. Web server for comparative structural alignment

more memory is needed and typically the larger the run time. Using default values, CMO only considers distances smaller than 7.5 Å, PAUL considers distances smaller than 8.5 Å (for C_α distances, 9.5 Å), MATRAS scoring uses distances up to 50 Å and DALI scoring all distances. As a consequence, the exact DALIX and MATRASX solvers are extremely memory-demanding, and we currently restrict the computations of these two scores to protein pairs with a maximum average length. The allocation time for setting up all data structures is given, as well as the time actually spent on computing the alignment. The number of visited branch-and-bound nodes gives a good estimate on the progress of the computation. The proteins are superposed according to the alignment and visualized in Jmol. The trace of aligned residues and the distance difference matrix is plotted.

We upload an additional alignment in the tab for the first custom alignment. This alignment aligns only the 38 residues that belong to the respective CUE domain and that are structurally equivalent according to SISY. Furthermore, we upload a second custom alignment which has been generated by the DALI server [40], which uses a heuristic algorithm to find a good alignment according to the DALI score.

Improving, verifying optimality and assessing quality of heuristic alignments.

Many different scores and quality measures can be compared in the *Comparison tab*: the CMO, PAUL, DALI, and MATRAS raw and similarity scores, DALI z-score, TM-score, number and percentage of aligned residues, RMSD_c, RMSD_d, RMSD100 and sequence identity (see Section 4 for details about these scores). For 1otrA and 2di0A, all six computed and uploaded alignments differ from each other. While CMO and PAUL alignment are computed to optimality in less than a second, the DALIX alignment has the potential to be improved by up to 12% and the MATRASX alignment by up to 24%. We also observe that the alignment that was computed by the DALI server and then uploaded is better with respect to DALI score than the alignment computed by our exact algorithm within 30s. We thus increase the maximum run time for DALIX and MATRASX to 10 minutes. Now, both alignments are computed to provable optimality and our top-scoring DALIX alignment slightly improves the heuristic solution returned by the DALI server. DALIX and MATRASX alignments thus can be used to obtain quality guarantees for DALI or MATRAS alignments and in some cases also to either prove their optimality or to compute a better alignment.

Multi-criteria comparison and consensus alignment. Alignment trace comparison as introduced in [27] gives a visual overview about agreements and differences between alignments. Here, any subset of alignments can be shown. Using this visualization, we find that all alignments (except the SISY reference, which excludes 3 residues in the center of the domain) correctly align all 41 residues of the CUE2 domain, and that they differ in aligning the neighboring N- and C-terminal residues. A radar chart compares the different scores, see Figure 4.4. This chart helps to quantify score differences and allows to decide whether one alignment is clearly preferable, *i.e.*, better with respect to all criteria. The chart also allows to make an intuitive decision which alignment is most appropriate in cases in which different scoring schemes disagree as it is the case for 1otrA and 2di0A. Here, intuitively the DALIX alignment is the best choice since it performs good or best according to all criteria.

Chapter 4. Scoring schemes and quality criteria for structure alignment

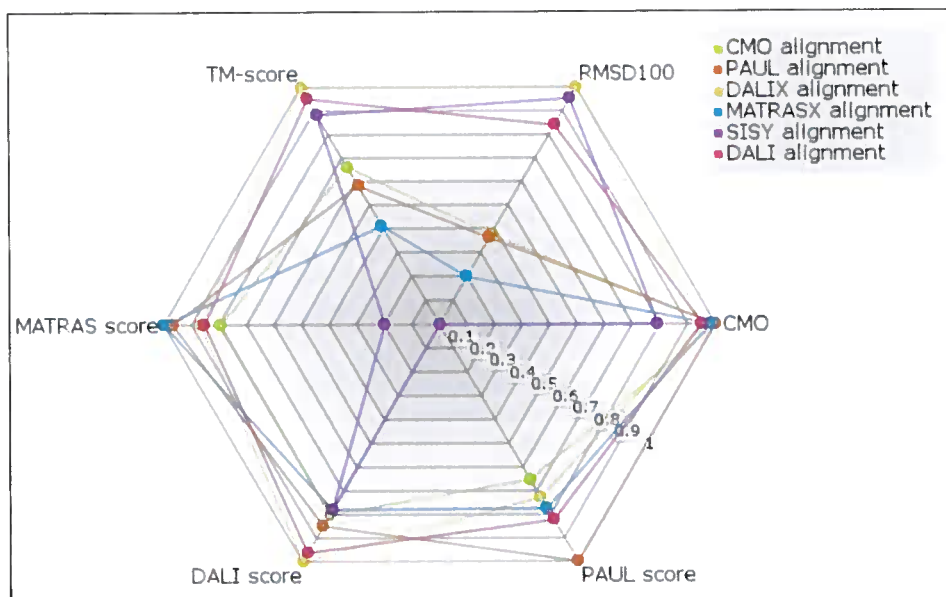


Figure 4.4: A radar chart for comparison of alignment scores for six different alignments of *lotrA* and *2di0A*. The closer a point is to 1, the better the corresponding score. CMO, PAUL, DALIX and MATRASX alignments have been computed by our exact algorithm and are provably optimal concerning their respective score. The SISY reference alignment aligns 38 residues of the CUE2 domain. The DALI alignment was computed by the DALI server and has slightly lower DALI score than the optimal DALIX alignment. The reference alignment is far behind in all scores except RMSD100 and TM-score, for which it performs quite well. The MATRASX alignment performs especially poor for these two measures. Intuitively, the DALIX alignment is most preferable since it has optimal DALI and close to optimal CMO, PAUL and MATRASX scores, as well as the best TM-score and RMSD100.

Two residue pair lists show aligned residues that appear in all or in the majority of the alignments, respectively. They each constitute a consensus alignment. In the case of aligning *lotrA* and *2di0A*, we see that such a consensus is useful: all alignments only agree in aligning the CUE2 domain. The consensus thus highlights the structurally conserved and biologically relevant region of the alignment.

Alignment of flexible proteins. We illustrate the usefulness of comparing structure alignments in the case of protein flexibility. This is a challenge for most structure alignment methods because flexible proteins typically do not superpose well unless the flexibility is accommodated for, *e.g.* by explicitly introducing a hinge.

4.3. Web server for comparative structural alignment

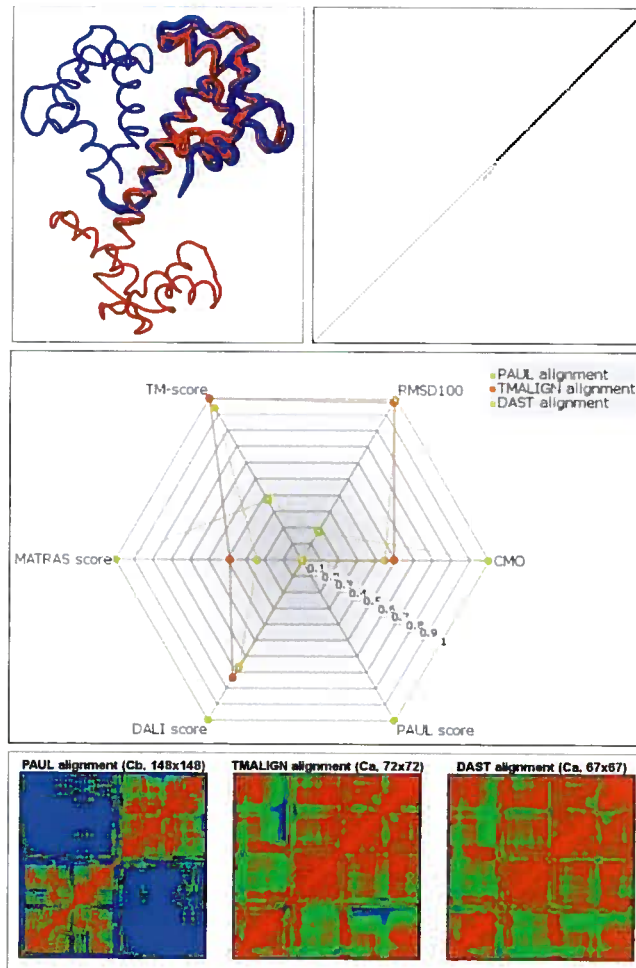


Figure 4.5: Top left: The two calmodulin conformers (PDB IDs 4cln and 2bbm) superpositioned according to the TM-alignment, which aligns only one of the two regions that move relative to each other. Top right: Comparison of the alignment traces. Each axis corresponds to one conformer. Black boxes denote residue pairs aligned by all three scorings, PAUL, TM-ALIGN and DAST. Light (intermediate) shades of gray denote residue pairs aligned by only one (two) scorings. PAUL aligns all residues of the two conformers, TM-ALIGN and DAST the C-terminal region. Center: The radar chart illustrates the difference between scoring schemes that are more in favor of a flexible alignment (CMO, PAUL, DALI and MATRAS) and scoring schemes that are more in favor of a rigid superposition of low RMSD. Bottom: Distance difference matrices show the difference between the flexible PAUL alignment, that aligns all residues in spite of large distance differences (colored blue), and the TM- and DAST alignment that only align the C-terminal region.

Chapter 4. Scoring schemes and quality criteria for structure alignment

Comparing flexible and rigid scoring schemes. We align two conformations of the calmodulin protein (PDB IDs 4cln, chain A and 2bbm, chain A, with a length of 148 residues). In structure 4clnA, calmodulin is bonded to a ligand, in 2bbmA it is unbound. In bound conformation, a central helix is split and the components at the ends of the helix are moved towards each other. We align the two conformations using CMO and PAUL. We furthermore upload the alignments computed by TM-ALIGN [129] and by DAST [69], a local structure alignment method that determines the longest alignment with RMSD_d less than 4 Å. We find that both CMO and PAUL return the same alignment and correctly align the two conformations over their entire length. Figure 4.5 displays the two conformers superposed according to the TM-alignment and the alignment trace comparison. While PAUL and CMO align all residues of the two conformers, TM-ALIGN aligns only the C-terminal, rigidly superposable region (except the C-terminal residue). DAST also aligns the C-terminal region, but excludes and shifts further residues from the alignment. The radar chart as well as the distance difference matrices displayed in Figure 4.5 show why: while CMO, PAUL, DALI and MATRAS scoring by far favor the alignment of the entire conformers, TM-score as well as RMSD100 clearly favor the TM- and DAST alignments, which have a much smaller RMSD, but align only the C-terminal region.

Detecting flexibility and hinges. For each alignment we display the distance difference matrix. This is a symmetric square matrix with entries $|d_{ij}^A - d_{ij}^B|$ at position (i, j) , where i is the i -th aligned position and j the j -th aligned position. Distance differences are visualized using a color gradient in which 0 Å is colored red, 2.5 Å green 5 Å blue. Regions with low inter-residue distance differences correspond to rigidly superposable fragments. For the PAUL alignment of 4clnA and 2bbmA, red blocks in the distance difference matrix indicate that both the N-terminal and C-terminal regions can be superpositioned very precisely. The distance differences between these two regions, however, are large, denoted by the blocks in blue color. The two regions can thus only be well superpositioned individually. A hinge is present at the residue bordering the two blocks (position 80) [26]. TM-ALIGN and DAST align only the C-terminal region, thus avoiding any large distance differences. DAST is more restrictive in excluding large distance differences, it does not align a few residues that are still aligned by the TM-alignment and which have distance differences of about 5 Å, colored in blue.

Scores as CMO and PAUL, which implicitly ignore RMSD, are useful to gain information about flexible regions. While this feature is beneficial for flexible proteins it may also introduce flexibility where this is not appropriate. Protein similarities consisting in compact, well superposable fragments are therefore often better detected by maximizing scores like the TM- or the DAST score.

Computational results

In this chapter we present the computational results that were obtained using the different mathematical models and algorithmic approaches described in this thesis. In our experiments, we follow two lines of investigation. First, in Section 5.1, we make our algorithms practical and competitive by developing and optimizing a dedicated scoring scheme called PAUL scoring. It excludes as many distance pairs as possible while still computing alignments as accurate as those of state-of-the-art structure alignment programs. Second, in Section 5.2, we investigate the much more difficult problem of aligning distance matrices in the most general setting: considering all distance pairs and assigning positive and negative structural scores to them. This is especially important, since many existing heuristic structure alignment algorithms fall into this category, for example the DALI program, which is one of the most accurate and most widely-used structure alignment tools. Therefore we focus in our second line of investigation on optimal distance matrix alignment using the DALI scoring function.

5.1 PAUL optimization and evaluation

PAUL is the name of our implementation of the Lagrangian approach of Caprara *et al.* which has been presented in Section 3.5.2. For details about the corresponding software program [119], see Section 3.8.2. It uses an iterative double dynamic programming algorithm which is based on solving the Lagrangian dual of a model in which a technique called variable splitting has been applied. Caprara *et al.* developed it for solving the contact map overlap problem. With PAUL, we go one step further towards protein structure distance matrix alignment by generalizing the objective function to incorporate additional elements used by many 2-dimensional scoring schemes. These are discussed in detail in Section 4.1.3. Of those, we finally decide to use a distance threshold d_t , a distance difference threshold Δ_t , a penalty c for aligning two residues which functions as an offset on the structural score, and affine gap costs with gap open penalty ν and gap extension penalty $\xi = 0.25 \cdot \nu$. The structural score of a distance pair decreases with increasing distance difference and the slope of the corresponding linear function is described by a parameter θ . PAUL aligns C_α , C_β or all-atom inter-residue distance matrices. In the following Section 5.1.1, we conduct a large-scale optimization of these parameters in order to obtain our custom PAUL scoring scheme.

Chapter 5. Computational results

In Section 5.1.2, PAUL is comprehensively benchmarked. For example, we compare the accuracies of PAUL alignments to the accuracies of alignments from many other state-of-the-art structure alignment programs and find that they are competitive. We also characterize some properties, strengths and weaknesses of PAUL alignments and consider a few alignment instances in more detail. Further, we investigate how the algorithm performs when using the non-negative portion of the established MATRAS scoring function together with a distance threshold. We need to impose these two restrictions, because employing negative structural scores and omitting the distance threshold is both beyond the means of the algorithmic approach. Negative structural scores can not be handled by the algorithm and considering all distance pairs using a larger or no distance threshold is out of scope because of memory issues due to the large number of $O(n_A^2 n_B^2)$ Lagrangian multipliers. We address these points with an alternative Lagrangian approach whose results are presented in the next section. Still we also compute good alignments using only a subset of the structural scores from MATRAS. Our initial results on PAUL optimization and evaluation have been published in conference proceedings [118] and the comprehensive, final study in [116].

5.1.1 Parameter optimization

Experimental setup. In order to make a first step towards aligning general distance matrices, we use, besides our own PAUL scoring, the positive portion of an established scoring function which is employed in the structure alignment program MATRAS [58].

Analogous to sequence substitution matrices, MATRAS uses log-odds value matrices M as scores. A value $M_{|d_A|,|d_B|}$ indicates the log-likelihood that distance d_A is aligned with distance d_B . Matrices are determined for 20 different sequence separations, *i.e.*, numbers of residues in the sequence over which the distance ranges, *cf.* Section 4.1.2. A positive log-likelihood means that the corresponding distances are aligned more often than expected by chance. We use positive MATRAS log-odds scores as structural scoring. In addition, we use MATRAS scores for filtering. In such log-odds filtering, we omit all distance pairs with negative MATRAS log-odds score and evaluate the remaining distance pairs using PAUL score.

Further, we investigate the influence of filtering based on SSE assignment. Our SSE-based filter disallows aligning residues from α -helices with residues from β -sheets (using DSSP assignment). We investigate how alignment accuracy and performance change when using filters and when changing scoring function parameters.

To optimize parameters we use a training set consisting of structure-based alignments from the HOMSTRAD database (Oct 2008 release). As these alignments are manually curated by experts, we consider them as gold standard reference alignments (*cf.* Section 4.2.3). From HOMSTRAD we consider only those non-corrupt 302 protein families with exactly two members from the twilight or midnight zone of sequence identities below 35%. Here, sequence identity denotes the percentage of identically aligned residues in comparison to the total number of aligned residues. We optimize the parameters on a training set of 200 alignments and evaluate them on a test set that consists of the remaining 102 alignments. We measure the quality of the results computed by

structure alignment algorithms in terms of the achieved alignment accuracy, which is the number of correctly aligned residues divided by the number of aligned residues in the reference alignment.

We conduct an extensive parameter sweep in order to determine robust scoring function parameters that promote a high average alignment accuracy. See Section 4.1.3 for a detailed description of the parameters of PAUL scoring. For the best overall parameter sets $(d_t, \Delta_t, \theta, c, \nu)$ for combinations of C_α , C_β , or all-atom distances with or without filtering, we align the HOMSTRAD test set. On this test set, we compare PAUL performance to DALI, a widely-used state-of-the-art structure alignment algorithm that is ranked very high in many benchmarks [74, 93, 89, 8]. For evaluation, we use the 99 test set protein pairs for which DALI returns an alignment; this subset we call the consolidated test set.

Because parameters depend on each other, we optimize them in two stages. First we find good values for the structural scores and afterwards we adjust the sequence penalty and gap costs accordingly. In the first stage, we evaluate a range of parameter values for the distance threshold d_t , the distance difference threshold Δ_t , and the steepness θ . For the best parameter set that we find, we subsequently adapt the sequence penalty c and the gap open costs ν . As in MATRAS, we define the gap extension cost ξ as $0.25 \cdot \nu$. We optimize parameters for three different variants of the model: without applying any filtering, with filtering based on SSE assignment, and with filtering based on SSE assignment as well as log-odds values.

We optimize parameters for the distance thresholds $d_t \in \{7 \text{ \AA}, 7.5 \text{ \AA}, \dots, 10.5 \text{ \AA}\}$ for C_α and C_β distances and $d_t \in \{4 \text{ \AA}, 4.5 \text{ \AA}, \dots, 7 \text{ \AA}\}$ for all-atom distances. We do not use larger distance thresholds because they do not allow to fit every instance into memory. Furthermore, we divide the angle determined by θ in ten equal parts, resulting in $\theta = \{1.5, 3.1, 5, 7.1, 9.8, 13.5, 19.2, 29.8, 59.3, 1000\}$. At a value of $\theta = 1000$, all distance pairs obtain nearly the same score. This parameter setting thus represents contact map overlap. For the maximum distance difference, we evaluate $\Delta_t \in \{1.5, 2, \dots, d_t - 1.5\}$ for C_α and C_β matrices and $\Delta_t \in \{0.5, 1, \dots, d_t - 0.5\}$ for all-atom matrices. Note that C_α and C_β distances do not differ more than $d_t - 1.5 \text{ \AA}$ and all-atom distances not more than $d_t - 0.5 \text{ \AA}$; the last parameter value of Δ_t thus refers to omitting the distance difference threshold altogether (as in CMO). We compute the average alignment accuracy for each parameter set (d_t, θ, Δ_t) . Subsequently, we optimize the sequence penalty c and the gap open penalty ν . The sequence penalty ranges over $c \in \{0, -\frac{1}{2}\theta, \dots, -5\theta\}$. The gap penalty ν lies in $\{0, 1, \dots, 30\}$. As in [58], the gap extension penalty ξ is fixed to $0.25 \cdot \nu$. When using positive MATRAS scores, we use gap costs ν in $\{0, 40, 80, \dots, 400\}$.

For computations, we use cluster nodes equipped each with two quad core 2.26 GHz Intel Xeon processors and 24 GB of main memory running 64 bit Linux. During parameter optimization, we compute on each node 4 PAUL alignments in parallel. In the first stage of optimization, we have to evaluate many parameter sets. Therefore, we choose a maximum time limit of 90 CPU s and a maximum number of 1 000 Lagrangian iterations for each computation. Whichever is reached first terminates the computation.

Before determining gap and sequence penalties, we apply 10-fold cross-validation in

Chapter 5. Computational results

order to assess the performance of PAUL on the HOMSTRAD training set alignments solely with the optimized structural score. Then, in the second stage, we choose the best parameter set (d_t , Δ_t , θ) for each distance type (C_α , C_β , all-atom) and each filter (no filter, SSE filter, SSE and log-odds filter). For all-atom distances, there are no log-odds scores available. For the resulting 8 parameter sets, we determine good values for the sequence penalty c and the gap penalty ν using a time limit of 30 CPU mins.

For updating Lagrangian multipliers, we use the scheme of Andonov *et al.* [4], which adjusts them more rapidly and therefore leads, in our experiments, in shorter time spans to results comparable to those of a more conservative update scheme like the one of Caprara *et al.* [14]. The update scheme of [4] is more effective within a branch-and-bound approach though, and without branch-and-bound (as it is the case in our algorithm), the multipliers become smaller than machine precision and we can solve a lot less alignments to optimality than when applying the update scheme of [14]. Therefore, we use the update scheme of [14] for aligning the SISO and RIPC set.

Results. In a preprocessing step we compute histograms of aligned distances over the HOMSTRAD training set alignments. Figure 5.1 displays the result for C_β distances. For all types of inter-residue distances the distribution adopts its maximum at the bisecting line which indicates identical distances. For close distances of less than 12 Å we observe distinct peaks for certain aligned distances. These peaks represent typical distances within secondary structure elements and within super-secondary structures.

We call the PAUL implementation that uses positive MATRAS log-odds values as structural score PMATRAS. For PMATRAS, a high distance threshold is beneficial. For C_α distances, the best distance threshold is 10 Å and for C_β distances it is 10.5 Å (see Table 5.1, which lists all values). Due to high memory requirements, we currently cannot evaluate higher distance thresholds. The optimized gap costs for the best distance thresholds are a gap open penalty of 160 and a gap extension penalty of 40 for C_α distances and a gap open penalty of 280 and a gap extension penalty of 70 for C_β distances. The average alignment accuracies for most distance thresholds lie below 85% when no gap penalties are used. Using gap penalties, C_α distances reach a maximum average alignment accuracy of 88.7% for $d_t = 10$ Å and C_β distances of 88.9% for $d_t = 10.5$ Å. Corresponding average test set accuracies are 91.1% for C_α and 90.4% for C_β distances (see Table 5.2).

d_t	7	7.5	8	8.5	9	9.5	10	10.5
C_α accuracy	82.4	84.0	84.2	84.2	84.9	85.3	85.8	85.5
C_β accuracy	81.2	82.0	82.9	83.4	83.2	83.6	83.7	83.9

Table 5.1: Average alignment accuracy for non-negative MATRAS scoring for C_α and C_β distances for each distance threshold. Maxima are denoted bold.

During the first stage of the parameter optimization for the PAUL scoring scheme, we determine only the parameters that influence the structural scores. These are the distance threshold d_t , the distance difference threshold Δ_t and the steepness θ of our scoring function (4.18). Average alignment accuracies on HOMSTRAD vary only slightly for different parameter values and lie between 85 and 88%; only very small distance

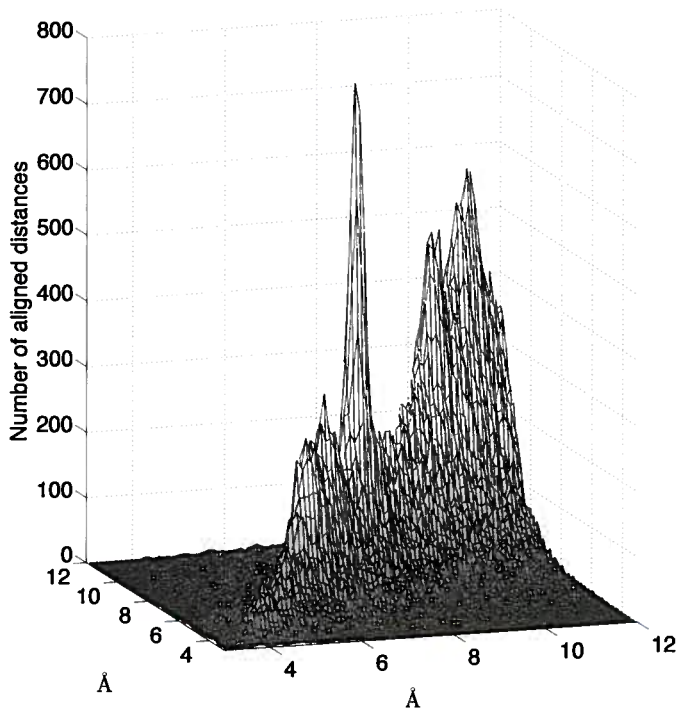


Figure 5.1: Number of aligned C_β distances over all HOMSTRAD training set pairwise alignments.

Scoring Filter	C_α				C_β				All-atom	
	No	PAUL SSE	Log-odds	MATRAS	No	PAUL SSE	Log-odds	MATRAS	No	SSE
d_t	9.5	9.5	10	10	8.5	8.5	10	10.5	5.5	6
Δ_t	7.5	4	4.5	-	3.5	7	5.5	-	3	3.5
θ	5	7.1	5	-	7.1	9.8	7.1	-	3.1	5
c	12.5	7.1	17.5	-	17.75	19.6	17.75	-	7.75	12.5
ν	10	9	25	160	21	23	30	280	10	15
Training aa	89.5	89.7	89.3	88.7	89.6	90.2	89.0	88.9	89.3	89.8
Average test aa	91.0	90.8	91.3	91.1	90.9	90.9	91.1	90.4	91.3	91.3
Median test aa	92.7	92.8	93.4	92.8	92.7	91.9	92.7	91.9	92.4	92.4

Table 5.2: Optimized parameters and corresponding alignment accuracies (aa) for C_α , C_β , and all-atom distances using either no filtering, SSE filtering or SSE and log-odds filtering. The average training and test set alignment accuracy for each distance type after optimization of structural score in a first step, and simultaneous optimization of sequence penalty and gap costs in a second step is given. Gap extension penalties ξ are 0.25 times the reported gap open penalty ν . MATRAS scores comprise only positive values. Test set accuracy is the average/median over the consolidated test set. Highest training and test set accuracy are denoted bold.

Table 5.3: Optimized parameter values, average alignment accuracy (aa), and cross-validation results for C_α (top), C_β (center) and all-atom distances (bottom) for different filter settings and distance thresholds. Respective maxima are denoted bold.

d_i	No Filter										SSE Filter										SSE and log-odds Filter									
	7	7.5	8	8.5	9	9.5	10	10.5	11	11.5	7	7.5	8	8.5	9	9.5	10	10.5	11	11.5	7	7.5	8	8.5	9	9.5	10	10.5	11	11.5
Δ_i	2.5	3	3	3.5	4	4.5	5	5.5	6	6.5	2.5	3	3	3.5	4	4.5	5	5.5	6	6.5	2.5	3	3	3.5	4	4.5	5	5.5	6	6.5
θ	3	5	5	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Maximum aa	81.6	84.3	85.8	87.1	86.8	86.2	85.9	85.5	82.2	84.6	81.6	84.3	85.8	87.1	86.8	86.2	85.9	85.5	82.2	84.6	81.6	84.3	85.8	87.1	86.8	86.2	85.9	85.5	82.2	84.6
Cross-validation aa	81.0	84.1	85.4	86.7	85.4	85.9	85.4	85.4	85.4	85.4	81.0	84.1	85.4	86.7	85.4	85.9	85.4	85.4	85.4	85.4	81.0	84.1	85.4	86.7	85.4	85.9	85.4	85.4	85.4	85.4

thresholds d_t have smaller alignment accuracies. Table 5.3 lists the best parameter values for the three types of inter-residue distances and three filter settings at each distance threshold together with the corresponding average alignment accuracy and cross-validation accuracy.

When optimizing the PAUL parameters of scoring function (4.18) we find that the average alignment accuracies on HOMSTRAD vary only slightly for different parameter values (*cf.* Table 5.3). The parameter values after optimization are robust: slightly changed parameter settings still have comparable average alignment accuracy. Using a secondary structure filter always leads to better results, but an additional log-odds filter usually not. The best distance difference threshold Δ_t lies mostly between 3 and 4 Å for C_α and C_β distances and between 2 and 3 Å for all-atom distances. The best parameter θ is in the range of 3 to 10. When a log-odds filter is used, higher distance thresholds lead to better alignments. The average alignment accuracies reached with C_α and all-atom distances are comparable, and those reached with C_β distances are lower. The best cross-validation accuracy on the training set is 87.6% for C_α , 87.4% for C_β , and 87.8% for all-atom distances (Table 5.3 lists all values).

In a second stage we fix the best parameter set (d_t, Δ_t, θ) for each combination of an inter-residue type (C_α , C_β , all-atom) with a filter type and optimize the parameters without influence on structural scores. These are the sequence penalty c and the gap open penalty ν . We vary ν and fix, in analogy to MATRAS gap costs, the gap extension penalty ξ to $0.25 \cdot \nu$. The best value for the sequence penalty c is in most cases 2 or 2.5 times θ and the best gap penalty ν is usually up to one θ higher than c . The final average alignment accuracies on the training set are in the range of 89.0 to 90.2%, see Table 5.2. The average alignment accuracies on the consolidated test set lie in the range of 90.8 to 91.3% and the median alignment accuracies in the range of 91.9 to 93.4%. This slightly higher test- than training set accuracy is due to batch effects. DALI reaches on the consolidated test set an average accuracy of 89.0% and a median accuracy of 91.0%.

We obtain the highest average training set alignment accuracy of 90.2% for C_β distances with SSE filter and a parameter set $(d_t, \Delta_t, \theta, c, g) = (8.5, 7, 9.8, 19.6, 23)$. Refer to Figures 5.1 and 4.2 for a histogram of aligned C_β distances and a visualization of the scoring function, respectively. We use this setting for the evaluation on the SISY and RIPC sets. This is also PAUL's default scoring scheme.

Conclusion. The results on HOMSTRAD show that our algorithm and scoring scheme perform well with any type of inter-residue distance matrix, despite the qualitative difference between C_α , C_β and all-atom distances. On this test set PAUL reaches a higher average and median alignment accuracy than DALI with all types of inter-residue distances and any type of filter. Furthermore, our optimized scoring function is robust against slight parameter changes and always performs better than contact map overlap scoring, which constitutes a special case of the parameter settings that have been evaluated. Introducing additionally sequence penalty and gap costs consistently increases the alignment accuracy.

5.1.2 Evaluating alignment accuracy

Experimental setup. We use the parameters optimized on HOMSTRAD to evaluate PAUL on two distinct, more challenging data sets, SISY and RIPC. Section 4.2.3 provides details on SISY and RIPC and their construction, Tables 4.3 and 4.4 list the corresponding alignment instances. We compare PAUL to the state-of-the-art structure alignment programs DALI, MATRAS, MATT, SHEBA, FATCAT, CE and CA, which were benchmarked by Berbalk *et al.* [8].

Two recent structure alignment methods, PROTDEFORM [93] and MAX-PAIRS [89] have been evaluated on an older version of the SISY set, which we name in the following SISY.v1. For SISY.v1, the consolidated set of alignments for which no method fails comprises 106 protein pairs. In [93] and [89], DALI, SSAP, PROTDEFORM, MATRAS, MATT, PPM, LGA, VOROLIGN, and RASH have been benchmarked on SISY.v1. We evaluate PAUL also on the consolidated SISY.v1 set and compare the results.

XML formatted alignments of the SISY and RIPC benchmark of Berbalk *et al.* [8] are available at <http://biwww.che.sbg.ac.at/RSA>. For RMSD_c computations, we use these files. If an alignment method returns several alternative alignments, these are listed in the corresponding XML file. For RMSD_c evaluations, we use always the first alternative.

SISY and RIPC alignment accuracies are taken directly from the supplementary data of [8]. The SISY.v1 alignments computed by Rocha *et al.* [93] are located at <http://dmi.uib.es/people/jairo/bio/ProtDeform> and the ones computed by Poleksic [89] at http://bioinformatics.cs.uni.edu/opt_align.html. PAUL alignments of SISY, SISY.v1 and RIPC can be downloaded from PAUL's website at <http://www.mi.fu-berlin.de/w/LiSA>.

From the list of structure alignment algorithms that have been evaluated on SISY.v1, we omit FLEXPROT, because Rocha *et al.* [93] decided to take it out of their evaluation because there were unfixed technical problems perhaps due to reading and/or assigning PDB numberings. For similar reasons, we exclude TM-ALIGN from the comparison. Furthermore, we need to recompute the individual alignment accuracies of the algorithms evaluated by Poleksic [89]. For this task we use the newly available evaluation scripts of Berbalk *et al.* [8] and their XML-formatted reference files. We realize, that for the (not yet XML-formatted) reference alignments used in the study of [89], different criteria were applied for including residues in the analysis. Using the XML-formatted reference alignments, the average alignment accuracy changes only very slightly compared to the accuracy reported in [89].

For aligning proteins from SISY, SISY.v1 and RIPC, we use a maximum run time of 30 CPU minutes per alignment. Note that the actual run time in which we observe improvements depends strongly on the length of the proteins and is usually much shorter (*cf.* Figure 5.4). Nonetheless, in order to prove optimality of a solution, a conservative and rather deliberate update scheme for the Lagrangian multipliers (*e.g.* the one of [14]) is needed and thus a longer overall run time.

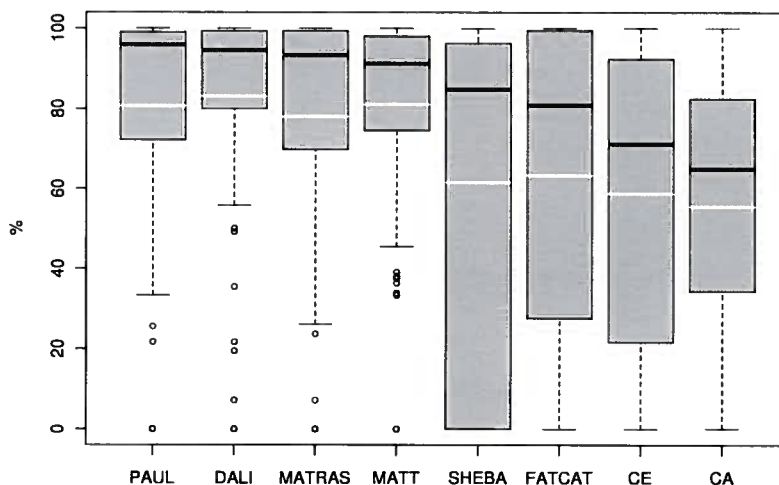


Figure 5.2: Box plots display median and quartiles of the distributions of percentages of alignment accuracies for the *SISY* set for PAUL, DALI, MATRAS, MATT, SHEBA, FATCAT, CE, and CA. The ordering is according to decreasing median. White lines denote average accuracies.

Results for *SISY*. Figure 5.2 displays the accuracies of PAUL, DALI, MATRAS, MATT, SHEBA, FATCAT, CE and CA. PAUL alignments reach with 95.9% the highest median, but with 80.7% a slightly lower average alignment accuracy than MATT and DALI. According to Wilcoxon signed rank tests, PAUL matches the reference alignments significantly better than SHEBA, FATCAT, CE and CA (p -value $< 10^{-4}$). PAUL furthermore computed more than 25% of alignments (26) to provable optimality. The correlation between alignment accuracies of different methods lies mainly between 0.5 and 0.6 and is thus generally low (see Figure 5.3). Figure 5.4 displays the run time versus the change in alignment score. In most instances long run times are needed to (try to) prove optimality, but very accurate alignments have been determined already early during the computation.

PAUL with traditional contact map overlap as scoring function ($d_t = 7.5 \text{ \AA}$) reached a median alignment accuracy of 88.5% and an average alignment accuracy of 72.1%. Because rounding can be applied for the decimal upper bound of the integer-value contact map overlap score, even 48 of the alignments are computed to optimality, that is, almost 50%.

We also test the performance of PMATRAS, our algorithm together with positive MATRAS scores. The results show that although using only positive scores and applying a distance threshold, PMATRAS achieves accuracies competitive with those achieved by MATRAS (see Figure 5.5), but not as high as PAUL using our customized scoring function. With C_α distances, PMATRAS reaches a higher median but a lower average alignment accuracy than MATRAS. PMATRAS with C_β distances performs slightly worse according to both median and average. The correlation between PMATRAS and MATRAS alignment accuracies is not particularly high (Pearson and Spearman correlations

Chapter 5. Computational results

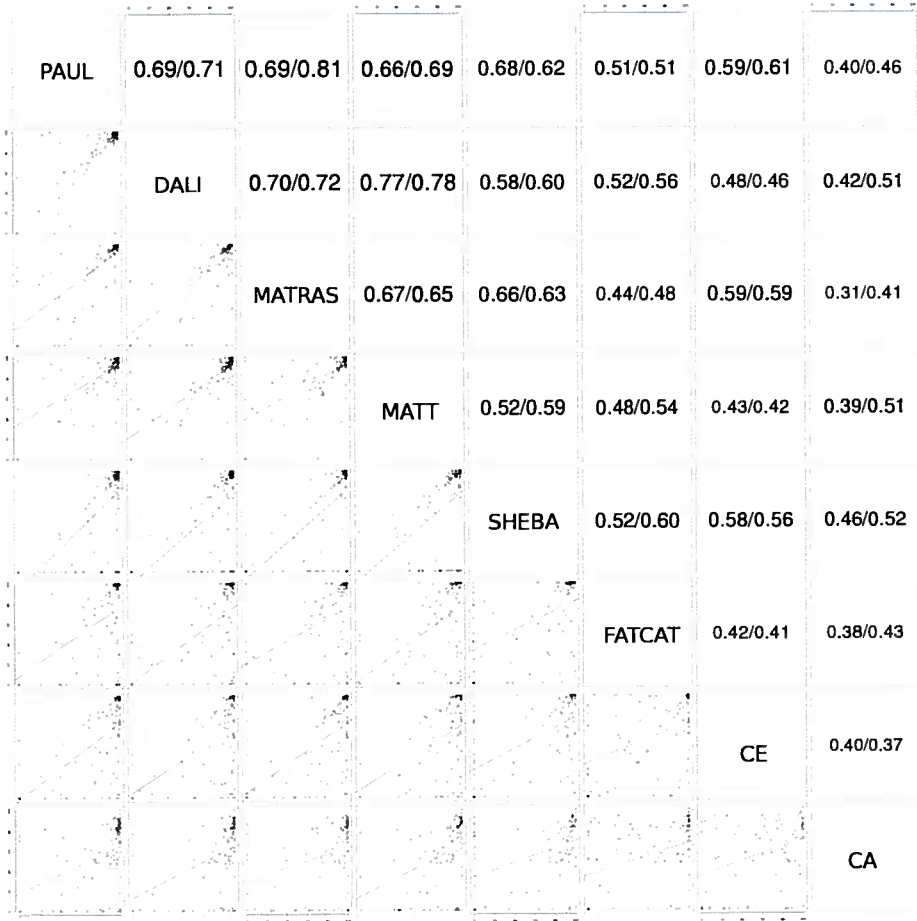


Figure 5.3: Correlations of alignment accuracies of different structure alignment methods for the consolidated **SISY** set. The first value is the Pearson correlation coefficient, the second the Spearman correlation coefficient.

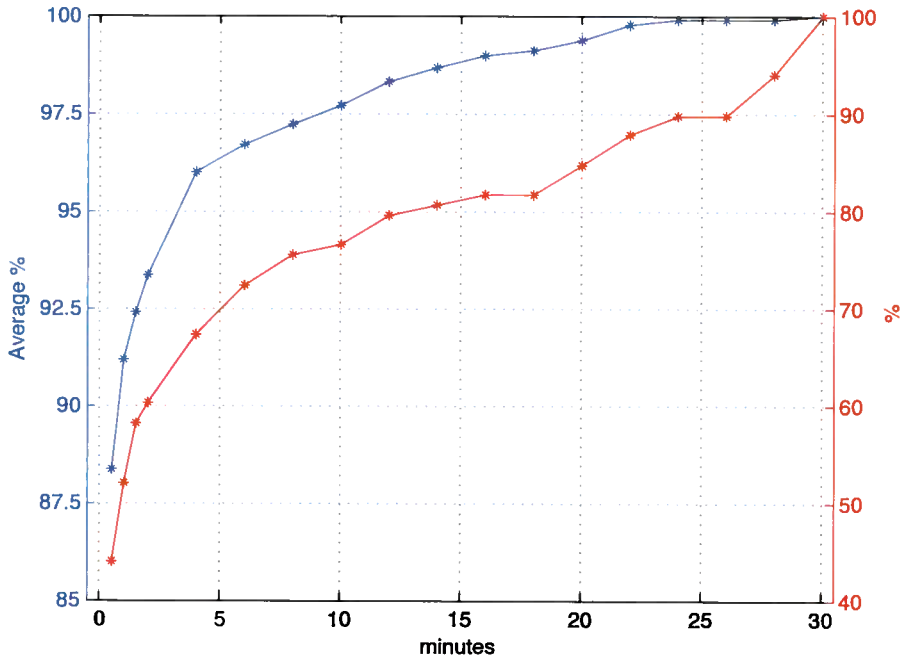


Figure 5.4: Run time versus accuracy of PAUL. We compute the consolidated **SISY** set of 98 alignments for different maximum run times. We compare the score of the alignments computed at each maximum run time with the scores of the alignments computed after 30 CPU minutes (since we maximize the score, it can either increase or remain the same with increased run time). The blue line denotes the average of the percentage of the current scores with respect to the highest score after 30 minutes. The red line denotes the percentage of alignments that have identical score to the score of the alignment computed after 30 minutes. For example, for 80% of the alignments, the score and thus also the alignment accuracy does not change any more after 12 minutes. The red line of the graphic shows that about 50% of the very accurate PAUL alignments can be computed in less than a minute. Alignments that still improve after a long time span tend to be very long and the increase of their scores is very small (see blue curve).

Chapter 5. Computational results

0.67 and 0.73, respectively, for C_α distances; 0.66 and 0.72 for C_β). The correlation between PMATRAS with C_α distances and PMATRAS with C_β distances is in contrast very high (Pearson correlation 0.91, Spearman 0.87).

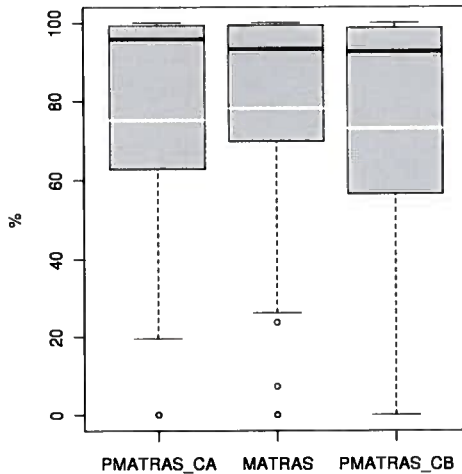


Figure 5.5: Box plots display median and quartiles of the distributions of percentages of alignment accuracies for the consolidated **SISY** set; displayed for PMATRAS alignments (alignments computed using our algorithm with MATRAS scores) for C_α (PMATRAS_CA) and C_β (PMATRAS_CB) distances respectively and for alignments computed using the MATRAS program. White lines denote the average alignment accuracies.

Results for SISY.v1. On **SISY.v1**, PROTDEFORM [93] and MAX-PAIRS [89] have been evaluated and compared to DALI, SSAP, MAX-PAIRS, PROTDEFORM, MATRAS, MATT, PPM, LGA, VOROLIGN and RASH. For this data set, PAUL alignments reach with 96.3% the highest median alignment accuracy of all methods and, slightly after MATT, with 82.4% the second highest average accuracy (for MAX-PAIRS we compare only to the version that is solely structural and does not use BLOSUM62 substitution scores). The box plots of alignment accuracies for each of the methods are displayed in Figure 5.6.

More than 30% (32) of the alignments are computed to optimality. Many of the methods that have been tested on this data set perform very well and should be considered competitive in terms of alignment accuracy. PAUL alignments match the reference alignments significantly better than LGA, VOROLIGN and RASH alignments (p -value $< 10^{-4}$). The correlation between alignment accuracies achieved by different methods is, as on **SISY**, generally low, see Figure 5.7.

Average run times of the compared structure alignment methods vary notably, between less than a second to up to 24 seconds (see Table 5.4, which is taken from [93]). PAUL tries to compute the alignments to optimality and is terminated after 30 CPU minutes; nonetheless, often alignments do not change any more after much shorter time spans of 1 to 2 minutes; this is visualized in Figure 5.4.

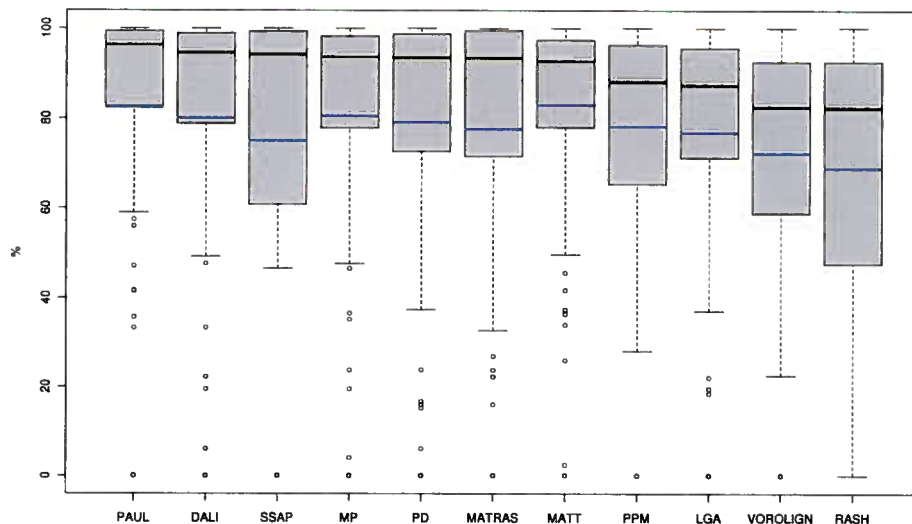


Figure 5.6: Box plots display median and quartiles of the distributions of percentages of alignment accuracies for the consolidated **SISY.v1** set for the alignment methods PAUL, DALI, SSAP, MAX-PAIRS (MP), PROTDEFORM (PD), MATRAS, MATT, PPM, LGA, VOROLIGN, and RASH. Blue lines denote the average alignment accuracies.

	SSAP	DALI	MATT	RASH	PROTDEFORM	PPM	MATRAS	VOROLIGN
Time (s/pair)	23.9	21.9	12.4	10.4	8.5	3.6	1.1	0.6

Table 5.4: The speed of the structure alignment programs on **SISY.v1** in average number of seconds per protein pair. This table is taken from [93]. Their computations were carried out on an Intel Centrino Duo at 1.66 GHz, running Linux. PAUL attempts to compute an alignment that is provably optimal with respect to the scoring function and is terminated after 30 CPU minutes. For a visualization of the typically much shorter run times in which we observe changes of the alignment (and thus improvements of the lower bound of the score) refer to Figure 5.4.

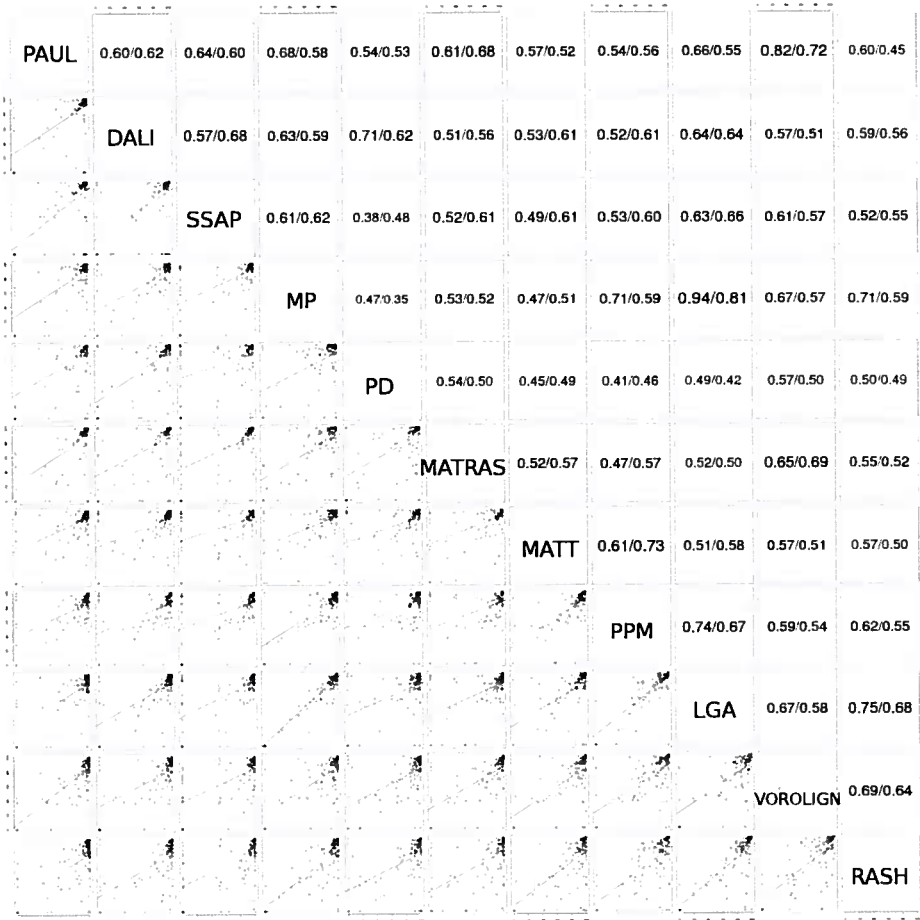


Figure 5.7: Correlations of alignment accuracies of different structure alignment methods (PAUL, DALI, SSAP, MAX-PAIRS (MP), PROTDEFORM (PD), MATRAS, MATT, PPM, LGA, VOROLIGN, and RASH) for the consolidated SISY.v1 data set. The first value is the Pearson correlation coefficient, the second the Spearman correlation coefficient.

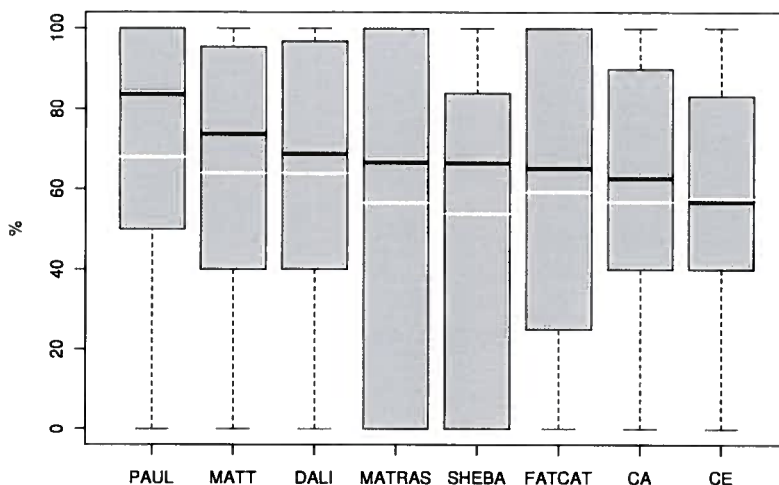


Figure 5.8: Box plots display median and quartiles of the distributions of percentages of alignment accuracies for the RIPC set for PAUL, DALI, MATRAS, MATT, SHEBA, FATCAT, CE, and CA. The ordering is according to decreasing median. White lines denote average accuracies.

Results for RIPC. For RIPC, PAUL reaches the highest average and median alignment accuracy of all methods tested (DALI, MATRAS, MATT, SHEBA, FATCAT, CE, and CA). Box plots of alignment accuracies are given in Figure 5.8. Four of the 22 PAUL alignments are computed to optimality, but only two of those four reach 100% alignment accuracy with respect to the reference alignment (see Table 5.5).

PAUL is furthermore unsuccessful in four cases, generating alignments with 0% alignment accuracy (see Table 5.5). All these problematic cases correspond to pairs of proteins related by circular permutation where the reference alignments also include extensive indels. This result is expected as PAUL was not designed to account for circular permutation. In general the other tested methods also perform poorly on these four pairs, with the exception of CA, which allows for non-sequential alignments.

There are five cases where PAUL performs better than most of the other methods in terms of alignment accuracy (*d1d5fa_ vs. d1nd7a_*, *d1dlial vs. d1mv8a1*, *d1l5ba_ vs. d1l5ea_*, *d1gsaa1 vs. d2hgaa1*, and *d2bbma_ vs. d4clna_*). These cases correspond to pairs of proteins with extensive structural variation resulting from flexibility or from divergent evolution. Figures 5.9 and 5.10 provide two examples. In four of these cases PAUL alignments have 100% accuracy, with FATCAT and MATRAS also generating high quality alignments. The fifth case (*d1gsaa1 vs. d2hgaa1*) corresponds to two remote homologous proteins with considerable structural variation where the PAUL alignment is 80% accurate, while other methods achieve only 40% accuracy. See Figure 5.11 for a visualization.

Discussion and conclusion. The SISY and RIPC sets give us insight into the alignments produced by PAUL. When compared to reference alignments, the alignment

Chapter 5. Computational results

Domain 1	Domain 2	Accuracy	Gap[%]	Optimal
d1an9a1	d1npxa1	63.6	32.1	
d1ay9b_	d1b12a_	90	0.1	
d1b5ta_	d1k87a2	50	62.6	
d1crla_	d1edea_	66.7	375.5	
d1d5fa_	d1nd7a_	100	<0.1	
d1dlia1	d1mv8a1	100	0.3	
d1gbga_	d1ovwa_	100	6.1	
d1ggga_	d1wdna_	97.3	0	yes
d1gsaa1	d2hgsa1	80	<0.1	
d1hava_	d1kxfa_	100	6.1	
d1hcyb2	d1lnlb1	50	112.7	
d1jj7a_	d1lvga_	100	46.8	
d1jwyb_	d1puja_	0	71.4	
d1jwyb_	d1u0la2	0	66.0	
d1kiaa_	d1nw5a_	0	52.5	
d1l5ba_	d1l5ea_	100	0	yes
d1nlsa_	d2bqpa_	83.3	131.7	
d1nw5a_	d2adma1	30.8	39.7	
d1qasa2	d1rsya_	84	0	yes
d1qq5a_	d3chya_	0	24.3	
d2adma1	d2hmyb_	100	74.7	
d2bbma_	d4clna_	100	0	yes

Table 5.5: RIPC results: the aligned domains, PAUL alignment accuracy, relative gap $(UB - LB)/|LB|$ in percent and indication whether the alignment is provably score-optimal.



Figure 5.9: PAUL successfully aligns two conformations of a flexible protein. This RIPC protein pair corresponds to two conformations of the *Drosophila melanogaster* calmodulin. The calcium bound conformation is shown on the right (d4clna_) [106]. On the left conformation, calmodulin is also bonded to the target peptide (d2bbma_) [47]. The N and C-terminal subdomains (residues 1-73 and 83-148 respectively) move relative to each other when binding the target peptide. The RMSD_c of the N-terminal subdomain (in orange) is 3.65 Å, the RMSD_c for the superposition of the C-terminal domain (in red) is 2.10 Å. The structures are viewed in the same orientation according to the superposition of the C-terminal subdomain. The PAUL alignment has 100% accuracy. The two structures correspond to the same protein (100% sequence identity), but sequence similarity is not used by PAUL to compute alignments. Image prepared with CHIMERA [86].

accuracy is very high and often better than that of other state-of-the-art structure alignment methods. This good performance of PAUL can be attributed to several factors. First, PAUL always computes alignments on single-residue level (contrary to fragments) and without using any hierarchical approach that might introduce mistakes on the first, broad level which cannot be corrected later on. Second, structure alignment methods that optimize a 2-dimensional scoring scheme and are thus based on inter-residue distances (such as PAUL) are more adequate to compare flexible proteins or proteins with considerable structural variation than methods that optimize a 3-dimensional scoring scheme. The latter methods, which rely on rigid body superposition, have to introduce twists to accommodate for structural variation. DALI and MATT also perform well on the SISKY and RIPC data sets as they both take into account protein flexibility in the comparisons. DALI relies on inter-residue distances, and MATT is a superposition-based method that allows for different relative orientation between fragment pairs during alignment generation. When evaluating RIPC results, one should be cautious, because the data set is very small, nonetheless, the results indicate that PAUL compares favorably to other methods in protein pairs that are challenging for structural comparison, in particular when these proteins show considerable structural

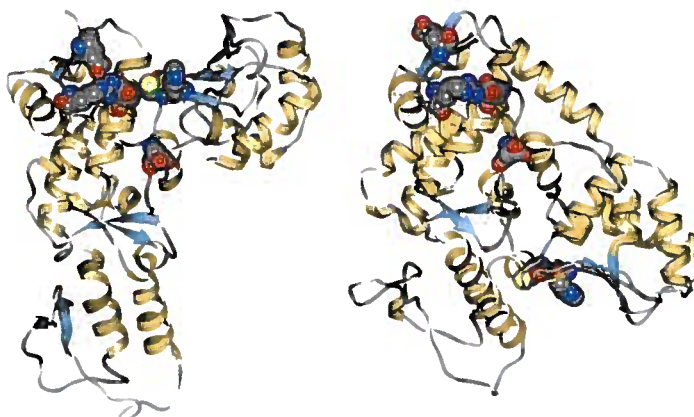


Figure 5.10: Large conformation variation in an RPC pair. Ubiquitin-protein ligase E3A (d1d5fa_) [46] on the left, E3 ubiquitin-protein ligase WWP1 (d1nd7a_) [110] on the right. Sequence identity is 35%. Both PAUL and FATCAT compute an alignment with 100% accuracy, while other tested methods generate less accurate alignments. β -strands are in blue, α -helices in orange. The six positions matched in the reference alignment are represented as spheres. The structural variation between the two proteins results from different relative orientations of the three common subdomains. Image prepared with CHIMERA [86].

variation or flexibility.

Furthermore, in the **SISY** set we find that the accuracy of **PMATRAS** alignments, which are computed based on positive **MATRAS** scores and with a distance threshold, is competitive with the accuracy obtained by **MATRAS** itself. This result indicates that (i) scores for some pairs of inter-residue distances might be irrelevant for a correct alignment and (ii) that we use a very accurate algorithm to compute the structure alignment.

The **SISY** set contains also pairwise alignments with multiple chains. These are excluded from our study because many algorithms handle multiple chains differently or cannot handle them at all. In **PAUL**, we add an option to concatenate several chains, which is reasonable if they correspond to one biological unit. Using this option, we are able to run **PAUL** also on the instances of multiple chains and obtain alignments in very good agreement with the reference alignments.

Looking closer at **PAUL** alignments, we find that they still share a characteristic with alignments that have been computed with contact map overlap scoring: the exclusively positive structural scores encourage to align even remotely structurally similar regions. The alignments tend to be longer and have higher RMSD_c than alignments of other structure alignment methods, as displayed in Figure 5.12. On the other hand, a high RMSD_c is expected when proteins are related by considerable structural variation or flexibility. Figure 5.13 shows scatter plots of RMSD_c values versus alignment accuracy for **PAUL**, **DALI**, **MATRAS**, **MATT**, **SHEBA**, **FATCAT**, **CE**, and **CA** together with the



Figure 5.11: An example from the RIPC data set. The N-terminal domain of Glutathione synthetase from *Escherichia coli* on the left (d1gsaa1) [35], and from human on the right (d2hgsa1) [88]. β -strands are in blue, α -helices in orange. The two proteins show considerable structural variation and several indels are required to align them. The sequence identity is 16%. The five positions matched in the reference alignment are represented as ball-and-stick, the rigid body superposition of their C_α atoms has an $RMSD_c$ of 2.05 Å. PAUL misaligns only one position (marked with arrow), but the shift is only one residue in the sequence. The proteins are viewed in the same orientation as given by the superposition according to the reference alignment. Image prepared with CHIMERA [86].

corresponding Pearson and Spearman correlation coefficients. According to this figure, $RMSD_c$ does not tend to correlate with alignment accuracy as measured by comparison to reference alignments. $RMSD_c$ is a useful measure of structural similarity, but does not seem to be an adequate indicator of alignment quality.

Our results give a partial answer to a question raised by Pelta *et al.* [85]: Can the performance of DALI be achieved using strategies based on contact maps? On the data sets of our study and with respect to alignment accuracy, the answer is that an increased distance threshold and a more sophisticated scoring function is needed. Protein structure classification will benefit from increased alignment accuracy. For classification, the scoring function can be adjusted to obtain a better accuracy to speed ratio. Furthermore, integrating fast heuristics like the ones of Pelta *et al.*, Jain and Obermayer or Di Lena *et al.* [85, 52, 21] into exact algorithms can help to find high-scoring alignments earlier (even if proving optimality might still be time consuming) which can render the approach competitive also in terms of speed.

We show that the PAUL algorithm and scoring scheme is, in terms of alignment accuracy, competitive with state-of-the-art structure alignment methods. High accuracy and optimality or near-optimality of the computed alignment come at the price of a longer run time in comparison to heuristic methods. PAUL is therefore not competitive with faster methods when performing extensive structure comparison over a database. Instead our program should be applied when there is a need for a high-quality pairwise

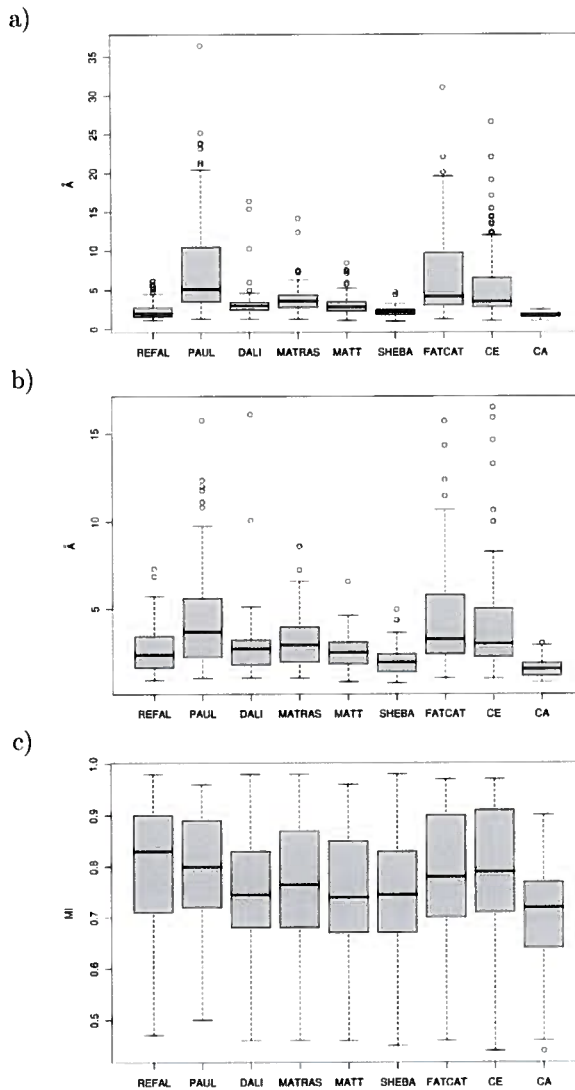


Figure 5.12: Box plots of $RMSD_c$ and normalized $RMSD_c$ values for the consolidated SISO set structure alignments produced by different methods. $RMSD_c$ has been adjusted for different alignment lengths using two different normalizations. Ordering of the box plots according to decreasing median alignment accuracy. REFAL denotes the reference alignments. a) Box plots of $RMSD_c$ values. b) Box plots of the normalized $RMSD_c$ according to $RMSD_{100}$ [15] of the 67 consolidated SISO set alignments for which all computed alignments as well as the reference alignment have length of more than 40 residues (according to [15], $RMSD_{100}$ should only be applied to alignments of length greater 40). c) Box plots of MI (match index) according to Kolodny *et al.* [61], which is 1-OMI (with OMI the original match index of Kleywegt and Jones [60]). Smaller values are considered better.

5.1. PAUL optimization and evaluation

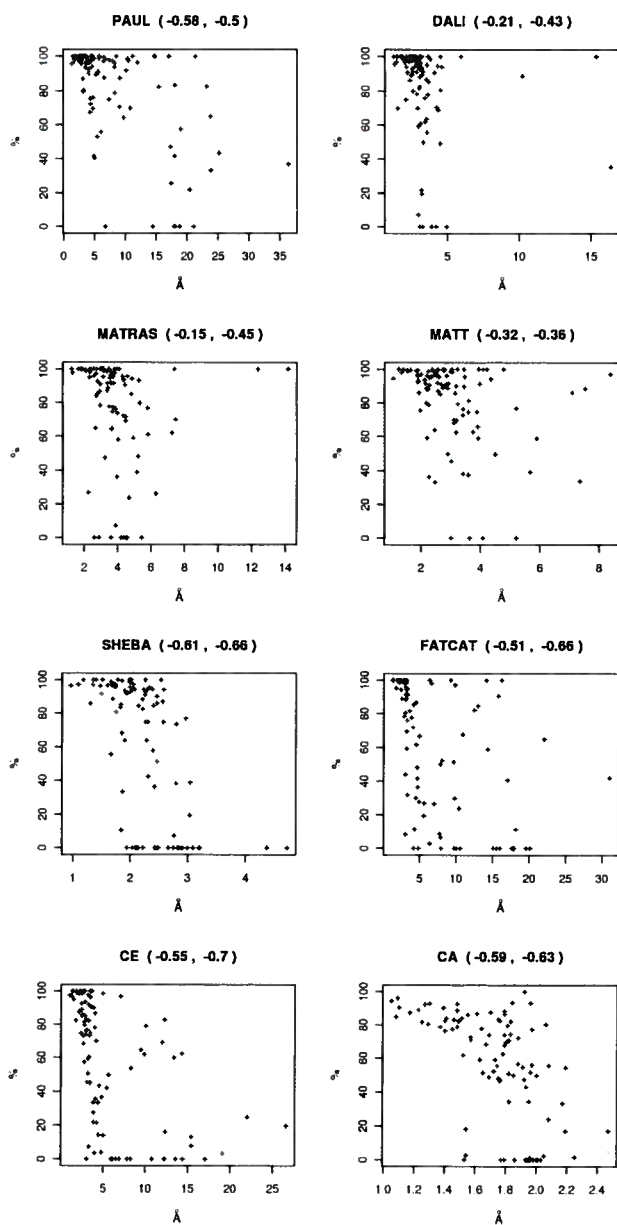


Figure 5.13: Scatter plots of RMSD_c values versus alignment accuracy of **SISY** alignments computed by different structure alignment methods. For each method, the corresponding Pearson and Spearman correlation coefficients are given in the title of the graph. All methods show a poor correlation between RMSD_c and alignment accuracy.

structure alignment, where a run time of minutes is acceptable.

Our study demonstrates that using combinatorial optimization and a sophisticated scoring function we can compute very accurate and in many cases score-optimal alignments. Although the current approach works well, it is only a first step towards a general ILP-based approach for optimal structure alignment of complete protein inter-residue distance matrices. On the way to such a generic method, there are three major challenges. First, we have to compute more alignments to optimality. Second, an algorithm has to be developed that can effectively handle negative structural scores. Third, we need to be able to handle more or even all pairs of inter-residue distances. All these three points were subsequently successfully addressed during the work performed for this thesis. We achieved this by switching to different mathematical models and algorithms as described in Sections 3.5.1 and 3.5.3. The corresponding results are presented in the next section.

5.2 Optimal distance matrix alignment using DALI scoring

In this section we present the computational results that are obtained using two different mathematical models and algorithms presented in Chapter 3: branch-and-cut as well as a Lagrangian approach within a branch-and-bound framework. In Section 5.2.1 we evaluate the efficiency and importance of variable elimination, a preprocessing step that has been described in Section 3.6. In the following Section 5.2.2, we apply the cutting plane approach of Section 3.5.1 in order to obtain score-optimal alignments of entire inter-residue distance matrices for the very first time. For the corresponding computational results we use the DALI scoring function. In Section 5.2.3, we improve these results by applying the model and Lagrangian approach of Section 3.5.3 within the branch-and-bound framework of Section 3.7. This allows us to compute many more optimal distance matrix alignments using DALI scoring and to benchmark the widely-used heuristic DALI program for the first time.

During our work on PAUL we identified three challenges when moving towards general protein structure distance matrix alignment. First, all $\binom{n_A}{2}\binom{n_B}{2}$ distance pairs must be considered. In the Lagrangian relaxation of Caprara *et al.*, which has been used for PAUL, this amounts to $\binom{n_A}{2}\binom{n_B}{2}$ Lagrangian multipliers; too many to fit problem instances of medium or large size into memory. We tackled this problem by switching first to a cutting plane method (Section 3.5.1) and later to the Lagrangian approach of Andonov *et al.* (Section 3.5.3). The latter one only has a cubic number $O(n_A^2 n_B)$ of Lagrangian multipliers ($n_A \geq n_B$). While aligning large proteins is still infeasible if all distance pairs are considered, this approach can handle domains of average size without memory problems. Further, we apply a preprocessing that eliminates alignment graph nodes and corresponding ILP variables that provably can not be part of the optimal alignment. Variable elimination decreases the problem size for similar proteins dramatically.

Another challenge was the integration of negative structural scores. Back then, both

5.2. Optimal distance matrix alignment using DALI scoring

to-date mathematical models and exact algorithms for CMO (from Caprara *et al.* and Andonov *et al.*) could not handle negative structural scores. Therefore, we extended them with additional, so-called activation constraints. These activation constraints are then separated as cutting planes or relaxed in a Lagrangian way. Finally, we embedded the Lagrangian approach into a branch-and-bound framework. This improved the overall performance of the corresponding algorithm; more instances were computed to optimality.

This section presents the results of these combined efforts. Only by putting all the pieces on which we worked during this thesis together, we are finally able to compute many distance matrix alignments to optimality. We consider in the following the most general and most challenging experimental setting: aligning entire distance matrices using positive and negative structural scores. For this task, we use the DALI scoring function (see Section 4.1.2). DALI (Distance matrix ALIgment) is one of the most widely used structure alignment heuristics [41]. It is available via the EBI structural analysis tool box, a dedicated server processes about 1500 pairwise alignment user requests a month, and the first DALI paper has been cited almost 3000 times, more often than any other structure alignment program. Including closely related and follow up papers, DALI was cited more than 5000 times. In the following we present our results on computing, for the very first time, DALI alignments to provable optimality.

5.2.1 Importance of variable elimination

Introduction. Real-life problem instances for aligning complete inter-residue distance matrices are huge, consisting of $n_A n_B$ x -variables and $\binom{n_A}{2} \binom{n_B}{2}$ y -variables. Especially for large proteins we therefore have to rely on an effective preprocessing. For this task, we presented in Section 3.6 a variable elimination scheme that reduces the number of variables that have to be considered explicitly in the ILP model. The effectiveness of this preprocessing step depends on the similarity of the two proteins. If we apply preprocessing to two identical proteins, for example, only the x -variables denoting identical residues and the y -variables of corresponding pairs of distances remain.

Experimental setup. We experimentally evaluate the relationship between structural similarity and rough alignment graph node elimination efficiency. Therefore, we use the 780 all *vs.* all protein pairs of the 40 proteins of the SKOLNICK clustering data set, which belong to five different SCOP families (see Table 4.2). In order to assess structural similarity and cluster the proteins of the SKOLNICK benchmark, we use the heuristic DALI program to compute for each pair of proteins an alignment. The pairwise similarity score is then the empirical DALI z-score of the alignment; it is based on the alignment's DALI score (for details, see Section 4.1.2). Additional to DALI z-score, we use the percentage of eliminated alignment graph nodes as pairwise similarity score.

Results. According to Holm *et al.* [39], a z-score above 8 yields good structural superpositions. In the SKOLNICK data set there are 164 alignments with z-score greater than 8. They all correspond to pairs of proteins from the same family and are considered

Chapter 5. Computational results

“easy” instances in [4]. All alignments between pairs of proteins from different families have a z-score of less than 4 (or DALI detects no similarity). This gap between alignments with high z-score and alignments with low z-score is promoting the well-known fact that the SKOLNICK data set is a rather easy benchmark for clustering [21]. As expected, DALI thus correctly identifies the five SCOP families.

The percentage of x -variables excluded during node elimination highly correlates with structural similarity as measured by DALI z-score of the alignment. The Pearson correlation coefficient is 0.91. A scatter plot of the percentage of eliminated alignment graph nodes *vs.* z-score is displayed in Figure 5.14. The percentage of eliminated x -variables can even be used successfully as a pairwise similarity score for a correct classification of the SKOLNICK data set, see Figure 5.15. These findings indicate that the more similar two proteins are, the easier it is to align them optimally—an observation first made with exact CMO alignment algorithms.

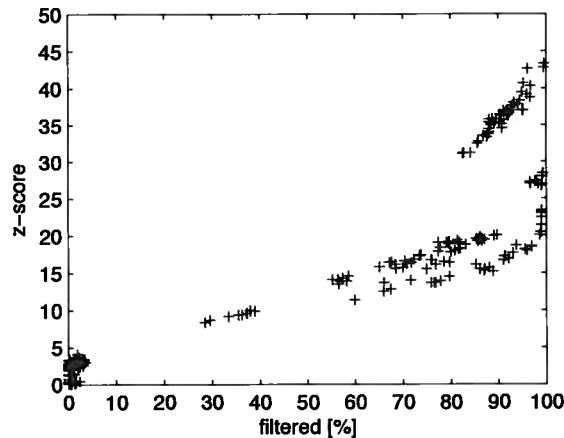


Figure 5.14: A scatter plot of the percentage of filtered alignment graph nodes *vs.* DALI z-score for the 780 all *vs.* all protein pairs of the SKOLNICK data set. The more similar two proteins according to z-score, the larger is usually the percentage of nodes and corresponding variables that can be eliminated.

Conclusion. Variable elimination has to be effective, firstly in order to reduce the model such that it fits into memory and secondly such that we do not have to add too many cutting planes or compute too many Lagrangian iterations. Independent of protein size, this is oftentimes feasible for the class of alignments that leads to good structural superposition (*e.g.*, DALI z-score greater than 8). For pairs of proteins with more subtle structural similarities, for example if proteins share only local similarities, preprocessing is less efficient or may even fail. In the case of poor preprocessing, the subsequent structure alignment algorithm has to consider almost the entire ILP model. The length limit of instances that can be computed then depends on the algorithm itself, its memory requirements and performance. The branch-and-cut approach of

5.2. Optimal distance matrix alignment using DALI scoring

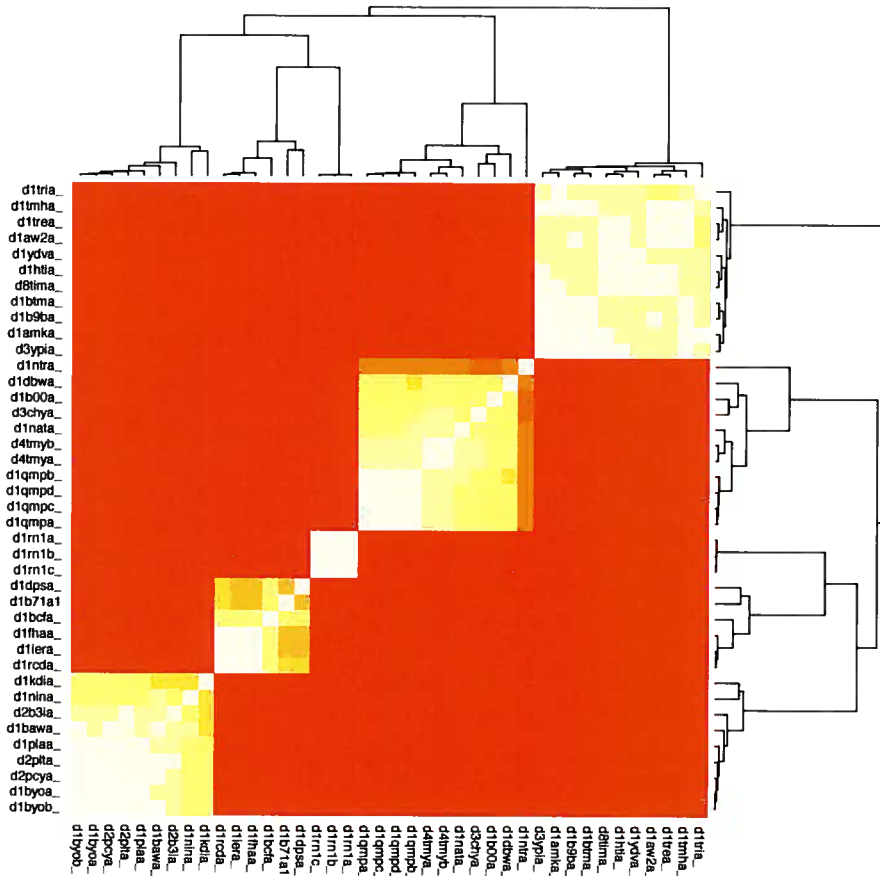


Figure 5.15: The percentage of filtered alignment graph nodes, *i.e.*, x -variables, is used as pairwise similarity score for clustering the SKOLNICK data set. The five SCOP families from Table 4.2 can be clearly distinguished from each other.

Chapter 5. Computational results

Section 3.5.1 or the Lagrangian approach based on variable splitting of Section 3.5.2, for example, run into memory problems for much shorter proteins than the Lagrangian solver of Section 3.5.3.

5.2.2 Branch-and-cut and preprocessing

Introduction. In this section we present computational results for the mathematical model (3.4)–(3.10) of Section 3.4 and the corresponding branch-and-cut approach described in Section 3.5.1 and summarized in Algorithm 4. Its implementation has been specified in Section 3.8.1. After introducing the experimental setup, we present the very first optimal structure alignments using the DALI scoring function. In a subsequent conclusion we summarize the significance of these results within a greater context and illustrate the biological benefit and algorithmic difficulties of using DALI scoring instead of basic CMO scoring.

Experimental setup. We evaluate the branch-and-cut approach on the SKOLNICK data set, a benchmark for clustering proteins (see Section 4.2.3). It consists of 40 protein chains of length between 97 and 255 residues that belong to five different SCOP families. DALI skips hetero atom records and residues with incomplete backbone. Therefore we edited 7 PDB files of the SKOLNICK data set; we changed heteroatom records to atom records and excluded residues that are for other reasons ignored during DALI computation. Furthermore, in order to be in line with DALI, we consider only the first decimal place of C_α atom coordinates. The remaining computations are carried out with double precision leading to slight differences between the overall score reported by DALI and the recomputed score. We focus in our subsequent analysis only on the 164 SKOLNICK alignments of proteins from the same family, all with z-score greater than 8. The poor performance of variable elimination for dissimilar proteins renders it currently infeasible to fit the remaining problems into memory.

Furthermore, we use alignments from SISY, a more challenging data set that has been designed with the objective to provide difficult structure alignment instances. These alignments are individually inspected by experts and essentially manually created; therefore we consider them gold-standard reference alignments. Refer to Section 4.2.3 and Table 4.3 for a detailed specification of the SISY data set.

In order to compute score-optimal DALI alignments using our branch-and-cut approach, we use CPLEX version 12.1 and a maximum run time of 30 hours for each alignment. Alignments have been computed on cluster nodes each equipped with two quad core 2.26 GHz Intel Xeon processors and 24 GB of main memory running 64 bit Linux.

First optimal DALI alignments. Our branch-and-cut approach solves 75 of the 164 SKOLNICK alignments (46%) to provable optimality. The run times for the solved instances vary between 24 and 70296 seconds and depend on the number of y -variables after variable elimination. The biggest solved instance has little less than 3 million y -variables. For 23 protein pairs (14%), CPLEX runs out of memory before the time

5.2. Optimal distance matrix alignment using DALI scoring

limit is reached. Those instances have more than 15 million y -variables, *i.e.*, pairs of distances.

For 32 of the solved **SKOLNICK** instances (43%), the heuristic solution provided by the DALI heuristic was proven to be optimal. In the remaining instances, the optimal solution improved the heuristic solution slightly (less than 2% improvement in DALI score). With improvements in this order of magnitude, DALI might not fail to produce the optimal alignment, but determine a different alignment because it possibly uses different precision during computations. Furthermore, the compared **SKOLNICK** proteins are very similar in length and structure and the optimal alignment usually globally aligns them with a few short gaps. Therefore a good performance of the heuristic DALI program is expected. We assume that in more difficult instances, in which the optimal alignment is not global, there is more room to improve on the heuristic DALI solution. We will in fact observe such examples in the following Section 5.2.3.

We use small **SISY** alignments in order to evaluate our branch-and-cut approach on more difficult instances. Four structure alignments are solved optimally. We prove the optimality of the heuristic DALI solution for two of them and for two improve with our exact solution the heuristic DALI score slightly (by less than 1%).

Conclusion. The branch-and-cut approach presented in Section 3.5.1 and published in [113] was the first exact algorithm that offers the feature of penalizing the alignment of non-compatible distances, *e.g.*, a small and a large distance. Furthermore, it illustrates for the first time that it is feasible to compute provably score-optimal alignments of complete inter-residue distance matrices by means of preprocessing followed by a branch-and-cut approach that is implemented within a general-purpose ILP solver. We were able to compute DALI alignments to optimality, demonstrating that this popular and widely used heuristic structure alignment method generates optimal or close-to-optimal alignments, at least for relatively similar and relatively small problem instances.

For global alignments of structurally very similar proteins, *e.g.*, **SKOLNICK** instances, the shift from a simple scoring function like **CMO** to a more sophisticated scoring of all distances using for example the DALI scoring function, shows little effect. Nonetheless, our evaluation of the complete **SISY** data set in the previous section on **PAUL** illustrates that for detecting more complex structural similarities, basic scores such as **CMO** are not always sufficient to obtain gold-standard reference alignments, and the DALI scoring function performs on average better [116].

The branch-and-cut approach is currently successful only in the cases of similar proteins, in which many variables are eliminated by preprocessing. Although in the next section we compute many more score-optimal structure alignments using our Lagrangian approach, we believe that it is not realistic to provide exact solutions for all instances. On the other hand, all our results show that it is not necessary to consider all pairs of inter-residue distances in order to obtain good alignments. Methods from molecular distance geometry [121], for example, can uniquely reconstruct a protein's 3D structure from a small subset of distances. Alignments of sparse inter-residue distance matrices using for example **CMO** or **PAUL** scoring might thus perform promis-

ing because they capture to a large extent the protein's structure. Nonetheless, we observed in the last section that it is essential to penalize non-compatible distances in order to avoid overaligning, for example by assigning them a negative score. An illustration is given in Figure 5.16: the CMO scoring function greedily aligns as many residues as possible, which leads to an increase of alignment length at the expense of precise structural similarity of compact substructures as measured by RMSD_c .



Figure 5.16: Alignment of 1aawA (gray) and 1gxiE (pink), an instance of the **SISY** set. Optimal superposition according to the respective alignment. Residues colored in dark tone are aligned, residues colored in light tone are unaligned. Left: The **SISY** reference alignment (29 aligned residues, RMSD_c of 1.14). Middle: The heuristic **DALI** alignment correctly aligns all residues of the reference alignment, but extends the alignment length to 50 (RMSD_c of 2.55). Right: The optimal **CMO** alignment; it correctly aligns 96.55% of the aligned residues of the reference alignment. Alignment length is 56, RMSD_c 4.25. Additional gaps are inserted. Overaligning and insertion of additional gaps leads to a low RMSD_c value.

Because of these observations, we would like to move towards structure alignment scoring schemes that reduce the size of the model by excluding pairs of distances. A first successful, yet still non-negative, scoring scheme is implemented in **PAUL** [116]. It was discussed in detail in Section 4.1.3 and evaluated in Section 5.1. Secondly, in the case of sparse distance matrices our Lagrangian solver of Section 3.8.3, whose results are presented in the next section, is often able to provide score-optimal alignments even for very large and dissimilar proteins. In future work, we therefore would like to come up with hypotheses which pairs of distances can be excluded without losing alignment precision. Computing alignments to optimality, we can then objectively test these hypotheses.

5.2.3 DALIX: Optimal distance matrix alignment

Introduction. In this section we use the Lagrangian approach of Section 3.5.3 within the branch-and-bound framework described in Section 3.7. We call this implementation of our exact algorithm **DALIX** (see Section 3.8.3). It succeeds to handle larger problem instances than the branch-and-cut approach of the previous section and solves many more distance matrix alignments to optimality. We compare **DALIX** alignments

5.2. Optimal distance matrix alignment using DALI scoring

with the alignments computed by the heuristic DALI program. The improved performance of DALIX with respect to the branch-and-cut approach of the last section allows us to benchmark the DALI heuristic program for the first time on large data sets of diverse structural similarity. For this evaluation, we use (i) alignments of SCOPCATH domains with lengths between 30 and 50 residues who share family, superfamily or fold, (ii) alignments from SKOLNICK of proteins from the same family, and alignments from (iii) the SISY and (iv) RIPC collections. For details on the data sets refer to Section 4.2.3. We find that DALI is very reliable in returning a good alignment according to DALI scoring. Although we detect many cases where the DALI alignment is not score-optimal, the gap between the heuristic and the optimal objective function value is often negligible. However, we find that when aligning short protein domains, the heuristic misses to detect quite a few significant similarities and wrongly concludes that no such similarity exists. This DALI deficiency has not been reported before in the literature. We also evaluate the weak points of our exact algorithm DALIX, which are: (i) large proteins and (ii) subtle local structural similarities. In these cases, the (then suboptimal) alignment returned by DALIX is often worse than the heuristic alignment returned by DALI.

Experimental setup. The SKOLNICK data set consists of 40 proteins with length between 97 and 255 residues belonging to five protein families. For details on the data set refer to Section 4.2.3 and Table 4.2. We align only protein pairs from the same family, which amounts to 164 SKOLNICK instances.

SCOPCATH is a benchmark containing domains that are consistently classified in SCOP [version 1.75] and CATH [version 3.2.0] and that have a pairwise sequence similarity of less than 50% (*cf.* Section 4.2.3). We align all SCOPCATH domains with 30 up to 50 residues which belong to the same family (386 pairs), to different families but to the same superfamily (151 pairs), and to different superfamilies but the same fold (926 pairs). According to the DALI tutorial, protein chains must have a minimum length of 30 residues, which we therefore also inflict. Further, we limit the length to maximally 50 residues to obtain alignments for which our algorithm can explore multiple branch-and-bound nodes within a few minutes.

SISY and RIPC are data sets of manually curated pairwise structure alignments, which are difficult for alignment programs because of repetitions, large indels, circular permutations, conformational variability, *etc.* They consist of 98 and 22 alignments, respectively. For details on SISY and RIPC refer to Section 4.2.3 and Tables 4.3 and 4.4.

In summary, we assess the capability of our algorithm to compute score-optimal alignments with respect to the DALI scoring function on (i) 164 SKOLNICK alignments, (ii) 1463 SCOPCATH alignments, (iii) 98 SISY alignments and (iv) 22 RIPC alignments. On the two latter data sets, we also evaluate the alignment accuracy with respect to the manually curated reference alignment, which is defined as the percentage of correctly aligned residue pairs. We compare our alignments and their scores to those determined by the DALI program. Let S_{DALIX} be the DALI score of the DALIX alignment and S_{heur} the DALI score of the heuristic DALI alignment. We then evaluate the improvement in DALI score of our DALIX alignments compared to the heuristic DALI alignments with

Chapter 5. Computational results

respect to the score of the DALI alignment. More precisely, we use for the improvement in percent the formula $100|S_{\text{DALIX}} - S_{\text{heur}}|/S_{\text{heur}}$ and consider cases with $S_{\text{DALIX}} > S_{\text{heur}}$ and with $S_{\text{heur}} > S_{\text{DALIX}}$ independently.

For comparison with DALI we use DaliLite version 3.3. DALI computes a number of alignments and ranks them according to the z-score of protein unfolding units. Therefore, we parse all alignments returned by DALI and consider the one with largest DALI score. Note that maximizing the DALI score will also maximize the DALI z-score of the entire alignment. Nonetheless, in an attempt to report and rank interesting local similarities high, DALI also computes z-scores for parts of the alignment, the protein unfolding units. Because of this, a suboptimal alignment can receive the highest z-score.

In an attempt to handle the memory requirements of especially large proteins, we additionally make a SISY and RIPC evaluation in which we exclude inter-residue distances between residues that belong to different domains. The orientation of domains to each other is thus entirely disregarded during alignment. As a result, alignments are scored on the base of different criteria, and thus the optimal alignments in both cases may differ. We evaluate empirically the influence of omitting inter-domain distances on the alignment accuracy.

We compute pairwise alignments on cluster nodes each equipped with two quad core 2.26 GHz Intel Xeon processors and 24 GB of main memory running 64 bit Linux. In each branch-and-bound node we compute 1000 Lagrangian iterations. For SISY, RIPC and SKOLNICK, a maximum run time of 30 CPU hours per instance is applied and for the short SCOPCATH instances a time limit of 30 CPU minutes per instance.

Results for SKOLNICK. DALIX computed 136 of the 164 SKOLNICK alignments (83%) to provable optimality within the time limit. For 123 alignments (75%), the heuristic DALI solution was improved, but at most by 3%, and for 38 alignments (23%) the heuristic solution was proven to be optimal. Only three DALI alignments are slightly better than the corresponding DALIX alignment, see also Table 5.6, column SKOLNICK. Results for SKOLNICK indicate that DALI computes optimal or close to optimal alignments in the case of distinct structural similarities on family level. The results also demonstrate that it is feasible to compute structure alignments to optimality in the case of considerable structural similarity.

Results for SCOPCATH. When aligning the short SCOPCATH domains, for 661 (45%) neither DALI nor DALIX could compute an alignment with positive z-score, especially on fold level. It is likely, but unfortunately not proven by our upper bounds, that when using DALI scoring no such significant alignment exists in many cases. This situation illustrates that it is difficult to design a scoring scheme and algorithm that reliably detects gold-standard structural similarities on different classification levels, ranks them correctly and discriminates them from spurious similarities. Structure alignment approaches and their scoring schemes are often benchmarked using these criteria: The scores of their structure alignments should reproduce the SCOP hierarchy. A “perfect” scoring function would assign a significant score to protein pairs related on family, superfamily or fold level. In doing so, the score would decrease from family to

5.2. Optimal distance matrix alignment using DALI scoring

	SKOLNICK	SCOPCath		SISY	RPC	
		Family	Superfamily	Fold		
Alignments	164	386	151	926	62	11
Positive z-score	164	359	141	302	61	11
DALIX optimal	136	143	14	31	11	2
DALI optimal	38	50	5	5	3	0
DALIX better	123	287	118	258	31	6
DALI better	3	16	14	30	27	5
Missed by DALI	0	83	24	123	0	0

Table 5.6: Comparison of DALIX and DALI alignments. The table lists the number of alignments in each data set in the first row and the number of alignments for which either DALIX or DALI detected an alignment with z-score greater 0 in the second row. Given that only alignments with z-score greater zero are significant and that DALI reports only significant alignments, we only consider those alignments in the following table rows. “DALIX optimal” denotes the number of DALIX alignments that have been computed to optimality and “DALI optimal” the DALI alignments thereof that are also provably score-optimal. “DALIX better” lists the number of DALIX alignments with higher DALI score than the DALI alignment. “Missed by DALI” is the number of alignments thereof, for which DALI did not return an alignment at all. “DALI better” denotes the number of DALI alignments that are better than the (not yet optimal) DALIX alignment. The DALIX computation time limit for SCOPCath alignments is 30 CPU minutes and for all other data sets 30 CPU hours.

Chapter 5. Computational results

superfamily to fold level. Protein pairs that are not related on any of these levels would not obtain a significant alignment score. In practice, probably no scoring function can meet these requirements for any protein pair. Further it is hard to evaluate whether an alignment score for a structurally related protein pair is too low because the scoring function inadequately ranks the present structural similarity or because the algorithm fails to report the top-scoring alignment.

In summary, an exact algorithm maximizing a “perfect” scoring function would return a significant alignment for all our instances from the SCOPCATH benchmark. Given the DALI score and the DALI and DALIX algorithms, this is not the case. We thus exclude protein pairs from the analysis for which no algorithm returns an alignment with positive z-score.

From the 359 short SCOPCATH alignments of domains from the same family, 143 (40%) are solved to optimality within the time limit of 1800 seconds, and most within a few seconds. Table 5.6 and Figure 5.17 show the differences between DALIX alignments and DALI alignments. First, DALI fails to detect 83 significant alignments with z-scores up to 5 and falsely reports that there are no structural similarities. DALIX computes 143 alignments (40%) to optimality. From these cases, in which the provable top-scoring alignment is known, DALI returns 50 optimal and 93 close to optimal alignments. Altogether, our exact algorithm improves the heuristic solution in 287 cases (80%). The percentages of score improvement with respect to the score of the DALI alignment are visualized in Figure 5.17. For almost all instances, the score improvement is small. It is on average 7% for the instances in which DALIX improves the alignment and DALI does not miss a significant alignment altogether. In 16 cases, the solution returned by the exact algorithm after 1800 seconds is worse than the heuristic solution.

Computing exact alignments becomes more difficult when structural similarity gets less pronounced, see Table 5.6. For the 141 SCOPCATH alignments of proteins that share the same superfamily, but not the same family, only 14 (10%) are computed to optimality within 1800 seconds. Nonetheless, the exact algorithm improves the heuristic alignments in 118 cases (84%), as visualized in Figure 5.17. In 24 of them, DALIX returns a significant alignment that is entirely missed by DALI. The average score improvement on the instances for which DALIX is better than DALI and DALI returns an alignment increases in comparison to the family level; it is 10%. In 14 cases, the DALI heuristic alignment has larger DALI score than the DALIX alignment.

On fold level, only 31 of the 302 alignments are computed to optimality, see Table 5.6. 258 alignments (85%) returned by the exact algorithm are better than those returned by the heuristic algorithm. The average score improvement of these instances (without those for which DALI does not return an alignment) is 10%. Also here, the DALI score for alignments produced by DALI or DALIX is usually very similar, see Figure 5.17, but the exact algorithm detects significant alignments with z-scores up to 2.5 that are missed by the heuristic. In 123 cases (41%), DALIX determines a significant alignment that is missed by DALI. In 30 cases the DALI alignment is better than the one returned by our algorithm.

Figure 5.18 visualizes the alignment traces of three alignments, from family, superfamily or fold level, respectively. These are the instances for which the DALIX alignment im-

5.2. Optimal distance matrix alignment using DALI scoring

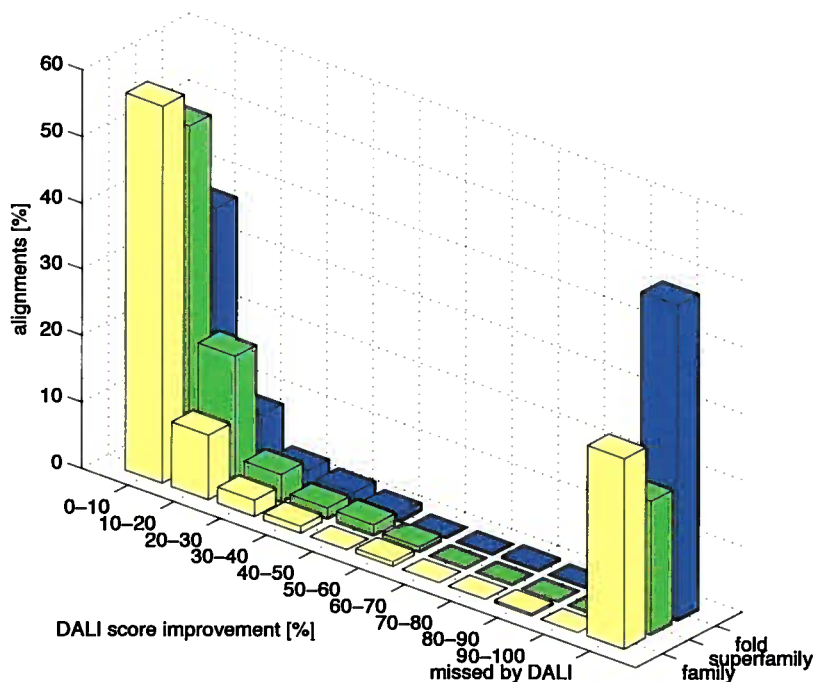


Figure 5.17: For the SCOPCATH data set, the bar plot bins the percentages of DALI score improvement for the cases in which the DALIX alignment has positive z-score and is better than the DALI alignment. On family level, these are 278, on superfamily level 118 and on fold level 258 alignments. The improvement is computed with respect to the DALI alignment. The DALIX computation time limit is 30 CPU minutes. For most alignments, the score improvement is small. There is furthermore a large percentage of protein pairs that are entirely missed by DALI, *i.e.*, for which DALI falsely reports that there is no structural similarity.

proves the DALI alignment with the largest relative score difference from all alignments of the respective similarity level. A comparison of the DALI and DALIX alignments using our web sever CSA [117] reveals that in all three cases the DALIX alignment is clearly preferable not only according to DALI score, but also with respect to all eight other computed quality measures and scoring schemes. For example, $RMSD_c$ and $RMSD_d$ of the DALIX alignments are smaller although DALIX aligns the same number of residues (d1b9wa2 *vs.* d1dx5j3) or even more residues than DALI (d2glia1 *vs.* d2glia4, d1pfwa3 *vs.* d1ryta2). These three cases illustrate how heuristics can miss good structure alignments.

Results for SISY and RIPC. If structural similarities are less pronounced or locally confined, determining the score-optimal alignment becomes inordinately more difficult.

Chapter 5. Computational results

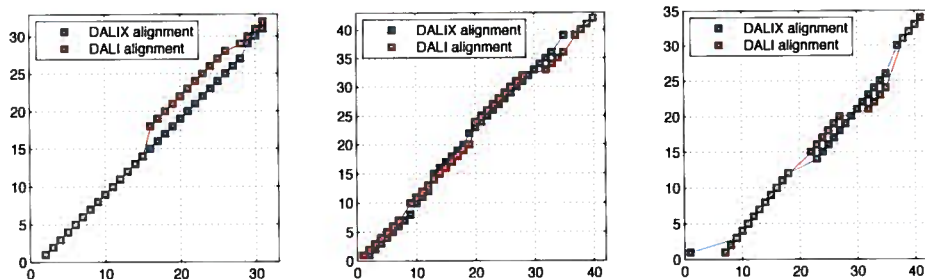


Figure 5.18: Three **SCOPCATH** examples in which the DALIX alignment improves the DALI alignment significantly. Proteins from common family, superfamily and fold are aligned (**SCOPCATH** IDs *d2glia1 vs. d2glia4*, *d1b9wa2 vs. d1dx5j3*, and *d1pfwa3 vs. d1ryta2*). Each alignment has the largest percentage of score difference from the respective level of similarity with respect to DALI score (82%, 54% and 90%). Residues aligned by DALIX are colored blue, residues aligned by DALI red and residues aligned by both methods gray. The three DALIX alignments are better than the respective DALI alignments according to all scoring schemes evaluated by **CSA** [117]. For example, the RMSD_c of *d2glia1 vs. d2glia4* is 2.0 (30 aligned residues) compared to 3.0 (29 aligned residues), the RMSD_c of *d1b9wa2 vs. d1dx5j3* is 2.9 (35 aligned residues) compared to 3.0 (35 aligned residues), and the RMSD_c of *d1pfwa3 vs. d1ryta2* is 2.1 (28 aligned residues) compared to 3.1 (26 aligned residues).

Furthermore, long proteins are problematic for our algorithm since the number of distance pairs grows in the order of $O(n_A^2 n_B^2)$. From the **SISY** set, whose difficulty is also confirmed by low alignment z-scores, only 62 alignments fit into memory. From these, 11 are solved to optimality in, on average, about 6000 seconds. Three of these optimal alignments are also detected by the DALI heuristic, and the remaining 8 non-optimal DALI alignments are less than 3% worse than the optimal ones. Altogether, our exact algorithm improves the DALI alignments in 31 cases by an average of 2%. For 27 instances, DALI performs by an average of 39% better than our algorithm. The reason is that the instances in the **SISY** set are large and of subtle similarity, which is both disadvantageous for our algorithm.

From the 22 **RIPC** alignments, 11 fit into memory. Of those, two are computed to optimality. An improvement of average 17% over the DALI solution was found in 6 cases. The heuristic solution is better than the solution of our algorithm for 5 alignments, with a large average DALI score difference of 71%. In 3 cases DALIX does not detect an existing alignment with positive z-score.

On **SISY** and **RIPC** we observe that, within the given time limit, our exact algorithm fails to produce good alignments for large or remotely similar protein pairs. Furthermore, we find that improving the DALI score does not necessarily implicate a higher alignment accuracy with respect to manually curated reference alignments: 13 DALIX

5.2. Optimal distance matrix alignment using DALI scoring

alignments from **SISY** have larger DALI score than the DALI alignment, but slightly less alignment accuracy with respect to the reference alignment. Only for three alignments an increased DALI score also results in a slightly increased alignment accuracy. For the **RIPC** data set, in two cases an improved DALI score increases the alignment accuracy, but in two other cases alignment accuracy decreases. Inspection of a few of these instances using **CSA** [117], our web server for comparative structural alignment, shows that in **DALIX** alignments with larger DALI score but less alignment accuracy often short, additional gaps have been inserted rendering the alignment more scattered and less intuitive than the respective DALI alignment.

We evaluate the benefit of preprocessing as described in Section 3.6. Four more **SISY** and one more **RIPC** alignments fit into memory. Two of these **SISY** alignments are computed to optimality. In most cases the alignment returned when additional preprocessing is used is better with respect to DALI score than the one obtained without preprocessing. In those instances that are solved, the overall running time including preprocessing is in most cases significantly smaller than without preprocessing, although the preprocessing itself takes, depending on protein length, up to 13 minutes. Similar observations hold for the **RIPC** alignments. Here, one previously unsolved protein pair is aligned optimally when preprocessing is used, in as little as 91 seconds.

Large proteins that share only little structural similarity cause memory and performance problems for our algorithm. It was thus only possible to fit 62 **SISY** (63%) and 11 **RIPC** (50%) alignments into memory. In order to reduce memory consumption while keeping alignments very close to the ones computed by the DALI program, we made another **SISY** and **RIPC** evaluation in which we excluded all inter-domain distances. Now, 92 **SISY** (94%) and 19 **RIPC** (86%) alignments fit into memory. While it allows to compute alignments for larger proteins and decreases computing time, omitting inter-domain distances has little influence on alignment accuracy with respect to the manually curated **SISY** and **RIPC** reference alignments.

Conclusion. The results of this section, obtained by our latest Lagrangian solver, improve over last section's initial results for general distance matrix alignment using a branch-and-cut algorithm. **DALIX** is therefore the first exact solver for general distance matrix alignment that is applicable to structures of average domain size and from different levels of structural similarity. In contrast to previously designed exact algorithms that consider only particular sub-cases of distance-based scoring functions with positive values, like contact map overlap [4] or **PAUL** [116], our algorithm can use any distance matrix-based scoring scheme. This permitted our implementation **DALIX** to benchmark and evaluate the precision of **DALI**—one of the most popular heuristic structure alignment methods. We found that computing DALI alignments to optimality is feasible for small to medium size proteins, or in case of a clear structural similarity. We observe and describe, as far as we know for the first time, some characteristics of the DALI heuristic. For example, on the **SCOPCATH** benchmark we detected an important subset of protein pairs related on family, superfamily and fold level for which DALI entirely misses structural similarities and wrongly does not return any alignment. Apart from these cases, the exact computations globally confirm the high quality of the DALI heuristic—although DALI alignments are often not score-optimal, they are nevertheless almost always very close to the optimum.

Conclusion

This thesis contributes towards the goal of truly score-optimal alignment of protein structures. To this end we focus on protein structure alignment using 2-dimensional scoring schemes, which are based on inter-residue distance matrix alignment. We aim to find the alignment which maximizes the sum of scores that are assigned to aligned distances. There are many structure alignment heuristics that are based on inter-residue distance matrix alignment (see Section 4.1.2), among them venerable and widely used programs such as DALI. The work presented in this thesis is the first comprehensive approach to compute this class of alignments *optimally* with respect to the given scoring scheme. For this purpose, we cast inter-residue distance matrix alignment into an optimization problem that we then describe with different mathematical models. We solve these models using techniques from combinatorial optimization: Branch-and-cut (Section 3.5.1), Lagrangian relaxation after variable splitting (Section 3.5.2) and an alternative Lagrangian relaxation dualizing constraints directly (Section 3.5.3). Our models and algorithms are either extensions of or inspired by previous work for the special case of structure alignment using contact map overlap [14, 4]. Our two best-performing exact algorithms use iterative double dynamic programming, an approach that has been used before for heuristic protein structure alignment [108, 107, 58, 12]. Most of these programs compute only one iteration of double dynamic programming; If several iterations are used, they perturb structural scores randomly in the hope of obtaining better alignments [107]. Our approach, in contrast, is mathematically sound. We obtain lower and upper bounds on the optimal alignment score and may use them within a branch-and-bound framework. If lower and upper bound coincide, the provably score-optimal structure alignment has been detected.

We tackle questions that can only be answered rigorously by the use of exact algorithms. For example, in Section 4.1.3 we find our own, dedicated scoring scheme, PAUL scoring, which allows us to compute provably better alignments with respect to gold standard alignments, see the results in Section 5.1. Further, in Section 5.2.2 we compute for the very first time provably optimal DALI alignments and compare them with the respective heuristic alignments. Finally, by switching from branch-and-cut within the general-purpose solver CPLEX to a dedicated algorithm using Lagrangian relaxation, we are able to compute many more DALI alignments to optimality, see Section 5.2.3. This allows to benchmark the DALI heuristic for a large number of short proteins and to identify its strengths and weaknesses. We further use our corresponding general Lagrangian solver

for distance matrix alignment to implement four scoring schemes, CMO, PAUL, DALI and MATRAS. They all are available for online computation of alignments via our web server CSA, see Section 4.3. CSA further allows to compare these and other, uploaded structure alignments with respect to many scoring schemes and quality criteria which have been introduced in Section 4.1 and generates visualizations that aid in such a comprehensive comparison. Finally, lower and upper bounds on the optimal score that we obtain from our solver can be used for assessing the similarity of proteins, for example by clustering protein structures into families, superfamilies and folds or by recognizing whether two proteins are related on any of these similarity levels. This is work in progress.

Most of these applications of our exact algorithms are problems that are rigorously approached for the very first time. Hence our results can be viewed as a proof-of-concept and it is desirable that they are still improved in the future. There are two main challenges, both dependent on each other. The first one is to fit into memory and compute larger instances while taking into consideration all inter-residue distances. The second challenge is to decrease run time, especially for large instances. Both are extremely difficult, since the number of distance pairs to consider in an exact algorithm grows with $O(n_A^2 n_B^2)$, where n_A and n_B are the protein lengths. As long as we want to solve the structure alignment problem to optimality, we can therefore not expect to be able to improve run time for all instances.

6.1 Future work

Beside the challenge of generally improving exact algorithms for distance matrix alignment, there are also several more straightforward directions for future research which we summarize in the following.

Using bounds on structure alignment scores for organizing the protein universe.

Malod-Dognin *et al.* [70] describe how bounds computed by algorithms such as the ones introduced in this thesis can be used for efficiently solving problems related to organizing the protein universe. They call the governing idea dominance. It is based on the lower and upper bounds on the similarity scores of a query protein with a number of target proteins. The observation is that while successively improving bounds, at some point the target protein with highest similarity score has a higher lower bound than the upper bounds of all other target proteins. At this point, the target protein with highest similarity to the query protein is provably known, and this even without computing any structure alignment to optimality.

Malod-Dognin *et al.* apply dominance for family identification. Given an existing protein classification, this is the problem of assigning a query protein to the protein family of the target protein with the highest similarity score. In our current work we investigate how the dominance idea can be extended and applied to solve many other problems related to protein classification. For example, we generalize it such that, using only bounds, we can return all target structures that exceed a given similarity score,

Chapter 6. Conclusion

which in turn can be used for hierarchical classification. Further, bounds are beneficial for preprocessing a set of target protein structures such that queries can be assigned more quickly. For example we can use similarity score bounds for any pair of proteins in the same family (or superfamily or fold) to designate one central, representative structure. Then we compare a query structure only to the family's representative and bound the similarity score to other structures of the same family by using the triangle inequality.

Note that no heuristic can solve in a rigorous way problems for which protein similarities implicitly need to be ranked according to some scoring scheme, *e.g.* protein classification. Moreover, heuristic methods need to compute all pairwise structure alignments to obtain the corresponding similarity scores, and this even for solving these problems only in a heuristic way. Using bounds we could therefore not only solve classification problems exactly, but even be faster than heuristic methods. The prerequisite is the use of powerful algorithms that quickly compute tight bounds. The approaches presented in this thesis are examples.

For the applications involving the ranking of structural similarities, we benefit from the properties that are experimentally inherent to real-life structure alignment instances. If a pair of proteins is similar, the similarity score is large and the gap between lower and upper bound small. If, on the other hand, a pair of proteins is dissimilar, the similarity score is low and the gap between lower and upper bound large. Nonetheless, the upper bound tends to be lower than the upper bound for instances of similar proteins. Exploiting this observation, we developed besides the three algorithms presented in this thesis additional algorithms for faster computation of weaker bounds. These determine various bounds whose quality varies with the time needed for their computation. Here we again apply double dynamic programming, but instead of computing precise structural profits in the lower level DPs, for some residue pairs these profits are only estimated. Further, we use a technique called banded alignment which allows to compute alignments more quickly under loss of the optimality of the solution in some cases.

Implementing established and novel scoring schemes. Using our existing framework, we can implement additional distance matrix-based scoring schemes like the ones of SSAP and VOROLIGN, allowing us to benchmark the corresponding heuristics. Also linear combinations of several scoring schemes are possible.

Similar as we did with PAUL, we further would like to investigate novel scoring schemes that exclude many distance pairs from the computation and that take also negative structural scores into consideration. The difficulty here is that an omitted distance pair has implicitly the score zero, and we can thus only omit distance pairs that are neither good nor bad for the alignment. Such neutral distance pairs are intuitively rather rare, and we thus have to consider again a large number of distance pairs in our scoring scheme. Along these lines, it would be interesting to identify distance pairs that are good, bad or neutral for a correct alignment, for example by machine learning. In a new, dedicated scoring scheme, good distance pairs are then rewarded by positive and bad distance pairs penalized by negative structural scores.

Algorithms for SSE alignment and protein threading. Another line of research is the application of (sequential) distance matrix alignment to other problems that can be cast into this framework. For example we can use the existing approach and implementation also for the alignment of secondary structures using the so-called Tableau representation [67].

With a few modifications, also the local protein threading problem [17] can be cast into our framework, which we consider an interesting subject for a larger future project. Protein threading is a method for protein structure prediction from sequence. It aligns a query sequence for which the structure is unknown to a set of target structures, aiming to maximize a scoring function that measures how well sequence and structure match. Such a scoring can for example use information about secondary structure and solvent accessibility of residues in the target structures [72].

Because of the large number of available sequences and comparably limited number of solved structures, protein threading is an active field of research with a large focus on the optimized scoring functions. Many sophisticated scoring schemes have been developed. The instance sizes for threading are typically smaller than for structure alignment because blocks of residues, *e.g.*, SSEs, are considered simultaneously. We are therefore optimistic that exact algorithms can compute score-optimal solutions in a representative number of cases. These can then be used to benchmark and compare scoring functions for threading in a similar fashion as described for structure alignment in this thesis.

Clustering distance- and distance difference matrices for domain and hinge detection. Distance matrix-based, 2-dimensional scoring schemes tend to produce alignments with larger RMSD_c as superposition-based, 3-dimensional scorings. We therefore would like to detect regions of large RMSD_c and possibly accommodate for them in the visualization of the superposition by introducing breakpoints in the chain. This can be done by investigating the corresponding distance difference matrices. Clustering the rows of the distance difference matrix will highlight regions of common distance differences which denote rigidly superposable fragments. Such an approach has been used before for the detection of flexible regions and hinges [26].

Domains in a protein can be detected in a similar way [42], because a domain may be characterized by residues with small inter-residue distances to each other and large inter-residue distances to the remaining residues of the protein. Also here, exact approaches for matrix clustering can be used [10, 16]. Programs that implement algorithms for domain detection as well as for better superpositions and the identification of flexible regions and hinges would be a useful extension of our current software tools. Further, using similar mathematical methods as described in this thesis, we may even be able to compute score-optimal and biologically better solutions for these problems.

6.2 Concluding remarks

This thesis deals with detecting and quantifying structural similarities of proteins. We tackle this task by devising comprehensive mathematical models and developing exact

Chapter 6. Conclusion

algorithms for pairwise protein structure alignment. The difference of our approach compared to previous work is our ambition to express the corresponding problems in a sound mathematical way and solve them mathematically rigorously. Capturing the complex biological relations between protein structures in mathematical terms is difficult. Nonetheless, we believe that accurate modeling and subsequent quantification is essential for successively improving scoring schemes and algorithms for structure comparison and therefore for gradually deepening our understanding of the underlying biological systems.

Summary

Proteins are the molecular machines of the cell, carrying out many diverse functions. A protein's particular function usually follows from its 3-dimensional structure, a relationship known as the structure-function paradigm.

A protein is a folded chain of amino acid residues connected by peptide bonds. Its structure has a regular backbone and varying amino acid side chains. There are four levels of protein structure: primary, secondary, tertiary and quaternary structure. Primary structure is the sequence of amino acid residues. Secondary structure comprises regular elements such as α -helices and β -sheets. Tertiary structure is determined by the precise location of every atom in the structure. Finally, quaternary structure is an assembly of several amino acid chains.

Structural similarities between proteins are extremely common. This is for various reasons. First, there is only a limited number of overall structural arrangements, or so-called protein folds. Then, parts of proteins which constitute evolutionary entities, so-called domains, re-appear in many different proteins, because they are deleted, inserted, swapped, mutated, *etc.* Further, there is a high evolutionary pressure on conserving protein function and hence on conserving protein structure during evolution. Proteins which share a common ancestor, *i.e.*, homologs, thus often have a similar structure. Finally, convergent evolution may lead to similar structures for similar functions.

Because of this structure-function relationship, protein similarities help us to learn about the evolution and biological tasks of proteins. Therefore we would like to reliably detect such structural similarities. For this, we resort to experimental data which provides the 3D location of every atom in the protein structure. Two proteins are considered structurally similar if they have a similar protein backbone conformation. We thus select only one representative atom of every amino acid residue. The resulting two chains of representative atoms are then compared. From such a comparison we obtain a sequential one-to-one mapping between structurally equivalent residues in the two proteins. This mapping is called a structure alignment. It can be used to quantify the similarity of two proteins by assigning a corresponding similarity score to it. Given an alignment we can superpose two structures in 3-dimensional space in order to obtain a visual impression of their similarity.

Using optimization, we aim to detect the best structure alignment. Two steps are important. First, we need to define a scoring scheme according to which biologically correct alignments are top-scoring. Second, we use an algorithm that finds the score-optimal alignment. For reasonable scoring schemes, finding the optimal struc-

Summary

ture alignment is difficult; it is often assumed to be an *NP*-hard problem. As a result, almost all of the many existing structure alignment algorithms are heuristics. Even worse, each heuristic uses its own scoring scheme that it optimizes. Currently, there is no consensus which algorithm or scoring scheme is best.

Our contribution to the structure alignment problem is two-fold. First, we cast the problem in mathematical models and design exact algorithms to solve it. To this end we formulate general integer linear programs for inter-residue distance matrix-based structure alignment. These models cast many existing structure alignment approaches into a common framework. Finally, we solve these models to optimality by designing exact algorithms. An inter-residue distance matrix alignment is a sequential assignment of a subset of distance matrix rows and columns from one matrix to a subset of distance matrix rows and columns from the other matrix. This assignment should maximize the overall score for paired inter-residue distances. An exact algorithm for this problem will either return a mapping of maximum score or, if not found within time limit, bounds on this maximum score. We apply techniques from combinatorial optimization for our exact algorithms: integer linear programming, Lagrangian relaxation, branch-and-bound and branch-and-cut.

Our second contribution is the application of our algorithms to problems that can only be tackled by the use of exact algorithms. For example, we compute provably better alignments, obtain alignment quality guarantees and accurately quantify protein similarities. Further, we evaluate heuristic algorithms and rigorously compare scoring schemes for protein structure alignment. Finally, we provide all these services to structural biologists via a web server.

In summary, the work described in this thesis entitled *Exact algorithms for pairwise protein structure alignment* contributes towards improving algorithms and scoring schemes for protein structure alignment.

Samenvatting

Eiwitten zijn de moleculaire machines van de cel en hebben hier vele verschillende functies. De specifieke functie van een eiwit komt voort uit zijn driedimensionale structuur; dit verband staat bekend als het structuur-functie paradigma.

Een eiwit is een gevouwen keten van aminozuurresiduen die verbonden zijn door peptidbindingen. De eiwitstructuur heeft een reguliere hoofdketen en verschillende aminozuurzijgroepen. De structuur van een eiwit wordt beschreven op vier niveaus: primair, secundair, tertiair en quaternair. De primaire structuur beschrijft de volgorde van de aminozuurresiduen. De secundaire structuur omvat eiwitmotieven zoals α -helices en β -sheets. De tertiaire structuur wordt bepaald door precieze positie van atomen in de structuur. De quaternaire structuur beschrijft de compositie van meerdere aminozuurketens.

Structurele overeenkomsten tussen eiwitten komen zeer vaak voor. Dit is om verschillende redenen. Ten eerste is het aantal mogelijke structurele configuraties, ook wel eiwitvouwingen genoemd, beperkt. Vervolgens komen zogenaamde domeinen, dit zijn delen van eiwitten die evolutionaire entiteiten zijn, voor in vele verschillende eiwitten omdat ze zijn verwijderd, toegevoegd, verwisseld, gemuteerd, *etc.* Daarnaast is er een hoge evolutionaire druk op het behouden van eiwitfunctie, en dus op het behouden van eiwitstructuur. Homologe eiwitten die dus een gezamenlijke voorouder delen, hebben daarom veelal een vergelijkbare structuur. Ten slotte kan convergente evolutie leiden tot soortgelijke structuren voor soortgelijke functies.

Door het verband tussen structuur en functie kunnen structurele overeenkomsten ons helpen om meer te weten te komen over evolutionaire en biologische taken van eiwitten. Om dit te doen moeten we de structurele overeenkomsten op een betrouwbare manier kunnen detecteren. Hiervoor gebruiken we experimentele data die de driedimensionale locatie van elk atoom in de eiwitstructuur beschrijft. Twee eiwitten worden beschouwd als structureel vergelijkbaar als de configuratie van de eiwit hoofdketen soortgelijk is. Dit houdt in dat we enkel één atoom per aminozuurresidu gebruiken. De twee resulterende ketens met de representatieve atomen worden dan met elkaar vergeleken. Middels deze vergelijking verkrijgen we een één-op-één mapping tussen structureel equivalente residuen in de twee eiwitten. Deze mapping staat bekend als structure alignment, welke gebruikt kan worden om de gelijkenis van twee eiwitten door het toewijzen van een gelijkenisscore te kwantificeren. Gegeven de alignment van de eiwitten kunnen we de twee structuren superponeren in de driedimensionale ruimte om een visuele indruk van hun gelijkenis te verkrijgen.

Samenvatting

Met behulp van optimalisatietechnieken beogen wij de beste structure alignment te vinden. Hiervoor zijn twee stappen van belang. Ten eerste moeten we een scoringsschema definiëren waarmee biologisch correcte alignment maximaal scoren. Ten tweede gebruiken we een algoritme dat de alignment vindt met de optimale score. Voor biologisch relevante scoringsschema's is het vinden van de optimale structure alignment moeilijk; het is vaak vermoedelijk een *NP*-moeilijk probleem. Als gevolg hiervan zijn veruit de meeste structure alignment algoritmes heuristisch van aard. Daarbij komt dat iedere heuristiek gebruik maakt van zijn eigen scoringsschema. Er is op dit moment geen overeenstemming over welk algoritme of scoringsschema het beste is.

Onze bijdrage aan het structure alignment probleem is tweeledig. Ten eerste beschrijven wij het probleem met behulp van wiskundige modellen en ontwerpen exacte algoritmen om deze op te lossen. Hiervoor formuleren we generieke geheeltallige lineaire programma's voor structure alignment gebaseerd op inter-residu afstandsmatrices. Onze modellen plaatsen een groot aantal bestaande structure alignment methodes in een gemeenschappelijk kader. Ten slotte vinden we de optimale oplossing van deze modellen door exacte algoritmes hiervoor te ontwerpen. Een inter-residu afstandsmatrix is een sequentiële toewijzing van een deelverzameling van afstandsmatrix rijen en kolommen van een matrix naar een deelverzameling van afstandsmatrix rijen en kolommen van de andere matrix. Deze toewijzing moet de totale score voor gepaarde inter-residu afstanden maximaliseren. Een exact algoritme voor dit probleem zal een mapping van de maximum scores teruggeven, of, als deze niet wordt gevonden binnen de gestelde tijdslimiet, de grenzen van de maximum score. We gebruiken technieken uit combinatorische optimalisatie voor onze exacte algoritmes, zoals: integer linear programming, Lagrangian relaxation, branch-and-bound en branch-and-cut.

Onze tweede bijdrage is de toepassing van onze algoritmes op problemen die enkel kunnen worden opgelost met behulp van exacte algoritmes. Bijvoorbeeld berekenen wij bewijsbaar betere alignments, verkrijgen wij kwaliteitsgaranties op alignments en kwantificeren wij eiwitgelijkenissen. Verder evalueren wij heuristische algoritmen en vergelijken wij op een nauwgezette manier verschillende scoringsschemas. Ten slotte stellen wij deze diensten beschikbaar aan structuurbiologen via een webserver.

Kort samengevat draagt het werk in dit proefschrift getiteld *Exacte algoritmen voor paarsgewijze eiwit structure alignment* bij aan het verbeteren van algoritmes en scoringsschemas voor eiwit structure alignment.

Acknowledgments

The time in Amsterdam has been great!

I still remember, four years ago, sitting with Thomas in our Berliner Zimmer, which was, very rare, flooded with sunlight, summer flowers on the table, being desperately sad about the move to Amsterdam, about the upcoming changes and the end of the time in Berlin. But looking back, I can say that I did never regret taking this decision. So many new experiences, a PhD subject, working place and colleagues that could not have been better and new people, becoming part of my life. All these people, in many different ways, contributed to the existence of this thesis. In the following I would like to acknowledge them.

I start with my supervisor Gunnar Klau. He made my PhD time possible, by offering me the great opportunity to start working with him at the CWI in Amsterdam, in the, back then, brand-new Life Sciences group. I enjoyed working with Gunnar, from my first steps in science up to, finally, this thesis. He taught me a lot. Always with critical, helpful feedback, he was open to ideas I brought along. His bigger picture on the things on which we worked the last years made this thesis a self-contained piece of work which to create, piece by piece, I had a great time.

I further would like to acknowledge Rumen Andonov from INRIA Rennes, who also accompanied me scientifically most of the last four years. Only our collaboration with Rumen made much of the work presented in this thesis possible. I enjoyed my stays in Brittany, and the work and scientific exchange there has been inspiring. I remember, a few evenings walking home from INRIA during mild spring air, like being on a perfect vacation, except for being all excited about some new ideas that had come up during the day.

There have been many more people who contributed directly to the work presented in this thesis and without them many things would not have been possible. It has been a pleasure to work with you! I am grateful towards Noël Malod-Dognin, Lars Petzold, Francisco Domingues, Mathilde Le Boudic-Jamin, Guillaume Chapuis, Antonio Mucherino, Maarten Dijkema and Leonardos Mageiros.

I also would like to acknowledge my office mates and colleagues, who, over the years, made the time at CWI so enjoyable, always available for a coffee break or a walk to the Nikhef candy machine. Many also helped out scientifically, listened to problems and gave advice: Stefan Canzar, Mohammed El-Kebir, Christine Staiger, Nora Toussaint, Sandro Andreotti and Frederik van Parijs. And of course, not to forget, the other members of the Life Sciences group, which grew over the years so large that I can not

Acknowledgments

name everybody, although quite a few of you kept me company for a few years and will be remembered warmly.

Moreover, many thanks go to Mohammed El-Kebir and Fernando de Oliveira Filho for proof-reading parts of this thesis and to Margriet Palm for a great translation of the summary into Dutch.

I further would like to thank the people from the RSG Netherlands board. It has been a lot of fun organizing events with you! These are Miranda Stobbe, Hanka Venselaar, Jeroen de Ridder, Jayne Hehir-Kwa, Jurgen Nijkamp and Umesh Nandal.

I also want to thank my friends and flat mates for all the memorable experiences here in Amsterdam and elsewhere. For nights of discussions on the patio at the gracht, of excessive going out and of not leaving the couch as well as for days of traveling and discovering: Fernando de Oliveira Filho, David Borrás, Sumanta Bhattacharya, Quynh-Anh Le Nguyen, Duong Anh Hoang, Ton Hellings, Anja Teehankee, Samira Jaeger, Anna Druschke, Freya Greiner, Paul Mattern and Kathrin Kruse.

Many thanks also go to my family, my parents Inge and Rolf, my brother Ulf and sister Anne. Thank you for your love and support!

Finally I want to thank my boyfriend Thomas Metzler for all the great years and for being the constant during all changes. You are the one with whom life becomes so enjoyable! I am looking forward to the new adventures awaiting us. ☺

Curriculum vitae

Inken Wohlers was born on December 4th, 1982 in Itzehoe, Germany. She grew up close-by and, after a high school exchange year in the US, finished secondary school in 2002. Subsequently, she started her Bachelor's studies in Computational Life Science at the University Lübeck. After she obtained her Bachelor's degree in 2005, she moved to the Berlin region. There she worked for three years as a scientific assistant at the Charité university hospital Berlin where she developed scientific, web-based applications for management of biological data. Simultaneously, from 2006 to 2008, she did her Master studies in Bioinformatics at the Freie Universität Berlin. After obtaining her Master's degree in 2008, she started her PhD research at the Centrum Wiskunde & Informatica in Amsterdam, the Netherlands. The topic was protein structure comparison and a continuation of the work of her Master's thesis, both supervised by Prof. Gunnar W. Klau. During her time as PhD student, a close collaboration with Prof. Rumen Andonov at INRIA Rennes was initiated. Strengthening this collaboration, Inken completed two scientific stays at INRIA Rennes, two and six months each. She finished her thesis in 2012 and will defend it at the Vrije Universiteit Amsterdam on December 11, 2012. The results obtained during the four years of PhD studies are presented in this thesis.

Bibliography

- [13] O. Camoglu, T. Can, and A. K. Singh. Integrating multi-attribute similarity networks for robust representation of the protein space. *Bioinformatics*, 22(13):1585–1592, 2006.
- [14] A. Caprara, R. Carr, S. Istrail, G. Lancia, and B. Walenz. 1001 optimal PDB structure alignments: integer programming methods for finding the maximum contact map overlap. *J Comput Biol*, 11(1):27–52, 2004.
- [15] O. Carugo and S. Pongor. A normalized root-mean-square distance for comparing protein three-dimensional structures. *Protein Sci*, 10(7):1470–1473, 2001.
- [16] S. Climer and W. Zhang. Rearrangement clustering: Pitfalls, remedies, and applications. *Journal of Machine Learning Research*, 7, 2006.
- [17] G. Collet, R. Andonov, N. Yanev, and J.-F. Gibrat. Local protein threading by mixed integer programming. *Discrete Applied Mathematics*, 159(16):1707–1716, 2011.
- [18] G. Csaba, F. Birzele, and R. Zimmer. Protein structure alignment considering phenotypic plasticity. *Bioinformatics*, 24(16):98–104, 2008.
- [19] G. Csaba, F. Birzele, and R. Zimmer. Systematic comparison of SCOP and CATH: a new gold standard for protein structure analysis. *BMC Struct Biol*, 9:23–23, 2009.
- [20] M. E. Csete and J. C. Doyle. Reverse engineering of biological complexity. *Science*, 295(5560):1664–1669, 2002.
- [21] P. Di Lena, P. Fariselli, L. Margara, M. Vassura, and R. Casadio. Fast overlapping of protein contact maps by alignment of eigenvectors. *Bioinformatics*, 26(18):2250–2258, 2010.
- [22] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, third edition, 2005.
- [23] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [24] Z. Dosztányi and A. E. Torda. Amino acid similarity matrices based on force fields. *Bioinformatics*, 17(8):686–699, 2001.
- [25] I. Eidhammer, I. Jonassen, and W. R. Taylor. Structure comparison and structure patterns. *J Comput Biol*, 7(5):685–716, 2000.
- [26] U. Emekli, D. Schneidman-Duhovny, H. J. Wolfson, R. Nussinov, and T. Haliloglu. HingeProt: automated prediction of hinges in protein structures. *Proteins*, 70(4):1219–1227, 2008.
- [27] A. Godzik. The structural alignment between two proteins: is there a unique answer? *Protein Sci*, 5(7):1325–1338, 1996.
- [28] A. Godzik, J. Skolnick, and A. Kolinski. Regularities in interaction patterns of globular proteins. *Protein Eng*, 6(8):801–810, 1993.
- [29] D. Goldman, C. H. Papadimitriou, and S. Istrail. Algorithmic aspects of protein structure similarity. *IEEE Annual Symposium on Foundations of Computer Science*, 0:512–521, 1999.

- [30] O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol*, 162(3):705–708, 1982.
- [31] L. H. Greene, T. E. Lewis, S. Addou, A. Cuff, T. Dallman, M. Dibley, O. Redfern, F. Pearl, R. Nambudiry, A. Reid, I. Sillitoe, C. Yeats, J. M. Thornton, and C. A. Orengo. The CATH domain structure database: new protocols and classification levels give a more comprehensive resource for exploring evolution. *Nucleic Acids Res*, 35(Database issue):291–297, 2007.
- [32] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, second corrected edition, 1993.
- [33] M. Guignard and S. Kim. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Math. Program.*, 39(2):215–228, 1987.
- [34] T. Hamelryck and B. Manderick. PDB file parser and structure class implemented in Python. *Bioinformatics*, 19(17):2308–2310, 2003.
- [35] T. Hara, H. Kato, Y. Katsube, and J. Oda. A pseudo-michaelis quaternary complex in the reverse reaction of a ligase: structure of *Escherichia coli* B glutathione synthetase complexed with ADP, glutathione, and sulfate at 2.0 Å resolution. *Biochemistry*, 35(37):11967–11974, 1996.
- [36] H. Hasegawa and L. Holm. Advances and pitfalls of protein structural alignment. *Curr Opin Struct Biol*, 19(3):341–348, 2009.
- [37] M. Held, P. Wolfe, and H. P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.
- [38] R. Hoffmann and A. Valencia. Implementing the iHOP concept for navigation of biomedical literature. *Bioinformatics*, 21 Suppl 2:252–258, 2005.
- [39] L. Holm, S. Kääriäinen, C. Wilton, and D. Plewczynski. Using Dali for structural comparison of proteins. *Curr Protoc Bioinformatics*, Chapter 5, 2006.
- [40] L. Holm and P. Rosenström. Dali server: conservation mapping in 3D. *Nucleic Acids Res*, 38(Web Server issue):545–549, 2010.
- [41] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *J Mol Biol*, 233(1):123–138, 1993.
- [42] L. Holm and C. Sander. Parser for protein folding units. *Proteins*, 19(3):256–268, 1994.
- [43] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273(5275):595–603, 1996.
- [44] L. Holm and C. Sander. Dictionary of recurrent domains in protein structures. *Proteins*, 33(1):88–96, 1998.
- [45] J. Hou, S. R. Jun, C. Zhang, and S. H. Kim. Global mapping of the protein structure space and application in structure-based inference of protein function. *Proc Natl Acad Sci U S A*, 102(10):3651–3656, 2005.
- [46] L. Huang, E. Kinnucan, G. Wang, S. Beaudenon, P. M. Howley, J. M. Huibregtse, and N. P. Pavletich. Structure of an E6AP-UbcH7 complex: insights into ubiquitination by the E2-E3 enzyme cascade. *Science*, 286(5443):1321–1326, 1999.

Bibliography

- [47] M. Ikura, G. M. Clore, A. M. Gronenborn, G. Zhu, C. B. Klee, and A. Bax. Solution structure of a calmodulin-target peptide complex by multidimensional NMR. *Science*, 256(5057):632–638, 1992.
- [48] S. A. Islam, J. Luo, and M. J. Sternberg. Identification and analysis of domains in proteins. *Protein Eng*, 8(6):513–525, 1995.
- [49] F. Jacob. Evolution and tinkering. *Science*, 196(4295):1161–1166, 1977.
- [50] G. Jacobson and K.-P. Vo. Heaviest increasing/common subsequence problems. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 644 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 1992.
- [51] B. J. Jain and M. Lappe. Joining softassign and dynamic programming for the contact map overlap problem. In *International Conference on Bioinformatics Research and Development (BIRD)*, volume 4414 of *Lecture Notes in Computer Science*, pages 410–423. Springer, 2007.
- [52] B. J. Jain and K. Obermayer. Bimal: Bipartite matching alignment for the contact map overlap problem. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1394–1400. IEEE Computer Society, 2009.
- [53] J. Jung and B. Lee. Protein structure alignment using environmental profiles. *Protein Eng*, 13(8):535–543, 2000.
- [54] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5):922–923, 1976.
- [55] W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.
- [56] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.
- [57] T. Kawabata. MATRAS: a program for protein 3D structure comparison. *Nucleic Acids Res*, 31(13):3367–3369, 2003.
- [58] T. Kawabata and K. Nishikawa. Protein structure comparison using the markov transition model of evolution. *Proteins*, 41(1):108–122, 2000.
- [59] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akad. Nauk SSSR*, 244:1093–1096, 1979.
- [60] G. J. Kleywegt and T. A. Jones. A super position. *Joint CCP4 and ESF-EACBM Newsletter on Protein Crystallography*, 1994.
- [61] R. Kolodny, P. Koehl, and M. Levitt. Comprehensive evaluation of protein structure alignment methods: scoring by geometric measures. *J Mol Biol*, 346(4):1173–1188, 2005.
- [62] R. Kolodny and N. Linial. Approximate protein structural alignment in polynomial time. *Proc Natl Acad Sci U S A*, 101(33):12201–12206, 2004.
- [63] 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, 2004.

- [64] R. H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Eng*, 7(9):1059–1068, 1994.
- [65] H. P. Lenhof, B. Morgenstern, and K. Reinert. An exact solution for the segment-to-segment multiple sequence alignment problem. *Bioinformatics*, 15(3):203–210, 1999.
- [66] H. P. Lenhof, K. Reinert, and M. Vingron. A polyhedral approach to RNA sequence structure alignment. *J Comput Biol*, 5(3):517–530, 1998.
- [67] A. M. Lesk. Systematic representation of protein folding patterns. *J Mol Graph*, 13(3):159–164, 1995.
- [68] S. C. Li and Y. K. Ng. On protein structure alignment under distance constraint. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 65–76, Berlin, Heidelberg, 2009. Springer.
- [69] N. Malod-Dognin, R. Andonov, and N. Yanev. Maximum cliques in protein structure comparison. In *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 106–117. Springer, Berlin, Heidelberg, 2010.
- [70] N. Malod-Dognin, M. Le Boudic-Jamin, P. Kamath, and R. Andonov. Using dominances for solving the protein family identification problem. In *Workshop on Algorithms in Bioinformatics (WABI)*, volume 6833 of *Lecture Notes in Computer Science*, pages 201–212, Berlin, Heidelberg, 2011. Springer.
- [71] T. Margraf, G. Schenk, and A. E. Torda. The SALAMI protein structure search server. *Nucleic Acids Res*, 37(Web Server issue):480–484, 2009.
- [72] A. Marin, J. Pothier, K. Zimmermann, and J. F. Gibrat. FROST: a filter-based fold recognition method. *Proteins*, 49(4):493–509, 2002.
- [73] L. Mavridis, V. Venkatraman, D. W. Ritchie, N. Morikawa, R. Andonov, A. Cornu, N. Malod-Dognin, J. Nicolas, M. Temerinac-Ott, M. Reiser, H. Burkhardt, A. Axenopoulos, and P. Daras. SHREC’10 Track: Protein model classification. In *Eurographics Workshop on 3D Object Retrieval (3DOR)*, pages 117–124. Eurographics Association, 2010.
- [74] G. Mayr, F. S. Domingues, and P. Lackner. Comparative analysis of protein structure alignments. *BMC Struct Biol*, 7:50–50, 2007.
- [75] M. Menke, B. Berger, and L. Cowen. Matt: local flexibility aids protein multiple structure alignment. *PLoS Comput Biol*, 4(1), 2008.
- [76] K. Mizuguchi, C. M. Deane, T. L. Blundell, and J. P. Overington. HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Sci*, 7(11):2469–2471, 1998.
- [77] R. Mosca and T. R. Schneider. RAPIDO: a web server for the alignment of protein structures in the presence of conformational changes. *Nucleic Acids Res*, 36(Web Server issue):42–46, 2008.
- [78] J. Moult, K. Fidelis, A. Kryshtafovych, and A. Tramontano. Critical assessment of methods of protein structure prediction (CASP)–round IX. *Proteins*, 79 Suppl 10:1–5, 2011.

Bibliography

- [79] A. Mucherino, I. Wohlers, G. W. Klau, and R. Andonov. Sparsifying distance matrices for protein-protein structure alignments. In *Cologne-Twente Workshop on graphs and combinatorial optimization (CTW)*, pages 211–214, 2011.
- [80] 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(4):536–540, 1995.
- [81] 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, 1970.
- [82] C. Notredame, D. G. Higgins, and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1):205–217, 2000.
- [83] C. A. Orengo, A. D. Michie, S. Jones, D. T. Jones, M. B. Swindells, and J. M. Thornton. CATH—a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108, 1997.
- [84] C. A. Orengo and J. M. Thornton. Protein families and their evolution—a structural perspective. *Annu Rev Biochem*, 74:867–900, 2005.
- [85] D. A. Pelta, J. R. González, and M. Moreno Vega. A simple and fast heuristic for protein structure comparison. *BMC Bioinformatics*, 9:161–161, 2008.
- [86] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin. UCSF chimera—a visualization system for exploratory research and analysis. *J Comput Chem*, 25(13):1605–1612, 2004.
- [87] P. A. Pevzner and M. S. Waterman. Generalized sequence alignment and duality. *Adv. Appl. Math.*, 14(2):139–171, 1993.
- [88] 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, 1999.
- [89] A. Poleksic. Algorithms for optimal protein structure alignment. *Bioinformatics*, 25(21):2751–2756, 2009.
- [90] A. Poleksic. On complexity of protein structure alignment problem under distance constraint. *IEEE/ACM Trans Comput Biol Bioinform*, 2011.
- [91] A. Poleksic. Optimizing a widely used protein structure alignment measure in expected polynomial time. *IEEE/ACM Trans Comput Biol Bioinform*, 8(6):1716–1720, 2011.
- [92] O. C. Redfern, A. Harrison, T. Dallman, F. M. Pearl, and C. A. Orengo. CATHE-DRAL: a fast and effective algorithm to predict folds and domain boundaries from multidomain protein structures. *PLoS Comput Biol*, 3(11), 2007.
- [93] J. Rocha, J. Segura, R. C. Wilson, and S. Dasgupta. Flexible structural protein alignment by a sequence of local transformations. *Bioinformatics*, 25(13):1625–1631, 2009.
- [94] B. Rost. Twilight zone of protein sequence alignments. *Protein Eng*, 12(2):85–94, 1999.

- [95] D. Sankoff and J. B. Kruskal. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*. Addison-Wesley, Reading, Massachusetts, 1983.
- [96] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.
- [97] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency, Volume B*. Algorithms and Combinatorics. Springer, Berlin, 2003.
- [98] S. C. Shih, G. Prag, S. A. Francis, M. A. Sutanto, J. H. Hurley, and L. Hicke. A ubiquitin-binding motif required for intramolecular monoubiquitylation, the CUE domain. *EMBO J*, 22(6):1273–1281, 2003.
- [99] I. N. Shindyalov and P. E. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng*, 11(9):739–747, 1998.
- [100] M. L. Sierk and G. J. Kleywegt. Déjà vu all over again: finding and analyzing protein structure similarities. *Structure*, 12(12):2103–2111, 2004.
- [101] M. L. Sierk and W. R. Pearson. Sensitivity and selectivity in protein structure comparison. *Protein Sci*, 13(3):773–785, 2004.
- [102] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–197, 1981.
- [103] D. M. Standley, H. Toh, and H. Nakamura. ASH structure alignment package: sensitivity and selectivity in domain classification. *BMC Bioinformatics*, 8:116, 2007.
- [104] D. M. Strickland, E. Barnes, and J. S. Sokol. Optimal protein structure alignment using maximum cliques. *Oper. Res.*, 53(3):389–402, May 2005.
- [105] 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.
- [106] D. A. Taylor, J. S. Sack, J. F. Maune, K. Beckingham, and F. A. Quiocho. Structure of a recombinant calmodulin from *Drosophila melanogaster* refined at 2.2-Å resolution. *J Biol Chem*, 266(32):21375–21380, 1991.
- [107] W. R. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Sci*, 8(3):654–665, 1999.
- [108] W. R. Taylor and C. A. Orengo. Protein structure alignment. *J Mol Biol*, 208(1):1–22, 1989.
- [109] A. Torres, A. Cabada, and J. J. Nieto. An exact formula for the number of alignments between two DNA sequences. *DNA Seq*, 14(6):427–430, 2003.
- [110] M. A. Verdecia, C. A. Joazeiro, N. J. Wells, J. L. Ferrer, M. E. Bowman, T. Hunter, and J. P. Noel. Conformational flexibility underlies ubiquitin ligation mediated by the WWP1 HECT domain E3 ligase. *Mol Cell*, 11(1):249–259, 2003.
- [111] A. Wallqvist, Y. Fukunishi, L. R. Murphy, A. Fadel, and R. M. Levy. Iterative sequence/secondary structure search for protein homologs: comparison with

Bibliography

- amino acid sequence alignments and application to fold recognition in genome databases. *Bioinformatics*, 16(11):988–1002, 2000.
- [112] S. J. Wheelan, A. Marchler-Bauer, and S. H. Bryant. Domain size distributions can predict domain boundaries. *Bioinformatics*, 16(7):613–618, 2000.
- [113] I. Wohlers, R. Andonov, and G. W. Klau. Algorithm engineering for optimal alignment of protein structure distance matrices. *Optimization Letters*, 5(3):421–433, 2011.
- [114] I. Wohlers, R. Andonov, and G. W. Klau. Optimal DALI protein structure alignment. Rapport de recherche RR-7915, INRIA, 2012.
- [115] I. Wohlers, R. Andonov, and G. W. Klau. Optimal DALI protein structure alignment. Submitted, 2012.
- [116] I. Wohlers, F. S. Domingues, and G. W. Klau. Towards optimal alignment of protein structure distance matrices. *Bioinformatics*, 26(18):2273–2280, 2010.
- [117] I. Wohlers, N. Malod-Dognin, R. Andonov, and G. W. Klau. CSA: comprehensive comparison of pairwise protein structure alignments. *Nucleic Acids Res*, 40(Web Server issue):303–309, 2012.
- [118] I. Wohlers, L. Petzold, F. S. Domingues, and G. W. Klau. Aligning protein structures using distance matrices and combinatorial optimization. In *German Conference on Bioinformatics (GCB)*, volume 157 of *LNI*, pages 33–43. GI, 2009.
- [119] I. Wohlers, L. Petzold, F. S. Domingues, and Klau G. W. PAUL: protein structural alignment using integer linear programming and Lagrangian relaxation. *BMC Bioinformatics*, 10(S-13):0, 2009.
- [120] I. Wohlers, H. Stachelscheid, J. Borstlap, K. Zeilinger, and J. C. Gerlach. The Characterization Tool: A knowledge-based stem cell, differentiated cell, and tissue database with a web-based analysis front-end. *Stem Cell Res*, 3(2-3):88–95, 2009.
- [121] D. Wu, Z. Wu, and Y. Yuan. Rigid versus unique determination of protein structures with geometric buildup. *Optimization Letters*, 2(3):319–331, 2008.
- [122] W. Xie and N. V. Sahinidis. A reduction-based exact algorithm for the contact map overlap problem. *J Comput Biol*, 14(5):637–654, 2007.
- [123] J. Xu, F. Jiao, and B. Berger. A parameterized algorithm for protein structure alignment. *J Comput Biol*, 14(5):564–577, 2007.
- [124] Y. Ye and A. Godzik. Flexible structure alignment by chaining aligned fragment pairs allowing twists. *Bioinformatics*, 19 Suppl 2:246–255, 2003.
- [125] A. Zemla. LGA: A method for finding 3D similarities in protein structures. *Nucleic Acids Res*, 31(13):3370–3374, 2003.
- [126] A. Zemla, C. Venclovas, J. Moulton, and K. Fidelis. Processing and evaluation of predictions in CASP4. *Proteins*, Suppl 5:13–21, 2001.
- [127] J. Zhang. Protein-length distributions for the three domains of life. *Trends Genet*, 16(3):107–109, 2000.

- [128] Y. Zhang and J. Skolnick. Scoring function for automated assessment of protein structure template quality. *Proteins*, 57(4):702–710, 2004.
- [129] Y. Zhang and J. Skolnick. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Res*, 33(7):2302–2309, 2005.

