Computational methods for RNA 3D structure prediction and comparison

Dissertation

zur Erlangung des akademischen Grades Dr. rer. nat.

an der Fakultät für Mathematik, Informatik und Naturwissenschaften

im Fachbereich Chemie der Universität Hamburg

von Nils Peter Petersen

April 2020

Erstgutachter: Prof. Dr. Andrew E. Torda
Zweitgutachter: Prof. Dr. Daniel N. Wilson

Tag der Disputation: 26.6.2020
Tag der Druckfreigabe: 27.7.2020

Zusammenfassung

Diese Arbeit beschreibt neue Methoden und Programme für automatische Vorhersagen und Vergleiche dreidimensionaler (3D) RNA-Strukturen.

Mit Hilfe zuverlässiger Vorhersagen makromolekularer 3D-Strukturen ließe sich viel Arbeit für aufwendige Experimente sparen. Das etablierte Programm NAST erzeugt RNA-Strukturmodelle mit niedriger Auflösung mittels Molekulardynamik-Simulation. In dieser Arbeit wird ein grundlegender und schwerwiegender Fehler im Aufbau des verwendeten Kraftfeldes beschrieben. Die Kombination von Termen für Bindungs- und Torsionswinkel führt zu extremen Steigungen in der Energiefunktion und dadurch zu numerisch instabilen Simulationen. Um dieses Problem zu beheben, wurde NASTI (NAST Improved) entwickelt. Form und Parameterisierung der Energieterme wurden durch unterschiedliche Veränderungen verbessert. Die wichtigste Erneuerung ist ein Energieterm für Torsionswinkel, der durch Hinzunahme der benachbarten Bindungswinkel die Simulationen stabilisiert. Während NASTI ohne numerische Probleme über Millionen Zeitschritte läuft, enden NAST-Simulationen regelmäßig in numerischen Katastrophen. Vergleicht man die vorhergesagten Modelle mit bekannten 3D-Strukturen, zeigt sich außerdem, dass NASTI gegenüber NAST gleichwertige oder bessere Ergebnisse erzielt.

3D-Strukturvergleiche sind zentrale Werkzeuge in Studien zur Evolution und Funktion von Makromolekülen. Im Rahmen der vorliegenden Arbeit wurden deshalb Methoden zum schnellen Vergleich von RNA-Strukturen entwickelt. Zunächst wird eine neue Methode, das RNA-Rückgrat durch ein Strukturalphabet zu beschreiben, vorgestellt. Strukturalphabete reduzieren 3D Molekülstrukturen auf eindimensionale Buchstabenfolgen. Die hier vorgestellte Methode ist speziell für den RNA-Strukturvergleich optimiert. Die Abbildung von Strukturfragmenten auf Alphabetsbuchtstaben wurde durch einen Suchalgorithmus mit dem Ziel, ähnliche Bereiche zwischen verschiedenen Strukturen zu finden, optimiert. Anschließend werden zwei neue Programme zur Berechnung von RNA-Strukturalignments vorgestellt. ALFONS verwendet das neue Strukturalphabet und CVRRY beruht auf der Verwendung geometrischer Deskriptoren für Rückgratfragmente. Die beiden Programme sind im Vergleich zu drei etablierten Methoden deutlich schneller und produzieren ähnlich gute Alignments. Für Anwendungen, die noch schnellere Strukturvergleiche erfordern, werden die Programme URSULA und FREEDOLIN vorgestellt. Indexstrukturen auf Sequenzen des neuen Strukturalphabets

ermöglichen die Suche nach ähnlichen Strukturen in großen Datenbanken. Ein Signifikanzmaß für einander gleichende Teilsequenzen erhöht die Genauigkeit der Suchen. Im Vergleich zu einem bekannten Programm produzieren FREEDOLIN und URSULA vergleichbare bis bessere Ergebnisse.

Abstract

This thesis presents several new methods and tools for the computational prediction and comparison of three-dimensional (3D) RNA structures.

The prediction of 3D macromolecular structures would reduce the effort invested in costly experiments. The established tool NAST uses molecular dynamics simulations and a knowledge-based force field to sample low resolution structure models of RNA molecules. In this work, a major flaw in the design of the force field is described, which causes numerical instabilities during the simulations. The way in which the respective energy terms are defined results in steep cliffs in the energy landscape for some combinations of bond and torsion angles. A new program NASTI (NAST Improved) is introduced as a numerically stable replacement for NAST. Several improvements were made to the formulation and parameterisation of the energy terms. The most important is a new smoothed energy term for the torsion angles which takes the neighboring bond angles into account removing the instabilities. Simulations were conducted to compare NASTI and NAST. NASTI runs steadily for millions of steps while NAST suffers routinely numerical catastrophes. Sampled structure models were assessed *via* comparison to crystal structures. For different test cases NASTI produces structures with equal or better quality compared to NAST.

Studies of evolution and function are often based on the comparison of large numbers of 3D structures. This requires fast methods. Several approaches to this problem were developed in this project. First, a new structural alphabet description of the RNA backbone is described. Structural alphabets are used to reduce the 3D shape of a macromolecule to a one-dimensional string of letters. The method presented here was developed specifically for RNA structure comparison. Unlike previous approaches, the procedure to find the mapping of backbone fragments to letters is formulated as an optimization task with the objective to find similar regions shared by distinct structures. The next developments presented here are two new tools for RNA 3D structure alignment. ALFONS uses the structural alphabet representation and CVRRY deploys novel geometric descriptors comprising internucleotide distances and a chirality measure. Both tools are implemented into the same framework and share algorithms for alignment and superposition. A benchmark against three state of the art programs shows that the new tools are considerably faster while producing satisfactory alignments. Finally, the programs

URSULA and FREEDOLIN were developed for applications where even faster comparisons are required. The tools use the new structural alphabet representation and enhanced suffix arrays for quick alignment-free comparison. An estimate of statistical significance of substring matches improves the accuracy. Database searches were conducted to demonstrate the ability of the two methods to find related structures. Both tools are very fast, while producing results of quality comparable to a previously published method.

Contents

Abbreviations

3D	three-dimensional
A	adenine
AUC	area under the curve
C	cytosine
DME	distance matrix error
fDME	fraction distance matrix error
G	guanine
GDT-TS	global distance test - total score
GSA	generalized suffix array
LCP	longest common prefix
LCS	longest common substring
ncRNA	non-coding RNA
PCA	Principal component analysis
PDB	protein data bank
PDB-ID	PDB-identifier
PSI	percentage of structural identity
RMSD	root mean square deviation
ROC	receiver operating characteristic
RNA	ribonucleic acid
U	uracil

Chapter 1. Introduction

Non-coding RNAs (ncRNAs) have diverse and vital functions in living cells. To fully understand life at the molecular level one thus requires knowledge and understanding of ncRNAs. The function of many ncRNAs is closely linked to their three-dimensional (3D) shape [1]. The determination and analysis of RNA 3D structures is therefore an important task in molecular biology research. Computer programs play an important role in this research field [2–4]. The work underlying this thesis was therefore dedicated to the development of structural RNA bioinformatics tools.

1.1. RNA structure

A typical RNA molecule is a single strand of nucleotides which folds onto itself. The sequence of nucleotides with bases adenine (A), cytosine (C), guanine (G) and uracil (U) determines the final structure to a large extent. The folding process is mainly driven by the formation of base pairs. Nucleotide bases that are not near each other in the sequence can hydrogen-bond with each other. There are many H-bond donors and acceptors, but the most common patterns lead to canonical base pairs (AU, GC) and wobble pairs (GU). Consecutive base pairs along the chain stack on top of each other and form an A-helix (Fig. 1.1 A). RNA structures typically consist of a set of helices with loops between them (Fig. 1.1 B-C). Tertiary connections determine the 3D arrangement. The importance of base pairs for the folding process led to a hierarchical view on RNA structure. We call the sequence of nucleotides the primary structure, the set of base pairs the secondary structure and

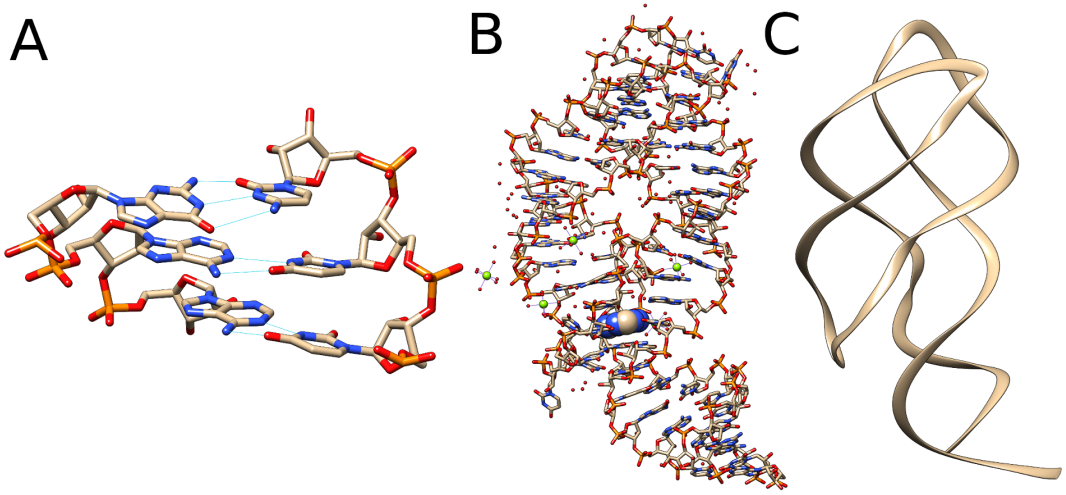

Figure 1.1.: (A) Stacked base pairs. Blue lines mark hydrogen bonds. (B) Crystallographic structure model of riboswitch-adenine complex (PDB-ID: 1y26) at atomic resolution and (C) drawn as simplified cartoon.

the overall 3D arrangement the tertiary structure [5]. On one side, this is justified by the belief that the folding process is often hierarchical, with helices folding first and further contacts being established afterwards [6]. On the other side, it is practical because knowledge about the distinct levels is gained from different experiments [7–11].

3D structure models for macromolecules such as proteins and RNAs are usually determined by nuclear magnetic resonance spectroscopy, x-ray crystallography or cryo-electron microscopy [10, 11]. Publicly available models are deposited in the protein data bank (PDB) [12]. As the main data source, this database is of major importance for research in structural biochemistry and hence also for the related bioinformatics approaches.

1.2. Scope of this thesis: RNA structural bioinformatics

Two different computational issues were addressed in this work: the prediction and the comparison of RNA 3D structures. In the following, the motivations and objectives are briefly introduced.

1.2.1. RNA structure prediction

Experiments for the determination of macromolecular 3D structures are time consuming, expensive and do not always succeed. Sequences, on the contrary, are much more easy to get [7]. Hence, a lot of time and money would be saved if one could predict structures computationally based on sequences.

A computational method for RNA 3D structure prediction is typically built with three ingredients. First, a representation for the 3D structure is required. Reduced representations are routinely used instead of full atomistic models because they are computationally cheaper [13–24]. The second ingredient is the scoring function. Two different schools of thought are established here. Physics-based energy models are based on an analytical model for physical phenomena [17, 25], whereas Knowledge-based models rely on statistics extracted from known structure models. Probability distributions for the modeled features are fitted to values found across structure models from the database. The distributions are either used without modification [21, 26, 27], or they are turned into quasi-energy functions using the Boltzmann relation [14–16, 19, 24]. Some approaches combine knowledge-based and physics-based terms [13, 15, 23]. The third ingredient is a sampling algorithm. Molecular dynamics simulation [14, 18] and Monte Carlo sampling [19–21, 24, 26–28] are used frequently. Many tools incorporate a fourth ingredient, which is the knowledge about the secondary structure and tertiary contacts [14, 18–20, 22, 26–28]. This reduces the search space drastically and thereby enables the modeling of larger structures. Contacts may originate either from experiments [8, 9] or from predictions, which can be computed on the sequence of the respective molecule alone [29, 30] or with more confidence from a multiple alignment of a set of homologous sequences [31–33].

The simulation tool NAST uses a minimalist structure representation, a knowledge based force field and contact restraints for quick structure sampling in a molecular dynamics framework [14]. Unfortunately, the simulations are numerically unstable. The cause of the instability and a solution to the problem are described in chapter 2 of this thesis. Stabilizing the force field and the implementation of some additional improvements led to the development of the tool NASTI. To proof NASTIs abilities, a comparison to NAST using five example structures is presented.

1.2.2. RNA structure comparison

Reliable comparison methods are a basic requirement for the exploration of the world of RNA molecules. Close homologs are easily found and compared with sequence alignments [34–37]. However, not all similarities are observable at the sequence level. Evolution tends to conserve structure more than sequence [38]. Similarities may still be clearly visible for the 3D structures even if the sequence identity is very low [39]. Hence, methods for structure comparison are required and may, for example, be used to answer the following questions. Are two given structures related? For a given structure, are similar molecules with known function available in a database? Which parts of two related structures are conserved? Are there conserved motifs shared by molecules with different overall 3D shape? Can we find a sensible way to group structures into different folds or classes?

Several new methods and tools for RNA structure comparison are described in chapters 3 to 5. Computational efficiency is crucial for tasks which require a large number of comparisons. All approaches presented here were designed with the goal to compare structures quickly. Chapter 3 describes a reduced string representation of the RNA backbone, which is optimized specifically for the use in structure comparison approaches. It is employed by methods described in chapter 4 and 5. Two new tools for fast pairwise RNA structure alignments are described in chapter 4 and even faster tools, which implement new alignment-free methods, are presented in chapter 5.

2 Chapter 2. A revised RNA force field

Reproduced in part with permission from [40]. © 2019 American Chemical Society.

2.1. Introduction

Coarse-grain representations can help to explore a larger variety of conformations when sampling macromolecular structures. The terms used to model interactions must be chosen carefully to be applicable in dynamics simulations. This chapter describes how an RNA force field [14] from the literature is fundamentally flawed and how it can be repaired while retaining the same low level of molecular detail.

All-atom representations may be the most intuitive models for molecules but are not always the most useful. When looking at the overall structure of a macromolecule not every single atom is of interest. Merging particles into united pseudo-atoms, on the other hand, reduces the number of energy calculations and thus helps to speed up simulations [41]. However, there is no free lunch and one has to make some compromises. Computed energies will be less accurate, it becomes more difficult to include anisotropic interactions and one loses temperature-transferability by removing degrees of freedom. The challenge is to find a formulation which preserves as much as possible of the relevant features, but there are some pitfalls. One may find a solution which works often, but occasionally fails disastrously. Such a case is described herein.

Using a lower resolution will reduce the accuracy, but also the versatility and the transferability of the model [42]. Therefore, coarse-grain models are typically designed to work for a set of relevant conditions. This could, for instance, be a restricted temperature range or specific chemical environment. One can even take this one step further and tailor models for one specific task. This is the case for the NAST force field for RNA simulations [14]. The force field is designed for a setting where secondary structure and tertiary contact information is available from an experiment [8,9] or computational prediction [29–33]. This information is used in a Newtonian dynamics simulation to restrain the coarse-grain particle coordinates in order to form the respective base pairs and helices. One should keep in mind that the simulation does not resemble any physical process. Instead, it is used as a sampling method to find plausible conformations which fit the given restraints.

NAST deploys a minimalist structure representation with one spherical particle per nucleotide, which is centered at the C3' atom of the ribose ring. This is sufficient to describe the overall shape of RNA molecules. To make up for the accuracy loss due to this drastic simplification, particles are labeled as being part of a base-paired/helical or loop region. In helical regions, angles and dihedrals are restrained to ideal A-helix geometry. The respective force constants were chosen rather arbitrarily by trial and error. Energy terms acting on particles in loop regions were parameterised based on reference values collected from ribosome structures. Symmetric distributions were fitted to the measured geometries and used to calculate force constants using the inverse-Boltzmann relation.

The application of NAST was reported in various projects. The tool was used to model tRNA [43], RNA junctions [44], trans activation response regions [45], group I introns [46], the Panicum mosaic virus [27] and parts of the tobacco mosaic virus [47]. Weinreb et al. incorporated NAST into a pipeline for the prediction of RNA 3D structures based on contacts determined by evolutionary couplings [33].

The NAST approach may be well suited for its specific task, but there is a detail which causes numerical disruptions that often end in a collapse of the simulated system. In order to be used in Newtonian dynamics simulations, energy terms need to be continuous and differentiable. There is yet another requirement. If the rate of change of energy with respect to coordinates gets too large, the linear approximation used by a first-order integrator is not valid anymore. This causes numerical errors ranging from small

irregularities to a breakdown of the whole simulated system [48]. The energy terms defined in NAST fulfill these requirements to a large extent. Unfortunately, there are conformations, visited regularly in most simulations, which result in numerical explosions. Here, an explosion is defined as a temperature change by orders of magnitudes over only a few time steps. The problem was recognized before. Reading the distributed code, one can see that the program checks the temperature in each iteration and reassigns velocities if an explosion is detected. However, it is unclear whether the cause of the explosions was known since the authors did not report any unusual behavior of the simulations.

The observed instabilities are the result of an unfortunate interplay of the dihedral angle term and a soft bond angle term. A conventional form $E_\phi = k\cos(\phi - \phi')$ is used for dihedral angles with dihedral angle ϕ, a reference value ϕ' and a force constant k. This term is problematic when soft constraints are used for bond angles which allow conformations where the angle is close to 0 or π rad. If the adjacent bond angle is exactly 0 or π rad, the dihedral is undefined. This may be unlikely to happen in practice, but conformations close to this point are also problematic. The energy landscape becomes so steep in this area that a simple integrator fails to give a sensible approximation (Fig. 2.1 A). Tiny changes in Cartesian coordinates cause large jumps in energy. This happens regularly in NAST simulations, but is hidden by the strong interference of a temperature bath and the mechanism to reset velocities. A more elegant solution is presented in this chapter. Several changes are made to the energy terms and parameters while preserving the basic features of the method.

Repairing the problem of numerical instability, as well as changes to improve the fit to known data, have led to a new program called NASTI[1]. The most important difference between NASTI and NAST is the form of the dihedral angle energy term. A new term for dihedral angles is defined, which also considers the adjacent bond angles as depicted in Figure 2.1 B. The new function inherits the periodic form from the classical definition but is, in order to avoid the energy jumps seen in NAST, extended with additional terms to flatten the curve when either of the adjacent bond angles approaches 0 or π. Using the opportunity, some additional changes are made to update the functional forms and the parameterisation of some other terms. A more sophisticated treatment of helical regions and energy terms involving particles in loops and helices is introduced. Instead of enforcing

[1] NAST Improved

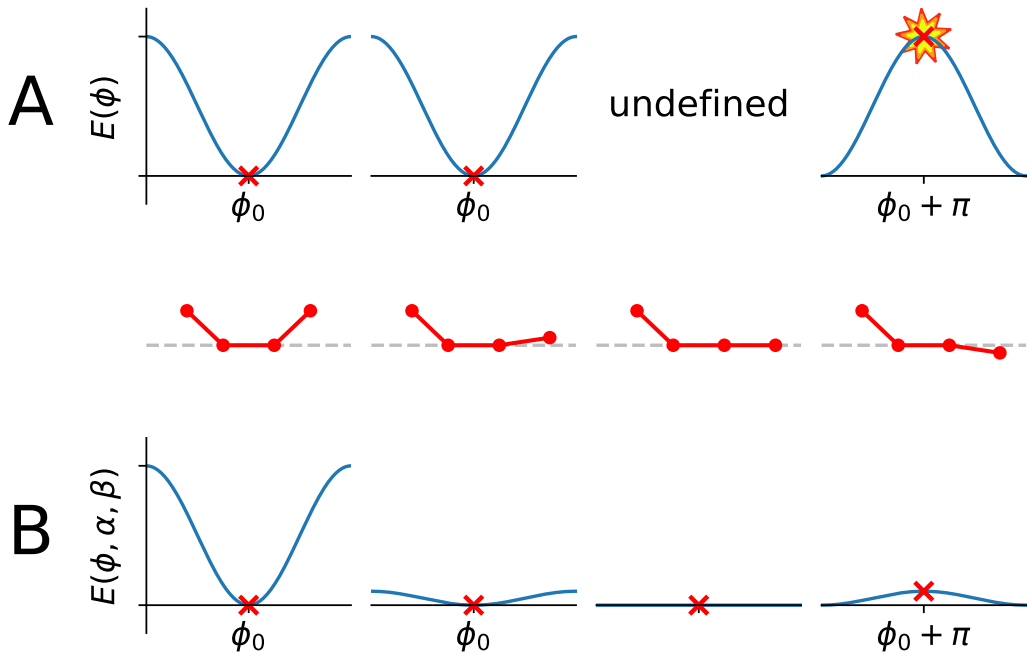

Figure 2.1.: Relative energies of stable and unstable dihedral angles in (A) NAST and (B) NASTI. The center row shows four arrangements of atoms. The top row depicts the corresponding NAST energies as a function of the dihedral angle ϕ. When passing π rad, the adjacent angle ϕ instantly changes its value by π rad. In the most extreme case the energy jumps from the minimum to the maximum of this energy term, as indicated by the star in the far right plot. The bottom row shows the same energy dependence in NASTI.
Reprinted with permission from [40]. © 2019 American Chemical Society.

ideal A-helix geometry, all energy terms, including those for helical regions, are fitted to values collected from crystallographic structures. Base pairing patterns are defined for a more specific parameterisation of energy terms acting on particles from both loop and helix regions. The harmonic energy terms for backbone angles were replaced with spline functions to account for the skewness observed on distributions taken from crystallographic structures. Structure ensembles were sampled with both NAST and NASTI for evaluation. The sampled structure models were assessed by root mean square differences (RMSD) and global distance test (GDT-TS) values [49] with respect to the crystal structures. Additionally, local geometries were computed on the sampled structures to compare the distributions to those found in ribosomal subunits. NASTI was tested with two objectives in mind. The structures sampled with NASTI should be as good as those produced

by its predecessor, and the simulations should be so close to conserving energy that they run steadily with minimal interference of the temperature bath.

2.2. Methods

2.2.1. Datasets

Three sets of RNA structures were used in the parameterisation and evaluation. For the parameterisation of energy terms, 40 structures, given in Table A.1, were taken from the list of non-redundant RNA structures by Leontis and Zirbel [50]. Only structures with a chain length of more than 150 nucleotides were used. Global structural alignments were computed for all pairs using FRIEs [51]. If the percentage of structural identity [52] within a pair was greater than 0.7, the molecule with the shorter chain was removed. Five structures given by Weinreb et al. [33] were used for evaluation simulations. Base pairs and contacts were determined using RNAView [53]. Contacts along the Watson-Crick edges of the nucleotides were used for the secondary structure input and pseudoknots smaller than three nucleotides were removed. For each structure a small list of additional tertiary contacts was used. A contact was added if it was amongst the strongest N predictions (for chain length N) by Weinreb et al. [33] and not already in the list of secondary structure restraints. NASTI was parameterised on a larger set of structures than NAST, so it might be unfair to compare simulation snapshots against the larger parameterisation set. To avoid this problem, distributions of distances, angles and torsion angles in simulations were evaluated by comparison against the three ribosomal subunits used by Jonikas et al. [14].

2.2.2. Energy function

NASTI uses the same coarse-grain representation as its predecessor with one pseudoparticle per nucleotide centered at the C3′ atom. As in NAST, each particle is labeled as to whether it is part of a base pair or a non-paired loop particle. The total energy of the system is

$$E_{total} = E_{non\_bonded} + E_{bonds} + E_{angles} + E_{dihedrals}, \tag{2.1}$$

where E_{non_bonded} is the energy due to non-bonded particles and E_{bonds}, E_{angles}, $E_{dihedrals}$ are energy terms due to bond distances, bond angles and dihedrals respectively. For E_{non_bonded} NASTI uses exactly the same functional form and parameters as NAST. Bonded energy terms stem from every backbone bond distance, angle and torsion angle. Specific helix geometry terms act on sites within base pairs. If particle i is paired with particle j, there are four additional restraints. One force restrains the distance r_{hlx} between the two particles and another one the angle θ_{hlx} affecting the triplet $(i+1, i, j)$. There are two different dihedral restraints: ϕ_{hlx} for particles $(i+1, i, j, j-1)$ and ϕ_{knight} for $(i-1, i, i+1, j-1)$. Topologically these terms have the same form as in NAST, but some of the functional forms were altered. The parameterisation is explained first, followed by the description of the functional forms.

NASTI applies different energy contributions to the particles in helix, loop and border regions. In the following, the labels 0 and 1 are used for non-paired and base-paired nucleotides respectively. The sequence of labels gives each geometry a signature. A backbone angle, for example, involves three particles and thus there are eight possible different signatures. The signature 011 would describe an angle between three particles of which the first is not paired while the other two are. Values for each combination of geometry and signature were collected from the Leontis and Zirbel set. The parameters of the corresponding energy term were then fitted to the data using the Boltzmann relation

$$E(x) = -RT \ln P(x), \tag{2.2}$$

where E is a pseudo-energy contribution, R and T are the gas constant and temperature and $P(x)$ the probability of geometry x.

Bond distances are restrained between neighboring particles along the backbone and between the two nucleotides that form a base pair. Normal distributions were fitted to the observed distributions. This directly leads to a harmonic energy term

$$E_{bond}(r_i) = k_i^{bond}(r_i - r_i')^2 \tag{2.3}$$

with distance r_i, distance optimum r_i' and force constant k_i^{bond} for bond i. The form of the energy term is thus the same as in NAST while the parameters are different. Before fitting, outlier distances were removed. Let Q_1 and Q_3 be the first and third quartiles and $\delta_Q = Q_3 - Q_1$ the interquartile range. Instances smaller than $Q_1 - 1.5\delta_Q$ or larger than $Q_3 + 1.5\delta_Q$ were removed.

The energy contribution from bond angles was given by

$$E_{angles} = E_{backbone\_angles} + E_{helix\_angles} \tag{2.4}$$

E_{helix_angles} accounts for the angles between base pairs and the backbone in helices. They were parameterised by fitting normal distributions, again yielding a harmonic potential. $E_{backbone\_angles}$ were parameterised with a more sophisticated approach using splines. First a histogram was built using bin width w following Scott [54]:

$$w = 3.49\sigma N^{-\frac{1}{3}}, \tag{2.5}$$

where σ is the standard deviation and N the number of samples. The probability density from this histogram was converted to pseudo-energies using equation 2.2. Finally, quadratic splines were fit to the energies to account for the asymmetry in the distributions. An implementation of a B-Spline fitting algorithm by Dierckx [55] provided by the SciPy library [56] was used. The energy terms in our force field implementation have the form

$$E_{backbone\_angle}(\theta_i) = \sum_{j=1}^{N_{intervals}} s_j(\theta_i) \tag{2.6}$$

with

$$s_j(\theta_i) = \begin{cases} a_{i,j}\theta^2 + b_{i,j}\theta + c_{i,j}, & \text{if } t_{i,j} \leq \theta < t_{i,j+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

where $t_{i,j}$ is the boundary of the interval j and $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$ are the interval specific coefficients for angle θ_i.

Dihedral angle distributions were fit to von-Mises distributions [57], giving a pseudo-energy term:

$$E_{periodic}(\phi_i) = k_i^{dih} \cos(\phi_i - \phi_i') \tag{2.8}$$

where k_i^{dih} is the force constant and ϕ_i' the reference dihedral for dihedral ϕ_i. To avoid the problems with sudden energy jumps, this was coupled to the angle term

$$f(\theta) = \begin{cases} 0.5\cos[\pi(\frac{\theta}{b} - 1)] + 0.5, & \text{if } \theta < b \\ 1, & \text{if } b \leq \theta \leq \pi - b \\ 0.5\cos[\pi(\frac{\theta - \pi}{b} + 1)] + 0.5, & \text{if } \theta > \pi - b \end{cases} \tag{2.9}$$

where θ is the angle and b is an arbitrarily set parameter that controls the range in which a changing angle changes the energy. The value was chosen empirically to be small while still giving numerically stable simulations. b was set to 0.1 rad for all dihedrals. Finally, 2.8 and 2.9 were combined to get

$$E_{dih}(\phi_i, \alpha_i, \beta_i) = f(\alpha_i)f(\beta_i)(E_{periodic}(\phi_i) - k_i^{dih}) + 2k_i^{dih} \tag{2.10}$$

where α_i and β_i are the neighbor angles to dihedral ϕ_i.

2.2.3. Initial random structures

Initial random structures were generated following Chen et al. [46] Starting from a random position in the sequence, particles are added successively at both ends at random coordinates but with a correct bond length and avoiding collisions. 5×10^4 steps of simulation at 300 K were used to relax the system after each particle addition.

2.2.4. Simulations and structure sampling

All simulations were done twice, once with NAST and once with NASTI. Using different initial random number seeds, 50 simulations were conducted without a thermostat (NVE) and 100 simulations with an Anderson thermostat (NVT) for each structure in the test set in both force fields. The collision frequency was set to 1 ps^{-1} for equilibration and to 1 ps^{-1} and 0.01 ps^{-1} during production in NAST and NASTI respectively. Each simulation started with a different initial random structure. After equilibration of 10^4 steps, each case was simulated for 10^6 steps with a time step of 0.005 ps and a snapshot saved every 10^3 steps. NAST simulations were run using the most recent version [58].

2.2.5. Structure quality evaluation

RMSD and GDT-TS values were computed for every snapshot with respect to the PDB-structure. The RMSD between two sets of C3' atoms measures the average error between the coordinates in angstroms. GDT-TS scores were originally popularised for protein comparisons [49] where they are the average of the fraction of C^{α} atom pairs superimposable to within 1, 2, 4 and

8 Å. This gives a convenient score between 1 (very similar structures) and 0 (very different). They were adapted for RNA using C3' backbone atoms.

2.2.6. Bootstrapping

The significance of differences in RMSD and GDT-TS scores was evaluated using bootstrap confidence intervals and permutation tests [59]. This kind of test assumes independence of measurements, but this is not the case for measurements within a trajectory. To minimize the effect of correlations, the average of all RMSD or GDT-TS scores for snapshots from the same trajectory was used. This led to 100 mean values from 100 simulations for both RMSD and GDT-TS. Working with average values from the NAST force field could be problematic since an unstable trajectory led to samples with outrageous energy values. This meant outliers had to be removed. If δ_Q for each run was the interquartile energy range from the sample distribution, then points were removed if $E > Q_3 + 1.5\delta_Q$ where Q_3 is the third quartile. In the following a sample refers to a collection of per simulation mean RMSD or GDT-TS scores.

50 000 resamples were taken with replacement from the original sample of per simulation averages to compute the 99 % bootstrap confidence interval. Each resample had the same size as the original. The average was computed for each resample yielding a sorted list of averages. The 0.5th and 99.5th percentile of that list were used as lower and upper limits of the 99 % bootstrap confidence interval.

A two sided permutation test was conducted against the null hypothesis that the means in the samples from NAST and NASTI are equal. A joint sample came from merging the outcomes of NAST and NASTI, again using per simulation averages. 50 000 permutations were sampled of this joint distribution. Each permutation was divided into two groups with the sizes of the original samples and the difference of the means of these subsamples was computed. The collection of these differences is referred to as the permutation distribution. Mean and standard deviations as parameters of a t-distribution were used to compute p-values for the observed differences.

2.2.7. Histogram distance

Histogram distances were computed following Cao and Petzold [60] to compare sampled distributions of bonds, angles and dihedrals with the respective distributions from the PDB reference data. The bin width was calculated using equation 2.5 with the statistics of the reference data.

2.2.8. Implementation

NASTI was implemented using the OpenMM library [61]. The new energy terms for dihedrals and angles were added to OpenMM as a C++ plugin. The main routines of NASTI are written in Python 2 [62] calling OpenMM functions through the provided interface.

2.2.9. Availability

NASTI is available at `https://gitlab.com/nilspetersen/nasti`.

2.3. Results

All results shown here were obtained from simulations of the five RNA chains used by Weinreb et al. [33].

2.3.1. Numerical stability

Numerical instabilities are best observed in simulations with no temperature coupling (NVE). In total, 250 NVE simulations were conducted with each tool to test for numerical stability. After a short equilibration run coupled to a temperature bath at 300 K, each system was simulated for 5 ns (10^6 steps) with no temperature coupling. Energies were sampled every 1000 steps. The most relevant result of this work is depicted in Figure 2.2. Every single simulation with NAST ends in a numerical catastrophe while NASTI simulations always remain stable with temperatures close to 300 K as shown by the inset.

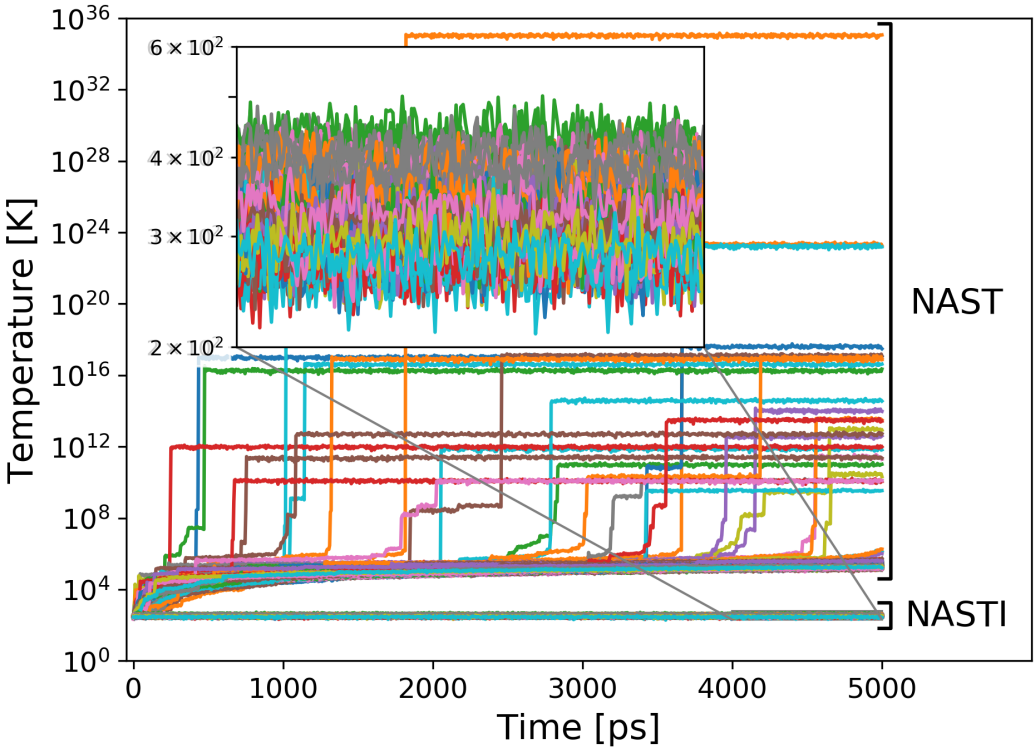

Figure 2.2.: Temperature from trajectories without coupling to a thermostat. Labels show whether NAST or NASTI was used. The inset is zoomed to show the slight difference between stable simulations. Colors are arbitrary and only serve to separate calculations. There are 250 simulations for both force fields.
Reprinted adapted with permission from [40]. © 2019 American Chemical Society.

2.3.2. Evaluation of sampled ensembles

In addition to numerical stability, NASTI should provide structure samples which are at least as good as those sampled with NAST. Simulations were carried out using temperature coupling at 300 K. 10^5 snapshots were collected for every molecule in the test set with each force field. These were assessed in two ways. Distributions of local geometries (bond lengths, angles, dihedrals) were collected and compared to reference distributions collected from crystal structures of ribosomal subunits. The overall quality of the sampled structure models was assessed by a comparison of the snapshots with the respective structure models from the PDB [12].

The results concerning local geometries are presented first. NASTI was parameterised on a larger set of ribosome structures than NAST. The evalu-

ation of the local geometries was therefore restricted to the chains of Jonikas et al. [14] in order to get a fair comparison. A distribution was assembled for each structural property corresponding to an energy term. The distributions collected from NASTI samples were closer to the reference than those from NAST in all but one of the cases (Table 2.1). Histograms for bond angles and dihedrals are depicted in Figures A.1 to A.3. The most outstanding observation was made for distributions of an angle given by the orientation of a base pair to the backbone (Fig. 2.3 A). Mean (μ) and standard deviation (σ) are 1.3 rad and 0.2 rad respectively for the samples produced with NAST. These values differ significantly from those measured for crystal structures, which are $\mu = 1.1$ rad and $\sigma = 0.1$ rad. NASTI reproduces these values more closely with $\mu = 1.1$ rad and $\sigma = 0.1$ rad. The consequence is that helices in structure models sampled with NASTI are more similar to those found in crystal structures (Fig. 2.3 B).

The force constants for dihedral terms in NASTI are smaller than those used in NAST (Table A.2). This implies weaker restraints on the torsion angles and therefore allows them to visit a wider range of conformations, which is reflected by significantly larger variances of the sampled distributions (Fig. A.2 and A.3). The differences are most evident for energy terms acting on both base-paired and non-paired particles.

To assess the overall accuracy of the structure predictions, RMSD and GDT-TS scores were computed between the sampled coordinates and the structures in the PDB. The measured scores for all five structures are depicted as box plots in Figure 2.4. NASTI performs always better than NAST with respect to the GDT-TS scores. Considering the RMSD values, NASTI achieves better results for three of five cases while the original NAST force field produced structures with lower RMSDs for the other two cases. Bootstrapping techniques were used to assess the statistical significance of these results. 99 % bootstrap confidence intervals were computed and permutation tests were conducted on the sampled means of per simulation averages. Means, confidence intervals and permutation test p-values for the RMSD and GDT-TS values of the five test cases are listed in Table 2.2. The means behave similar to the medians and quartiles shown in Figure 2.4. The highest permutation test p-value is ~ 0.5 % and the confidence intervals are not overlapping. This clearly supports the previous observations, which are thus unlikely to be due to chance.

		NAST	NASTI
r_{bb}	00	**0.02**	0.03
	01	0.03	**0.03**
	10	0.04	**0.03**
	11	0.06	**0.03**
r_{hlx}	11	0.03	**0.01**
θ_{bb}	000	0.09	**0.07**
	001	0.18	**0.05**
	010	0.17	**0.06**
	011	0.09	**0.06**
	100	0.09	**0.07**
	101	0.20	**0.05**
	110	0.11	**0.04**
	111	0.12	**0.06**
θ_{hlx}	10	0.26	**0.17**
	11	0.45	**0.07**
ϕ_{bb}	0000	0.08	**0.02**
	0001	0.08	**0.04**
	0010	0.19	**0.05**
	0011	0.15	**0.06**
	0101	0.14	**0.04**
	0110	0.16	**0.05**
	0111	0.15	**0.04**
	1000	0.07	**0.04**
	1001	0.14	**0.05**
	1011	0.14	**0.06**
	1100	0.17	**0.04**
	1101	0.17	**0.07**
	1110	0.15	**0.05**
	1111	0.11	**0.03**
ϕ_{hlx}	10	0.17	**0.03**
	11	0.07	**0.02**
ϕ_{knight}	111	0.07	**0.02**

Table 2.1.: **Histogram distances between sampled distributions and those collected from crystal structures.** The smaller value in each line is printed bold. The abbreviations *bb* and *hlx* refer to values computed on the backbone or helices, including particles on both sides of the helix.
Reprinted with permission from [40]. © 2019 American Chemical Society.

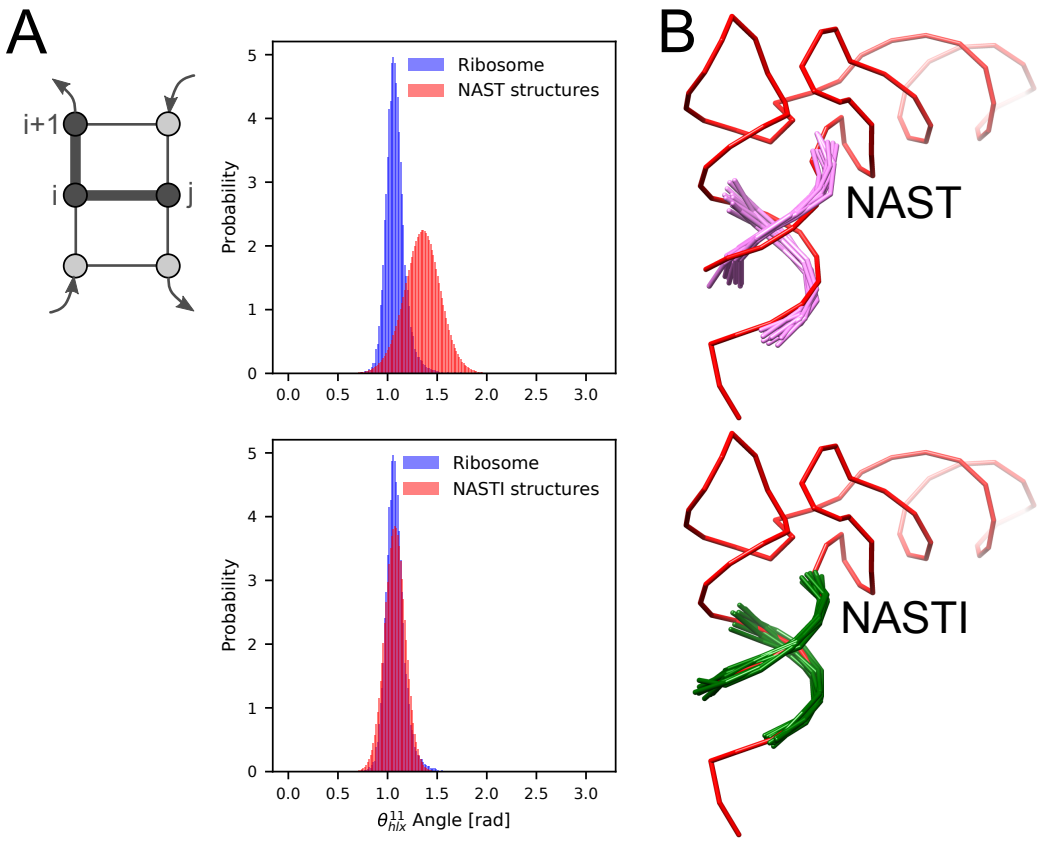

Figure 2.3.: (A) Comparison of inner helix angle (shown left) from ribosome reference structures (blue) with values from simulation snapshots (red) from NAST (top center) and NASTI (bottom center). (B) Helices from structures sampled with NAST (pink, top right) and NASTI (green, bottom right) superimposed on the target structure (red).
Reprinted with permission from [40]. © 2019 American Chemical Society.

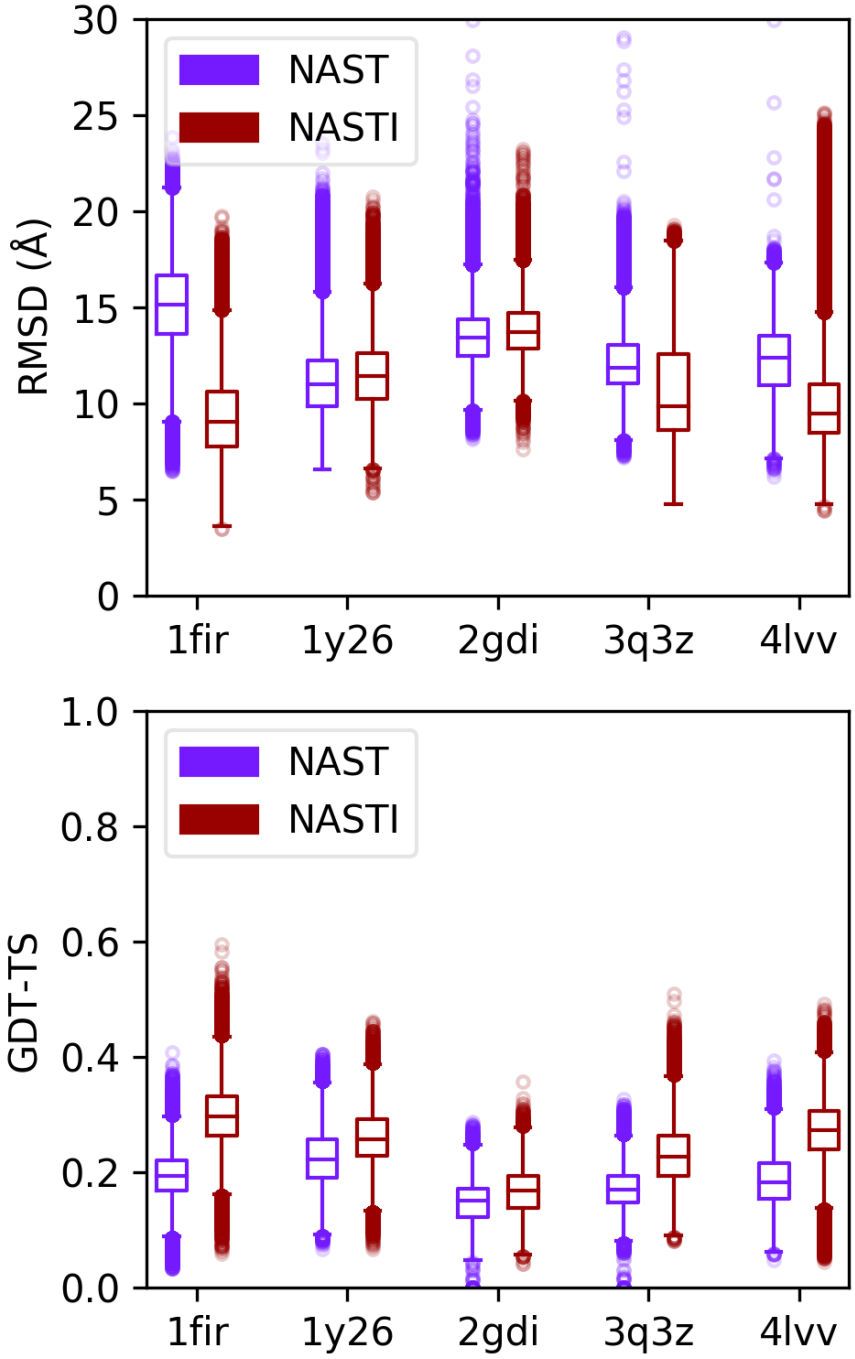

Figure 2.4.: RMSD (top) and GDT-TS (bottom) for structure ensembles sampled with NAST and NASTI. Boxes show medians, upper and lower quartiles. Whiskers stretch out to 1.5 interquartile ranges. Outliers are drawn as circles. Unstable NAST simulations occasionally led to very high RMSD values outside of the range of the plots and are not shown.
Reprinted with permission from [40]. © 2019 American Chemical Society.

RMSD Mean (CI) in angstroms

	NAST	NASTI	*P*-Value
1fir	15.1 (14.6, 15.6)	**9.4** (9.1, 9.9)	0
1y26	**11.1** (10.8, 11.4)	11.5 (11.4, 11.6)	5.1×10^{-03}
2gdi	**13.4** (13.2, 13.7)	13.9 (13.8, 14.0)	5.8×10^{-04}
3q3z	12.3 (11.9, 12.7)	**10.7** (10.1, 11.4)	8.2×10^{-07}
4lvv	12.3 (11.8, 12.7)	**10.2** (9.6, 11.0)	1.7×10^{-09}

GDT-TS Mean (CI)

	NAST	NASTI	*P*-Value
1fir	0.19 (0.18, 0.20)	**0.30** (0.29, 0.30)	0
1y26	0.22 (0.22, 0.23)	**0.26** (0.26, 0.27)	2.2×10^{-16}
2gdi	0.15 (0.14, 0.15)	**0.16** (0.16, 0.17)	3.3×10^{-10}
3q3z	0.17 (0.16, 0.18)	**0.23** (0.22, 0.24)	0
4lvv	0.19 (0.18, 0.20)	**0.27** (0.26, 0.28)	0

Table 2.2.: **Comparison of NAST and NASTI snapshots with PDB structures.** Mean RMSD and GDT-TS of simulation averages with 99 % confidence interval and permutation-test *p*-value. The means of the more correct results are printed bold.
Reprinted with permission from [40]. © 2019 American Chemical Society.

2.4. Discussion

Let us first quickly summarize the most important results. While producing similar results compared to its predecessor NAST, NASTI runs simulations seamlessly without numerical instabilities or explosions. The new force field may be as *ad hoc* as the old one, but it enables sampling in a better defined micro-canonical ensemble without strong interference of a temperature bath. This has several benefits. Checking for explosions and re-setting velocities, as implemented in NAST, is not necessary anymore. Also, analyzing the trajectories became much easier since no filtering for outliers is required. NASTI is therefore much better suited for the use in pipelines like the one proposed by Weinreb et al. [33].

The new formulation also enables the simulation of larger molecules. The number of dihedral angles is proportional to the length of the simulated chains. Hence, the probabilities of explosions also increased with the chain length when simulating using the original NAST. With numerically stable simulations and a well defined ensemble, NASTI is also much better suited to replica exchange methods than its forefather. Jonikas et al. [14] wrote that the benefit of such methods may be modest since the quality of the sampled structures is probably more limited by the low resolution of the model than by sampling efficiency. At least, it is possible now to implement a reliable sampling procedure.

A critical view on the developments and formulations presented here should be helpful for future work. Note that there is little physical basis for the terms used in NASTI. Using the Boltzmann-relation on macromolecular structures is based on several simplifications. The quantities from statistical potentials are therefore at best rough approximations of the true energies underlying the folding process [63]. Instead, one may describe the knowledge-based approach as a way to sample structure models with geometries that agree with the statistics of known 3D structures.

In defense of this approach, one could claim to have a useful scoring function for efficient conformational sampling and concede that one has no claims of statistical mechanical rigor. Especially the new dihedral terms are a pragmatic but also *ad hoc* solution. The problem originated from the adaption of terms conventionally used in atomistic force fields, where bond angles are extremely rigid. There are no problems with the classical cosine dihedral term if the bond angles do not approximate values close to 0 or π. However, the bond angles in NAST are not defined on single atoms but on sites which

are not really bonded to each other. By fitting the terms to the distributions from crystallographic structures one obtains very flexible angles. This caused the instabilities observed in NAST simulations.

Flattening the energy term in the critical area, as it was done in this work, may be one of the most direct ways to solve this problem. This is implemented using a cross term on bond angles and dihedrals. Separate terms are used to persuade the bond angles to resemble the distributions used for parameterisation. A more elegant approach would be to derive a single term from a joint distribution of bond and torsion angles. However, the solution used in this work fixes the instability problem and the structures sampled with NASTI reproduce crystallographic geometries quite well. It is therefore definitely an improvement over the formulations used in NAST.

Despite being parameterised on a larger set of structures, NASTI reproduces the geometry distributions of the original ribosomes used in the NAST parameterisation better than NAST. Although most improvements listed in Table 2.1 are quite small, some interesting observations were made. Small improvements can be observed for the distributions of bond lengths, bond angles and torsion angles both in backbone and helix geometry. There are two reasons for this. The first reason is the more sophisticated parameterisation scheme used in NASTI. NASTI has different terms for cases where all particles belong to either a loop or a helix and those where only some of the affected particles are part of a base pair. The effect is most obvious for dihedrals at helix borders. In NAST, a dihedral is simply restrained to helix geometry if two or more of the affected nucleotides are part of a base pair. This results in distributions with a very small spread for these cases. The more complicated scheme used in NASTI, on the other hand, produces distributions which are much closer to the reference values. The second reason is that backbone angles are modeled with spline functions in NASTI. These terms allow for skew and therefore reproduce the observed distributions better than the symmetric harmonic terms used in NAST.

The most salient difference between the old and the new force field was observed for an angle which describes the orientation of a base pair to the backbone. NAST sampled a distribution which deviates substantially from the reference values. The cause of this is a bad parameter choice. The corresponding line in the NAST source code assigns an ideal value of 1.5 rad. While the supplement provides the same number, an optimum at 1.76 rad is given in the main publication [14]. Both values are far from the mean observed in crystallographic structures, which is 1.08 rad.

Local geometry is maybe only partially important if one is interested in the overall shape of a molecule. RMSD and GDT-TS scores were computed with respect to crystallographic structures to get an idea about the quality of the sampled coordinates. Samples from both force fields have low GDT-TS scores. This reflects the sensitivity of the score to coordinate differences larger than 1 Å. Such a high accuracy is out of reach for force fields as simple as the ones used here. However, we can observe differences between the scores calculated for samples from NAST and those from NASTI, and the results differ for RMSD and GDT-TS. NASTI was always better with respect to GDT-TS and produced on average lower RMSDs in three of five cases. Although the bootstrapping tests suggest that the differences are not by chance, one would exaggerate by claiming that these are substantial improvements. An interesting difference between RMSD and GDT-TS can be observed. RMSD values are more affected by outliers due to flexibilities in the molecule while GDT-TS scores are more likely to reward good local geometry.

More examples favored by either NAST or NASTI are probably easily found. If a fragment contains both paired and non-paired bases, the rigid backbone in NAST simulations becomes favorable when the true shape really is helix-like. NASTI, on the other hand, is more likely to succeed for loop structures which deviate significantly from helix geometry. Furthermore, NASTI also produces helices which are more like those found in structures deposited in the PDB.

The NAST framework is simple, but this also means that there is room for extensions. One could, for instance, incorporate energy terms for coaxial stacking or specific loop motifs [64].

The work presented here may be useful beyond the specific task of RNA structure prediction. When designing a coarse-grain force field for macromolecular or similar structures, one may be tempted to use familiar energy terms and simply apply them in the new setting. This, however, is not always wise as one can see from the numerical instabilities in NAST. Critical testing is therefore required, sometimes followed by a little adjusting surgery.

2.5. Conclusion

The new program NASTI is a numerically stable alternative to NAST. RNA structures sampled with NASTI are at least as good, if not a little bit better, as those produced by NAST, while simulation runs are smooth as butter. NASTI should therefore replace NAST in practice. It is recommended for applications which require a simple, quick and easy-to-extend tool.

3 Chapter 3. Structural alphabet optimization

3.1. Introduction

A new method for designing a structural alphabet for RNA is described here. A structural alphabet is a technique to discretize the 3D shape of a macromolecules backbone. Typically, short overlapping backbone fragments are translated to a 1D sequence of letters. Structural alphabets have been extensively built and applied to protein structures. Common applications are fold recognition [65], the prediction [66] and the comparison [67–69] of structure models. The benefit of using structural alphabets for structure comparison is that it facilitates the application of methods developed for sequence comparison [67–69]. One looses information due to the reduction to one dimension and the discretization, but one gains the speed of fast algorithms.

The core of these methods is the encoding of backbone geometry to a sequence of letters. In many approaches, fragments spanning multiple residues are described with geometrical descriptors like distances [70,71] or torsion angles [68]. Based on this description the fragments are typically grouped in an unsupervised manner with algorithms such as nearest neighbor clustering [68] or hidden Markov models [70,71]. If the fragments seem to be distributed into discrete classes, then describing the structure by a series of class memberships may be a reasonable approximation. A downside of the discretization is that fragments in neighboring bins may be very similar if

they are both close to the border between the two bins. Because of their similarity it would be preferable to match them. Unfortunately, the fragments are not matched by the comparison algorithm since they were assigned to different letters. How often this happens obviously depends on both the encoding and the distribution of the fragments according to the chosen descriptors. It will happen more often the less well separated the clusters are. Another difficulty is the choice of the number and the size of the clusters. Variations of the fragment coordinates can be due to experimental accuracy, mobile regions in the molecule or mutations. Local fragments may differ but still contribute to a similar overall backbone shape. Other fragments are so different that it is very unlikely or impossible to observe them in corresponding positions of similar molecules. To counteract these problems methods to parameterise substitution matrices were adapted from protein sequence comparison [68,72]. These matrices are parameterised using previously existing multiple alignments with known 3D structures. Broadly speaking, the substitution matrix is parameterised to reward exchanges that have been seen often and penalize exchanges which have been rarely seen.

Alphabets were also built and applied in this manner for RNA structure comparison [73–78]. The RNA backbone was described by torsion [73,74] or coarse grained pseudotorsion angles [75–78]. Fragments were collected and clustered using vector quantization [73,74], affinity propagation clustering [76–78] and the Blackman window function [75,79]. The statistical method underlying the BLOSUM family of substitution matrices [80] was used to parameterise a scoring matrix in most approaches [73,74,76–78]. It has been claimed that the RNA backbone can be described using a finite set of discrete conformers [79,81,82]. However, some clusters depicted by Yen et al. [78] and also used by Wang et al. [76] and Yang et al. [77] separate densely populated areas of the fragment space with no obvious groups in the data. Hence, the previously described problems of the discretization do apply here.

In the following, an approach to the structural alphabet design is presented, which is fundamentally different to the methods described previously. The objective is to avoid the detour of searching for separable groups in the fragment space and naturally occurring substitutions. Instead, the encoding is optimized directly for its purpose, which is finding similar backbone stretches in distinct molecules. This requires essentially four ingredients: A basic mechanism for the translation of backbone fragments to letters, a target function to measure the performance of an alphabet, a dataset for the parameter training and an optimization algorithm. For the definition

of the fragments, the idea of spaced *k*-mers [83] is adapted from sequence comparison in order to reduce redundancy between overlapping fragments. A fragment contains atoms of four nucleotides across a stretch of seven nucleotides along the backbone. An alphabet is a composition of fragments acting as centroids. Each centroid fragment corresponds to a distinct letter. Chains are encoded by assigning the letter of the closest centroid to each fragment, using RMSD as distance measure.

The goal of the optimization is to find a set of centroids that performs well in finding similar structures. With all RNA structures in the PDB there is a large number of fragments to choose from. To search for a good set of fragments we need a defined performance measure. Here, an alphabet is found to be good if it finds many good and few bad matches in a large set of chain pairs. The evaluation therefore requires a method to find matches, a dataset to search for matches and a function to evaluate these matches. Enhanced suffix arrays [84] are used to search for exact substring matches (longest common substring) in a set of chains collected from the PDB. The difficulty is to estimate the quality of these matches. One has to trade similarity against the length of the match. The cost function used here compares the RMSD of the match to an estimate of what is expected for a random pair of chains of the given size. The last ingredient is the optimization procedure. A population based algorithm with Boltzmann weighted sampling and temperature annealing [85] is used. A set of alphabets referred to as population is iteratively refined. New alphabets are generated through the application of random changes at the beginning of each iteration. Subsequently, a new population is sampled with probabilities based on the abilities of the alphabets to find good matches. With this methodology the alphabet size can be constrained to a specific value or subject to the optimization. Some results are presented for both cases in the following. The composition of an example alphabet, average properties of matches and some individual matches are analyzed.

3.2. Methods

This section starts with the description of all the ingredients and preparations for the optimization. The optimization procedure is described subsequently.

3.2.1. Datasets

All RNA structure models from a list of chains with pairwise sequence identity smaller than 95 % [50] were downloaded from the PDB [12]. Many of these structure models are missing residues within the model. The backbone chain is thus not continuous. During the optimization exact matches between sequences of letters are optimized, but sequence matches across gaps in the chain are not meaningful. Therefore, all chains from the initial list were split at each position where nucleotides are missing. This new list of continuous chains was used to create the following three datasets, used for

- description of RMSD distributions for random chains,
- training,
- testing.

The first dataset was assembled to sample distributions of RMSD values for a range of different match lengths. This set contains all chains which are at least 500 nucleotides long. It will be referred to as the NR500 dataset. For the optimization the continuous chains were cut again to pieces with a length of not more than 250 nucleotides. Chains smaller than 50 nucleotides were removed from the set. This list of subchains was then divided into two distinct sets by random sampling with 80 % of the chains used for training and 20 % used for testing.

3.2.2. Significance of backbone similarity

Distributions of RMSDs for matches between two chains of a given length were estimated using a sampling strategy. 10^6 pairs of subchains were sampled with replacement for each chain length from 7 to 250 nucleotides. Chain pieces were sampled from a uniform distribution across all subchains with the given length in the NR500 set. The RMSD was computed for every pair. Mean μ_l and standard deviation σ_l of the RMSDs were computed for every group, with l being the chain length. Based on these we can compute a Z-score

$$Z = \frac{RMSD - \mu_l}{\sigma_l} \tag{3.1}$$

for any ungapped match with a length of l nucleotides.

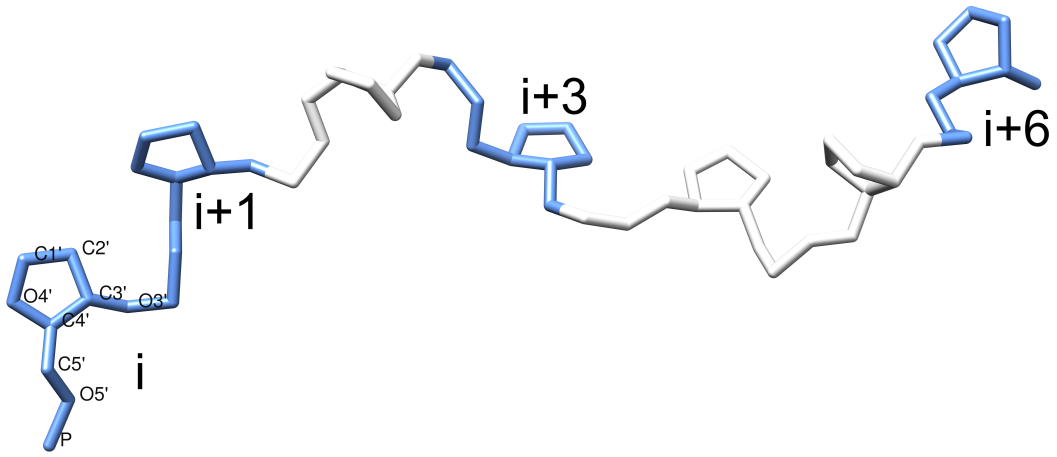

Figure 3.1.: RNA backbone fragment used for the assignment of structural alphabet letters. Atoms colored blue are considered for the assignment of a letter and white ones are ignored.

3.2.3. Translation of backbone fragments to letters

Structural alphabet letters of this alphabet are defined for backbone fragments spanning 7 nucleotides (Fig. 3.1). 4 nucleotides of each fragment are considered for the assignment of the letter. Starting at position i these are the nucleotides at positions i, $i+1$, $i+3$ and $i+6$. Every letter in the alphabet is represented by one specific centroid fragment. To translate a backbone fragment to a letter, RMSDs are computed between the fragment and the centroid fragment of every letter in the alphabet. The RMSDs are computed with respect to the P, O5′, C5′, C4′, O4′, C3′, O3′, C2′ and C1′ atoms of the two fragments. Missing atoms are simply ignored in the computation. The letter with the smallest RMSD is then assigned to the fragment.

3.2.4. Computation of substring matches

The longest common substring (LCS) of two strings is computed using enhanced suffix arrays [84]. The enhanced suffix array comprises a generalized suffix array and a longest common prefix (LCP) array. To build a generalized suffix array, a unique (sentinel) character is appended to the end of both strings. The two strings are concatenated and a suffix array is computed on the merged string. sais-lite [86], a fast implementation of induced sorting [87], is used for the suffix array construction. To compute

the LCP array, the algorithm by Kasai et al. [84] was implemented. The LCP array stores the size of the longest common prefix for all successive suffix pairs from the suffix array. The LCS can therefore be found by searching the largest entry in the LCP array where the two neighboring suffixes originate from the two different strings. The worst case runtime to build the enhanced suffix array and find the LCS is $O(l_1 + l_2)$ for two strings with lengths l_1 and l_2.

3.2.5. Preclustering fragments

A list of letter candidates for the optimization was created by clustering all fragments in the training set using the affinity propagation algorithm [88]. The negation of the RMSD on the previously described atom pattern (section 3.2.3) was used as affinity value between two backbone fragments. A preference value of -2.0 was used. The clustering was done in two iterations because clustering all fragments at once requires too much memory. For the first iteration, the set of all fragments was split into four subsets. Affinity propagation was performed and yielded a list of exemplars for each subset. These fragment lists were merged and clustered again. The cluster centers emerging from this clustering were used as candidate fragments for alphabet letters in the optimization.

Visualization of the clusters

The following steps were conducted to visualize the outcome of the clustering. A new coordinate space is constructed based on similarity between fragments. For each fragment, a vector of size N is computed comprising the RMSDs to the N candidate fragments. Merging the vectors of the candidate fragments yields an N^2 matrix. Principal component analysis (PCA) [89] was performed on this matrix yielding N eigenvectors. For visualization, vectors of all fragments were projected onto the first three eigenvectors.

3.2.6. Optimization

This section describes the search for optimal RNA structure alphabets. The overall algorithm is described first. Afterwards, the different parts of the procedure will be described in detail. The main routine is a population

based search heuristic with Boltzmann weighted sampling at the end of each iteration (Algorithm 1). The space in which the optimal alphabet is searched during the optimization will be referred to as alphabet space. It is spanned by the list of letter candidates L which is computed as described in section 3.2.5. A position in this space is an alphabet $A \subseteq L$. The search space comprises all possible alphabets, either of arbitrary size or constrained to a predefined size. Moves in this space are implemented as changes of an alphabet referred to as mutations. A population P is a set of alphabets. The algorithm starts with the creation of an initial population P. Subsequently, this population is refined for N_{Gen} iterations. In each iteration the population is enlarged using mutations to create new alphabets. The quality of every alphabet is assessed by computing the cost function on the training data D. Cost values and temperature-like parameter T are then used to sample a new set of alphabets from P that replaces P. Finally, the best alphabet is chosen from P based on the cost function.

Algorithm 1 Alphabet Optimization

procedure AlphaOpt(D, L, N_{Gen}, $N_{Survivors}$, $N_{Offspring}$, T_{Start}, T_{End})
  $P \leftarrow initialPopulation(L, N_{Survivors})$
  for $i \leftarrow 1$ to N_{Gen} **do**
    $P \leftarrow produceNextGeneration(P, L, N_{Offspring})$
    $T \leftarrow getTemperature(i, T_{Start}, T_{End})$
    $C \leftarrow computeCosts(P, D)$
    $P \leftarrow sampleSurvivors(P, C, T, N_{Survivors})$
  end for
  return $getBestAlphabet(P)$
end procedure

Moves in alphabet space

Three different moves were implemented to randomly mutate the alphabets. To mutate an alphabet, we either **add**, **remove** or **replace** a letter. For a given set of candidates L, we have an alphabet $A \subseteq L$ and a list of candidates not chosen yet $A' = L \setminus A$. $|A|$ is the number of letters in A. To enlarge the alphabet, a letter $a \in A$ is drawn with probability $\frac{1}{|A|}$. The alphabet is reduced by sampling a letter $b \in A'$ with probability $\frac{1}{|A'|}$. For the replacement the letters a and b are drawn with the given probabilities and exchanged between A and A'.

Generating new candidate alphabets

At each step of the optimization $N_{Offspring}$ new alphabets are created from each of the $N_{survivors}$ alphabets that survived the previous iteration (Algorithm 2). The new generation comprises the original and the mutated alphabets. A new alphabet is generated by copying its predecessor and applying exactly one of the moves described previously. Two different search strategies were used in this work. The first strategy includes all three moves. The optimization starts with empty alphabets ($A = \emptyset \forall A \in P$). When generating strings with this alphabet every fragment is assigned the same letter. It is thus equivalent to an alphabet comprising one letter. For every mutation one of the three moves is chosen randomly. If possible, each move has the same probability to be chosen. When $A = L$ letters can only be removed and $A = \emptyset$ implies that the only possible move is adding a letter. In the other strategy the size of the alphabet is fixed. The optimization starts with a random choice of letter candidates. At each move a letter is replaced as described previously.

Algorithm 2 Increase the population

```
1: function produceNextGeneration(P, L, N_Offspring)
2:     P^New ← P
3:     N_Survivors ← |P|
4:     for j ← 1 to N_Survivors do
5:         for k ← 1 to N_Offspring do
6:             P^New ← P^New ∪ {copyAndMutate(P_j, L)}
7:         end for
8:     end for
9:     return P^New
10: end function
```

Annealing schedule

An exponential annealing schedule [85] is used for the temperature. For an optimization with N_{Steps} iterations, initial temperature T_{Start} and final temperature T_{End}, the temperature at iteration i is

$$T_i = T_{Start} \left(\frac{T_{End}}{T_{Start}} \right)^{\frac{i-1}{N_{Steps}-1}}. \tag{3.2}$$

Cost for a match

The cost for a match of two continuous backbone pieces is given by

$$cost = \tanh(\frac{a}{\sigma_l}(Z - s))l, \tag{3.3}$$

where Z is computed according to equation 3.1 using the RMSD and the length l of the match. a and s are arbitrary constants. The term $\frac{a}{\sigma_l}$ adapts the slope of the sigmoid. a was set to 20 in all optimizations. s shifts the x-ordinate. It defines the tipping point for a good match being rewarded and a bad match being penalized. s was set to 2 in all optimizations. 2σ is a common choice for a significance threshold for Gaussians [59]. However, the choice is rather arbitrary. Both parameters were found by initial trial and error experiments. The parameterized score function is depicted in Figure 3.3.

Evaluate candidate alphabets

The quality of an alphabet is assessed by computing and evaluating matches between the chains from the training set. The procedure to compute the target function for each alphabet A in the population P is shown in Algorithm 3. First, each chain in the training set D is translated to a string. Subsequently, the LCS match m is computed for all possible string pairs S_i and S_j from the set of Strings S as described in section 3.2.4. The RMSD for the corresponding segments of the chains D_i and D_j is computed using Kabsch's superposition algorithm [90]. RMSD and length of the match are then used to calculate the cost according to equation 3.3. The cost of an alphabet is the average cost of all matches computed with this alphabet.

Sample survivors

In each iteration the population P is replaced. The new set of $N_{survivors}$ alphabets is sampled from P with replacement. Boltzmann weights are used to assign probabilities to the different alphabets. The probability to sample the alphabet P_i is

$$p_i = \frac{e^{-C_i T^{-1}}}{\sum_{j=1}^{|P|} e^{-C_j T^{-1}}}, \tag{3.4}$$

with temperature T and cost C_i and C_j for alphabet P_i and P_j respectively.

Algorithm 3 Evaluate alphabets

```
1: function evaluateAlphabet(A, D)
2:     S ← chainsToStrings(A, D)
3:     c ← 0
4:     for i ← 1 to |D| do
5:         for j ← 1 to i − 1 do
6:             m ← lcsMatch(S_i, S_j)
7:             l ← getMatchLength(m)
8:             RMSD ← kabschSuperposition(m, D_i, D_j)
9:             c ← c + cost(RMSD, l)
10:        end for
11:    end for
12:    return 2c / (|D|^2 − |D|)
13: end function
14:
15: function computeCosts(P, D)
16:    C ← ∅
17:    for i ← 1 to |P| do
18:        C ← C ∪ {evaluateAlphabet(P_i, D)}
19:    end for
20:    return C
21: end function
```

Time and space requirements

Computational requirements of the different parts of the optimization algorithm are analyzed in this section. Time requirements will be analyzed first. We will start with the generation, mutation and resampling of alphabets and then analyze the evaluation of the cost function. Memory consumption is analyzed afterwards.

In the implementation presented here, an alphabet is implemented as an array which contains all letters from the candidate list L. The array is sorted. The letters of the alphabet are at the beginning and the others are at the end. The size of the alphabet is stored in an extra variable. All three moves are implemented as two letters changing places and - if necessary - increasing or decreasing the number of letters chosen. A move requires thus $O(1)$ time. However, when generating a new alphabet, the predecessor is copied and then mutated. Copying an alphabet requires $O(|L|)$ time. Therefore, the

initialization of P uses $O(N_{Survivors}|L|)$ and the production of new alphabet candidates $O(N_{Survivors}N_{Offspring}|L|)$. In each iteration a new population is sampled with replacement. The probabilities for the alphabets are stored in a cumulative array. Sampling an alphabet is done by generating a random number followed by a binary search in this array. The sampling of $N_{Survivors}$ alphabets thus requires $O(N_{Survivors}\log(|P|)$ time and the copying requires $O(N_{Survivors}|L|)$ time. Given that $|L| \gg |P|$ we can conclude that the most expensive step described so far is generating new alphabet candidates.

The most demanding part of the optimization is the calculation of the target function for every alphabet in each iteration. A chain D_i from the training set D has length l_i. In the following we will ignore the fact that strings are slightly shorter than the corresponding chains. We will use l_i for the length of D_i and S_i, since this will not change the runtime complexity in O notation. To compute the set of strings S, every fragment in the training set is translated to a letter. Since the maximum size of an alphabet is the number of letter candidates $|L|$, finding the closest letter for a fragment has $O(|L|)$ time complexity. F is the group of all fragments found in D. Translation of all chains to strings thus requires $O(|F||L|)$ time. This is reduced to $O(|F|)$ if the alphabet size is constant. However, even for optimizations where the alphabet size is not fixed, the alphabets are smaller than 10 letters most of the time (section 3.3.3). Thus, the runtime can also be assumed to be $O(|F|)$ in practice. The computation of the LCS match between the strings S_i and S_j is $O(l_i + l_j)$ (section 3.2.4). The matching chain parts are superimposed with the Kabsch algorithm. The superposition and the calculation of the RMSD require time linear to the length of the match. We get $O(\max(l_i, l_j))$ time complexity. Since computing the score given the length and RMSD of the match requires $O(1)$ time, the overall time complexity for the computation and evaluation of one string match is $O(l_i + l_j)$. Each chain D_i is compared to $|D| - 1$ other chains. Since the contribution to the runtime for a match computation is linear, the contribution to all match computations is $O(|D|l_i)$. The sum of the contributions of all chains is $O(|D|\sum_i^{|D|} l_i) = O(|D||F|)$. To compute all strings and matches we thus need $O(|F|) + O(|D||F|) = O(|D||F|)$ time. For $|P|$ alphabets we get $O(|P||D||F|)$ and for N_{Gen} iterations this becomes $O(|P|N_{Gen}|D||F|)$. Previously, producing new alphabet candidates was the most demanding task described. Because $|P| = N_{Survivors}N_{Offspring} + N_{Survivors}$ we can simplify the runtime of this step to $O(|P||L|)$. Now let us compare this to the evaluation of the cost function. The set of letter candidates L is a subset from the fragment set F obtained by clustering as described in

section 3.2.5. The clustering drastically reduces the number of fragments (section 3.3.2). We have $|F| \gg |L|$ and it follows that $|D||F| \gg |L|$ and therefore $O(|P||D||F|) \gg O(|P||L|)$. Hence, the evaluation of the target function is the bottleneck regarding the speed of the computation. To speed up the computation it was parallelized. Computations of the target function for different alphabets were distributed on distinct threads.

The algorithm requires memory for chains, sequences and alphabets. Chains and sequences can be kept in $O(|F|)$ space. Storing $|P|$ alphabets requires $O(|P||L|)$. The total space required is thus $O(|F| + |P||L|)$. In the implementation presented here the RMSD values between the letter candidates in L and the fragments in F are precomputed. This requires $O(|F||L|)$ space. This is not necessary but convenient since the required memory is not a bottleneck here. The required space becomes $O(|F||L| + |P||L|)$.

3.2.7. Parameter choices

To find the parameters used in this work some initial trial and error experiments were performed. The following parameters were found to produce adequate results and therefore have been used to produce the results described in section 3.3. T_{Start} and T_{End} were set to 1.0 and 0.2 respectively. $N_{Survivors}$ was set to 12 and $N_{Offspring}$ to 4. The optimization was done in 1000 iterations (N_{Gen}). Every 10th iteration the average match length, RMSD, cost and alphabet size were collected for training and test set. Optimizations were done with a fixed and flexible alphabet size. Alphabets with fixed size of 5, 6 and 7 letters were optimized. Each optimization was started three times with different seeds for the random number generator.

3.2.8. Implementation and computation

Code was written in C and Python 3. GCC C-compiler version 7.4.1 [91] and the Python interpreter version 3.6.5 [92] were used. The interface was built using SWIG 4.0 [93]. OpenMP [94] was used to implement shared memory parallelization. For the affinity propagation clustering and the PCA the implementations from scikit-learn [95] were used. Code from sais-lite version 2.4.1 [86] was used for suffix array construction. The computations were done on a high performance computing cluster. The affinity propagation clustering was computed on machines with at least 200 Gigabytes RAM.

Optimizations were computed on 12 threads with shared memory of at least 12 Gigabytes.

3.3. Results

3.3.1. RMSDs of random matches

Pairs of chain pieces were sampled from larger chains to estimate the distributions of RMSDs for random RNA chains of a given length. For chain pairs between 7 and 250 nucleotides length the average RMSD is continuously increasing with the chain length while the rate of change is decreasing (Fig. 3.2). This has implications on the cost function given in equation 3.3 (Fig. 3.3). The cost function is a Z-score wrapped in a logistic function multiplied by the length of the match. The x-intercept is the turning point for a match from being considered favorable to unfavorable in the optimization procedure. The increasing mean of the distributions shifts this point to larger values. This means the tolerance with respect to RMSD increases with the length of the matches. Multiplication with the match length further means that larger matches contribute more to the overall cost.

3.3.2. Clustering backbone fragments to produce alphabet letter candidates

If every 7-mer in the training dataset is considered as a potential centroid for a letter, there are 156 782 letters to choose from in order to assemble an alphabet. This imposes some problems on the optimization. The search space is huge which causes the system to require more steps in order to converge to an optimum. Furthermore, it is not possible to precompute all distances between the fragments of the chains in the training set and the alphabet candidates and keep them in the memory. One would have to compute the distances between the chosen candidates for every alphabet at every step of the optimization. Therefore, the fragments in the dataset were clustered using affinity propagation to produce a smaller set of letter candidates. The original set of fragments was separated into four subsets of equal size. The cluster centers were merged and clustered again producing 4637 clusters.

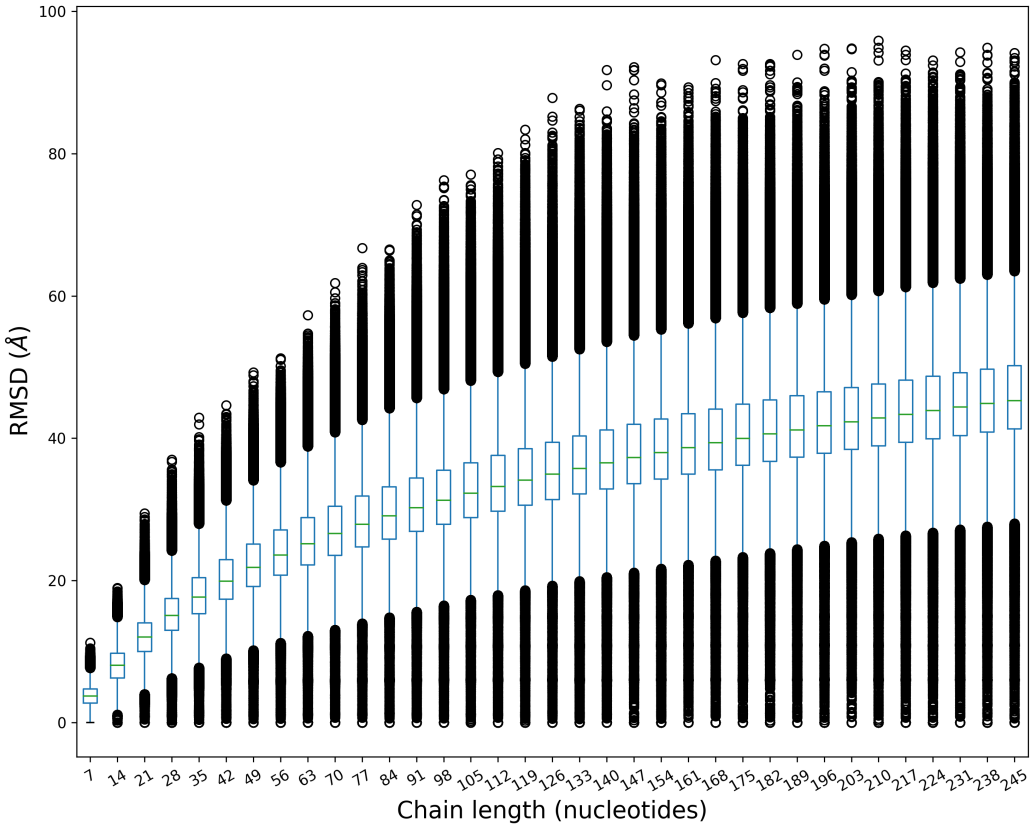

Figure 3.2.: Distributions of RMSDs for pairs of sampled chain pieces. Pairs were sampled for chain lengths 7 to 250 nucleotides. Every seventh distribution is depicted here. Boxes show medians, upper and lower quartiles. Whiskers stretch out to 1.5 interquartile ranges. Outliers are drawn as circles.

PCA was performed to roughly sketch and inspect the outcome of the clustering. Projection onto the first three principal components preservers 60, 23, and 5 % (total 88 %) of the variance of the distances (RMSD) between the cluster centers. Visual inspection shows a core area densely populated with cluster centers and a few cluster centers on sparsely populated areas (Fig. 3.4).

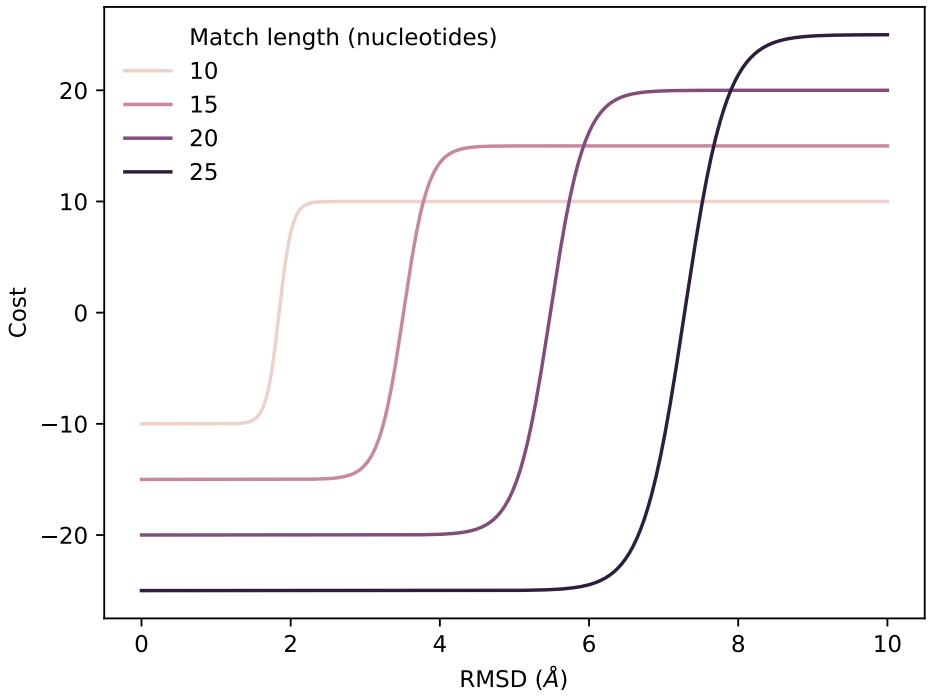

Figure 3.3.: Cost function for the alphabet optimization for different match lengths.

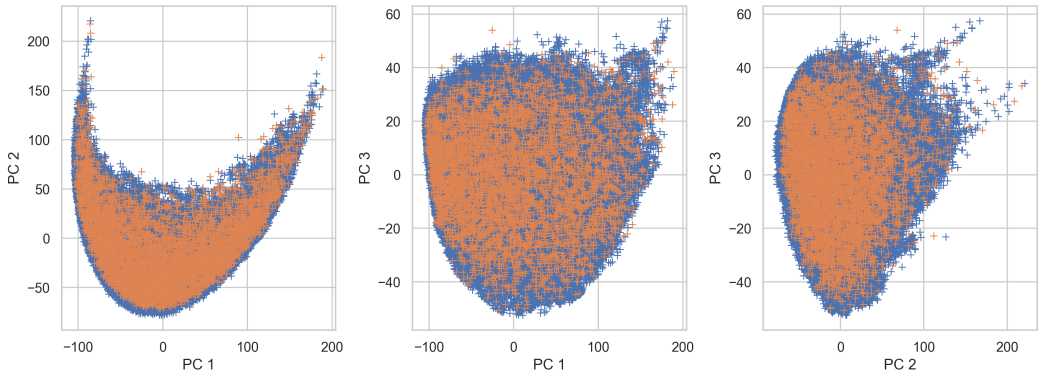

Figure 3.4.: Projection of distances between backbone fragments (blue) and alphabet candidates (orange) on the first three principal components.

3.3.3. Alphabet optimizations

Optimizations were performed with flexible or fixed alphabet size. Alphabet size and random seed were the only parameters that varied between the different optimization runs. An alphabet size of five, six and seven letters was used in optimizations were the size was fixed. Each optimization was done three times with different random seed. An annealing procedure was used for a more efficient search (Fig. 3.5 A).

Trajectories

Optimizations ran 1000 iterations. At every tenth iteration the average cost, RMSD and match length were computed and collected for the training and the test set. Furthermore, the average number of letters was collected when the alphabet size was not fixed. To get an idea about the course of such an optimization the trajectories of two cases are analyzed here. The first trajectory is from an optimization without restraints on the alphabet size (Fig. 3.5 B-E). The second one is from an optimization on alphabets restrained to a size of six letters (Fig. 3.5 F-H). The random seeds in these two optimizations were set to one and two respectively.

The optimization of alphabets with flexible size starts with alphabets of size zero. All fragments will be assigned to the same letters. Matches computed on these strings are meaningless. The energy is very high due to long matches with high RMSD. The average alphabet size increases to a value close to eight within the first 200 steps. Afterwards, this value stays close to eight for the rest of the optimization. With the first letters being added to the alphabets, the average cost drops very fast within the first ten steps (not shown in the figure). This comes with decreasing RMSD and match length. In the following steps the cost function still decreases very quickly compared to the rest of the optimization. This comes with a sharp increase of the average match length from less than 16 to values close to 18 nucleotides. In the following $\sim$ 500 steps cost, RMSD and match length further decrease on average. Afterwards, in the second half of the optimization, the cost computed on training and test data is further improved, but to a smaller amount than in the first half. The significant downwards trend for RMSD and match length, however, is not abundant anymore.

In the trajectory of the six letter alphabet the cost value decreases the most in two different phases of the optimization. The first overall drop in the

cost function is within the first 100 steps, the second drop happens between step 300 and 400. Afterwards, the value still improves on the training set but to a lesser extend than previously. However, this does not happen when computed on the test set. Both match length and RMSD drop within the very first steps. The match length has its minimum below 17.5. Afterwards both increase and start to decrease again around step 200. The final average RMSD computed on the training set is close to 6.0 Å thus lower than the start value of approximately 6.6 Å. The final average match length is approximately 18.6 nucleotides and therefore a little bit larger than the initial value of roughly 18.5 nucleotides. Hence, both values improve during the optimization.

Final scores

Table 3.1 lists the final average cost, RMSD and match length for each optimization run computed on the training and the test set. Furthermore, a line with mean values for the different alphabet sizes was added. Results for only three different random seeds per alphabet size may not be enough to compute meaningful statistics. We can still make a few observations. The lowest cost value is obtained from the optimization with fixed alphabet size of seven nucleotides using one as random seed. This yields an average cost of -10 on the training set and -9.5 on the test set. The optimization with seed two, on the other hand side, only achieves a score of -7.6 on the training set. This is the worst performance on the training set observed across all optimizations. Hence, the results of the different alphabets obtained from the different runs are overlapping.

Frequencies of different letters

Figure 3.6 shows the backbone fragments selected in the optimization of a six letter alphabet using seed two. The letters were named A to F. The fragment of letter E has a shape which is typical for RNA helices. The frequencies of the letters were computed for the chains of the training set (Fig. 3.6). The most abundant letter is the helix letter E. 66 % of all fragments are assigned to E. The second most frequent letter is A covering 12 % of the letters. Hence, E occurs 5.5 times as often as A.

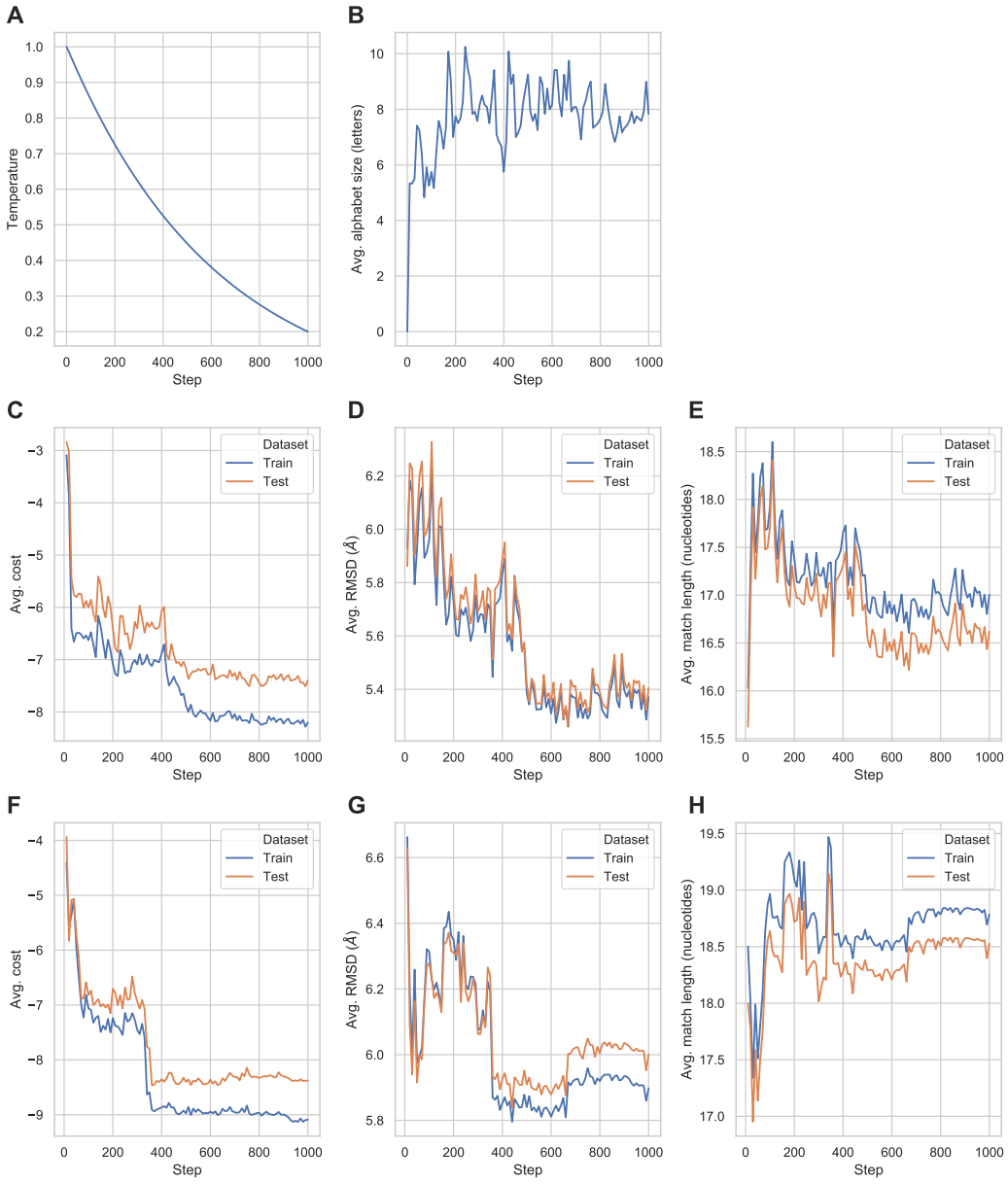

Figure 3.5.: Averaged properties for an optimization with flexible (B-E) and one with fixed alphabet size (F-H). Both optimizations used an exponential cooling scheme (A) and all measures were computed for the training and test set. Optimizations with flexible alphabet size initially have very high RMSD, match length and cost. Therefore, the initial values (step 0) are not shown to improve the visibility in plots C to E.

		Cost				RMSD (Å)				Match length (nucleotides)			
		5	6	7	Flex	5	6	7	Flex	5	6	7	Flex
	Seed												
Train	1	-7.9	-7.8	-10.0	-8.3	6.0	5.9	6.2	5.3	18.6	18.2	20.1	16.8
	2	-7.8	-9.1	-7.6	-9.1	5.9	5.9	5.8	5.9	18.1	18.8	17.9	18.7
	3	-8.3	-7.9	-8.9	-8.2	6.3	5.9	6.0	6.1	19.5	18.4	18.9	18.9
	Mean	-8.0	-8.3	-8.8	-8.5	6.0	5.9	6.0	5.7	18.7	18.5	19.0	18.1
Test	1	-7.1	-6.9	-9.5	-7.6	6.0	5.9	6.2	5.3	18.2	17.7	20.0	16.5
	2	-6.5	-8.4	-7.5	-7.9	5.9	6.0	5.8	5.9	17.4	18.6	17.7	18.2
	3	-7.9	-6.2	-7.9	-7.3	6.3	6.1	6.1	6.1	19.1	18.0	18.6	18.6
	Mean	-7.2	-7.2	-8.3	-7.6	6.1	6.0	6.0	5.8	18.3	18.1	18.8	17.7

Table 3.1.: **Final scores from alphabet optimizations.** The different columns are sorted and named by the size constraints on the alphabets. The mean was computed over the three runs for each case.

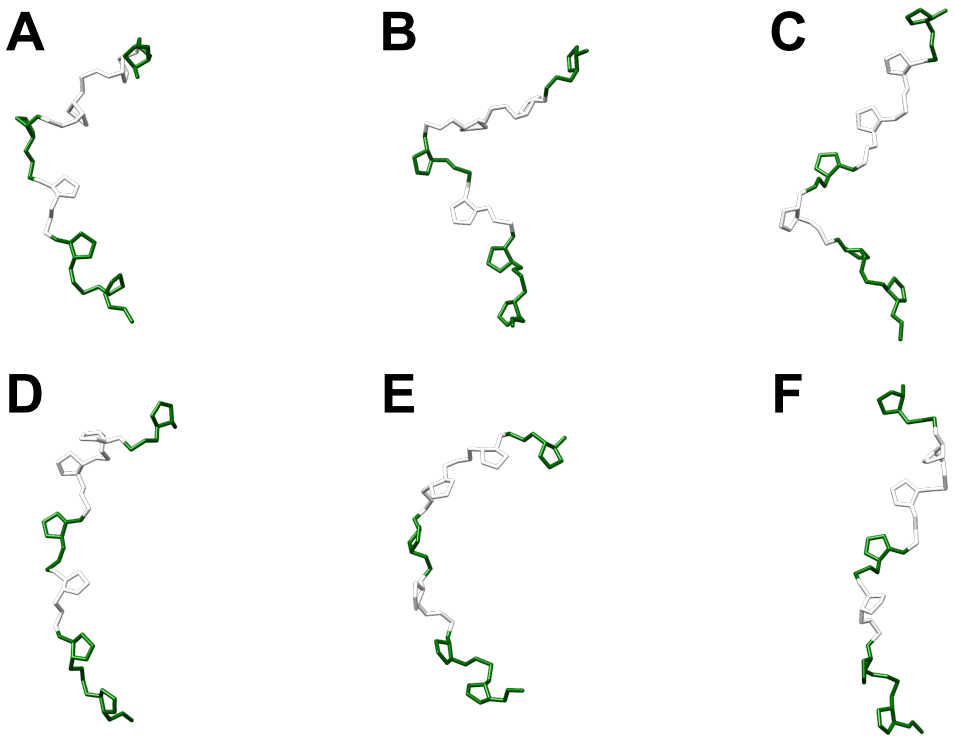

Figure 3.6.: Fragments of a six letter alphabet. The green positions are used for the assignment of the letter using the RMSD. The white parts are ignored in this computation.

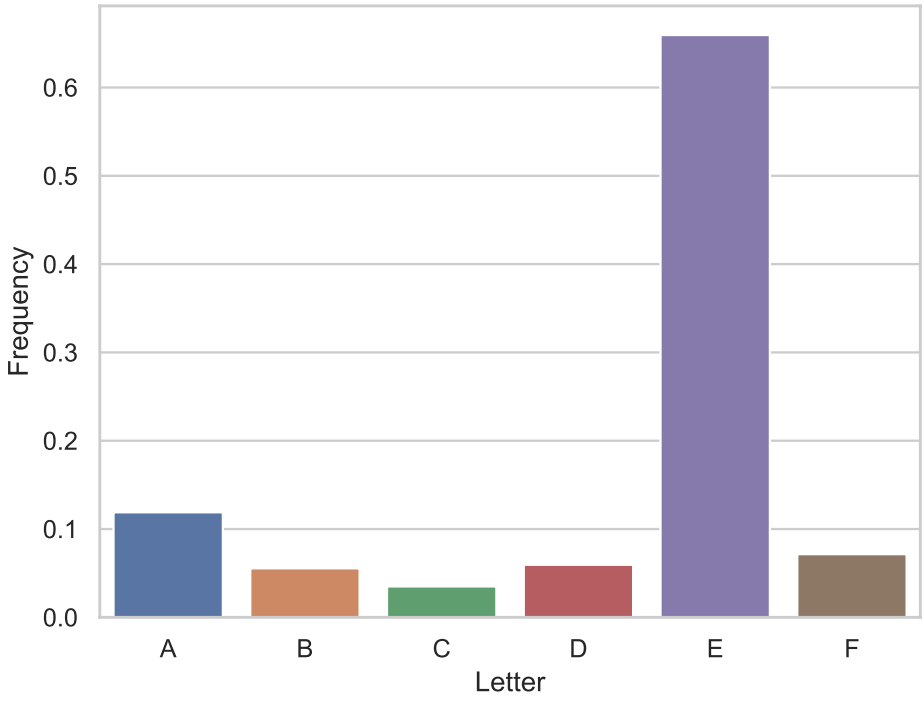

Figure 3.7.: Distribution of letters from a six letter alphabet. The fragments corresponding to the letters are shown in Figure 3.6.

Matches

A closer look on the individual matches can help to further understand and evaluate the performance of an alphabet. As an example, matches computed with the six letter alphabet described in section 3.3.3 are examined here. RMSD, match length and cost value were computed on the training and the test data (Fig. 3.8). Between the chains of the training set 68 % of the matches were rewarded with a negative cost value while 32 % were penalized with a positive cost value. For the test set 65 % of the matches were rewarded and 35 % penalized. In both cases less than 0.01 % of the matches did not have any letter in common and therefore were assigned cost zero.

Three example matches with different quality are presented here as an attempt to illustrate the range of matches found with the optimized alphabet (Fig. 3.8 and Fig.3.9). The matches shown are selected from the test set and have thus not been considered during the optimization. They are therefore

realistic examples for later use cases. Matches were selected that are significantly longer than most of the matches with a similar Z-score. This was done because similarities and differences are more visible for longer matches. The first match is 102 nucleotides long with 1.7 Å RMSD. Therefore, the match is rewarded with a cost value of -102.0. The two structures are very similar and can be superimposed closely (Fig. 3.9 A). The second match is 126 nucleotides long with 17.3 Å RMSD. It is thus a little bit longer but has a much larger RMSD compared to the previous example. The RMSD is still well below $\mu_{l=126} - 2\sigma_{l=126}$. Therefore, it is considered to be significantly better than expected for a random match and rewarded with a cost value of -123.4. Both chains are from long helical stretches. Each of them is only half a helix with the complementary strand missing. The matching strings are both completely made of the repeating letter E, which has the typical helical shape (section 3.3.3). At visual inspection after superposition one can see that the overall shape of the two chain pieces is similar. The matching nucleotides, on the other hand, are not placed on top of each other as seen for the previous example (Fig. 3.9 B). The third match is 61 nucleotides long with 24.4 Å RMSD. This RMSD is larger than $\mu_{l=61} - 2\sigma_{l=61}$. The match is penalized with a cost value of 61.0. Again, both strings only comprise letter E. The RMSD is higher than in the previous example, because one of the two matched chain pieces is bend while the other one is rather straight (Fig. 3.9 C).

3.4. Discussion

A central difficulty of the optimization described here is that one has to juggle two objectives. Large continuous substring matches are preferred to shorter ones. Longer matching segments may add more confidence for the indication of similar function or evolutionary origin. They can also be more helpful when assigning substructures in a global comparison. Obviously, this is only the case if the matched substructures really do have a similar shape, which is herein expressed in terms of RMSD. The challenge is therefore to trade match length and RMSD in a sensible way. This is not a trivial task as there is no obvious threshold on the RMSD to distinguish a good from a bad match. The problem becomes even more difficult considering that there is a non-linear dependence between the RMSD and the match length (section 3.3.1). The first step towards an optimization is therefore the definition of a target function that implements a sensible compromise

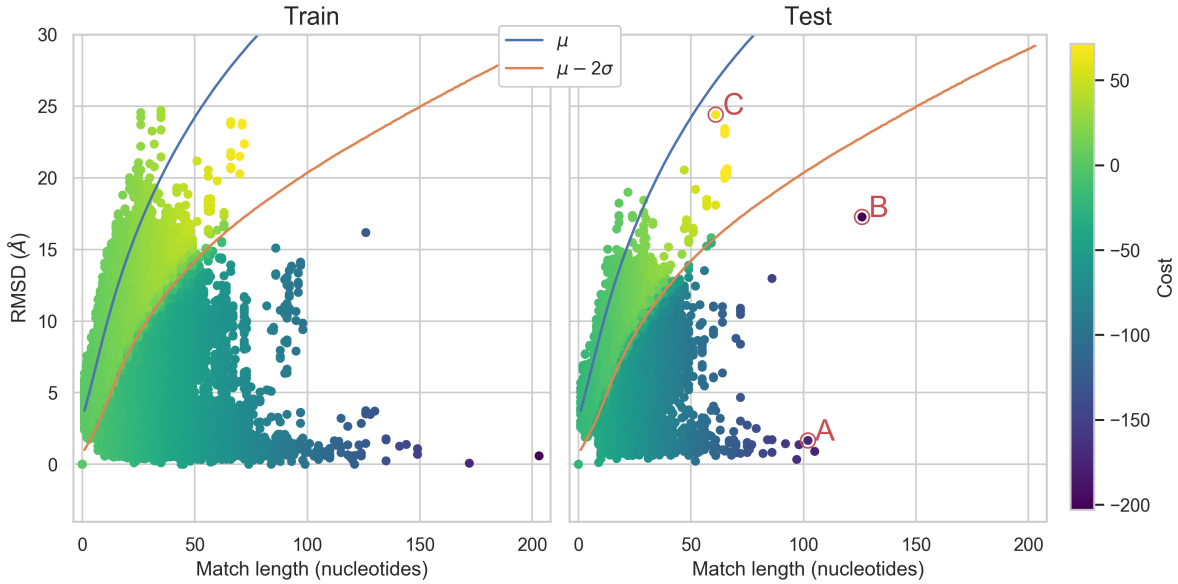

Figure 3.8.: RMSD, match length and cost for all matches between chains in the training and in the test set at the end of an optimization. μ and σ are the mean and standard deviation of RMSD distributions from sampled chain pairs. The $\mu - 2\sigma$ line marks the x-ordinate of the cost function, where the cost value switches from a negative reward to a positive penalty. The matches shown here were computed with a six letter alphabet. The superimposed structures for the highlighted matches A, B and C are shown in Figure 3.9.

between those two measures. The cost function used in this work rewards matches which have a significantly lower RMSD than expected for a random match and penalizes matches that do not fulfill this requirement. The reward or penalty is weighted by the length of the match. The benefit of this approach is its simplicity. No additional information about the relation of the specific structures in the training set is required. It can only be applied to a match of length l if a distribution of RMSDs has been sampled for l as described in section 3.3.1. This is not a problem for our optimization because the maximum match length is constrained to the maximum length of the chains in the training set, which are not longer than 250 nucleotides (section 3.2.1).

An alphabet cuts the continuous fragment space into discrete chunks. Borders are defined by the choice of the centroid fragments. The objective of the optimization is now to find an arrangement that optimizes the previously

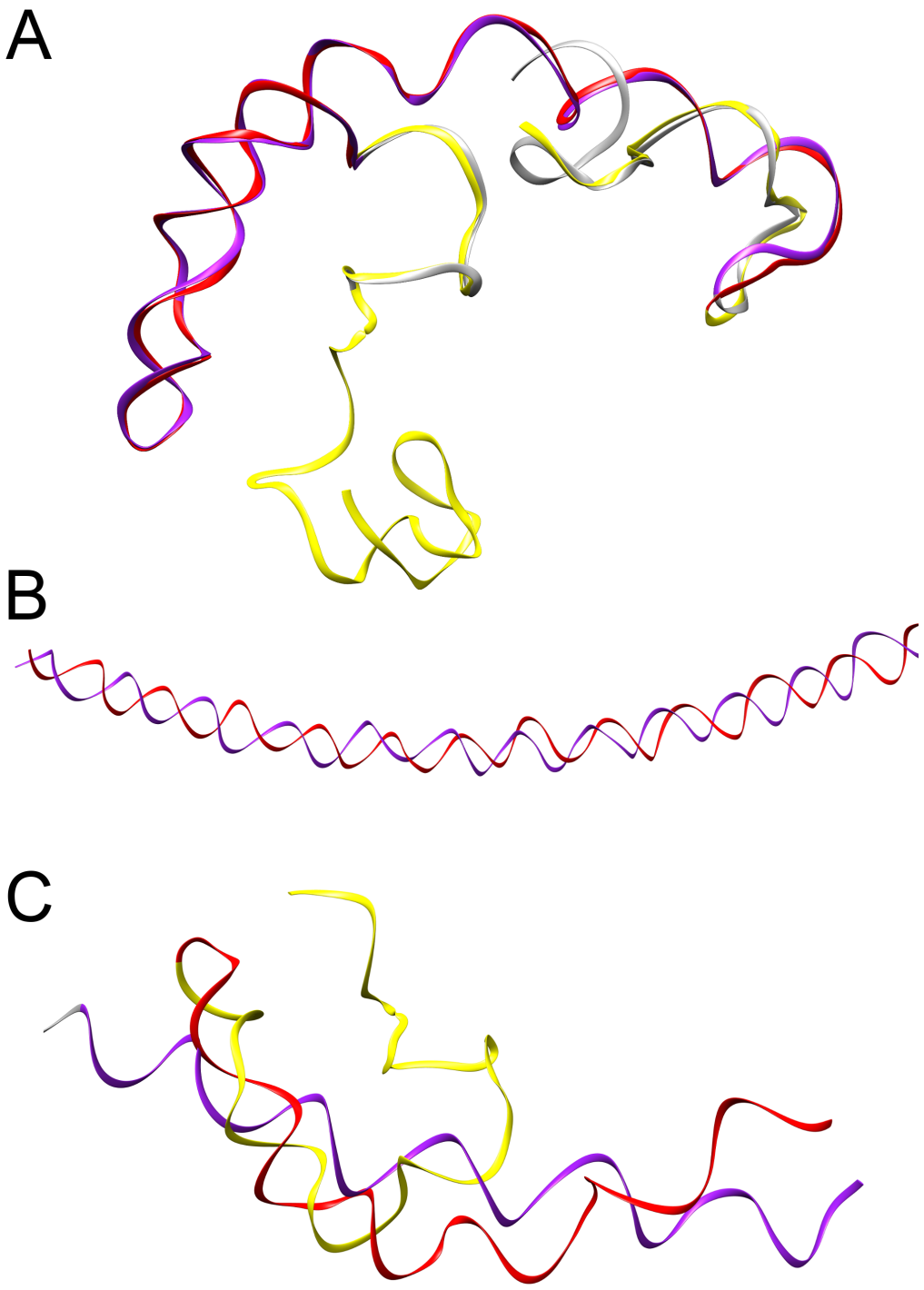

Figure 3.9.: Superimposed chains for matches highlighted in Figure 3.8. The chain segments colored purple and red belong to the matching substrings. The yellow and white parts are not matched.

described compromise of length and RMSD. The quality of the matches is directly linked to the division of the fragment space by the choice of the centroids. If the bins are too wide there will be unspecific matches. If they are too narrow, many similar fragments will be assigned to different letters and similarities will be missed out. This may be most easily understood when looking at alphabets with different size. Adding letters to an alphabet will add more boundaries. This will likely lead to the truncation of some matches in the training set and thus reduce the average match length. It will also reduce the RMSD at least a bit and more so if these matches had a large RMSD. Let us consider the match shown in Figure 3.9 C as an example. A straight helical stretch is matched to one with a hinge like structure. Both corresponding strings comprise only the letter E. A different letter assigned to at least one of the fragments in the hinge area would disrupt the match. The longest substring match would be shorter but also have a smaller RMSD. Having fewer letters in the alphabet on the other side would produce longer matches with higher RMSD. The most extreme case is a one letter alphabet which leads to arbitrary matches of maximum length and RMSDs distributed similar to those expected from random samples. The trajectory depicted in Figure 3.5 B-E clearly illustrates how the previously described effects drive the optimization. Within the first 200 steps the average alphabet size increases to about eight letters. The specificity of the matches increases rapidly as observed by a decreasing cost value due to decreasing average RMSD. In many steps the decreasing average RMSD concurs with a decreasing average match length.

The number of bins is obviously not the only factor. How the selection of the centroids improves the performance in finding matches is clearly visible for the trajectory depicted in Figure 3.5 F-H. The alphabet size does not change during this optimization. At the beginning six random centroid fragments are chosen for every alphabet. The performance increases significantly as shown by the decreasing cost value. At the end of the optimization there is an improvement on both objectives. The average match length is longer and the RMSD smaller for the final state compared to the initial state. Hence, the optimization definitely succeeds in searching for better alphabets with respect to the target function. We can make a similar observation on the trajectory of the optimization with a flexible number of letters (Fig. 3.5 B-E). From step 200 onward the average size of the alphabets oscillates around eight letters. The cost function still decreases. This indicates that the optimization finds better groups of centroids while a size close to eight letters seems to be optimal.

The quality of an alphabet is assessed by averaging the cost over all matches. This means there is another compromise to make. Finding a good overall score is achieved by balancing matches which are rewarded by the cost function against those that are penalized. The best score may be achieved tolerating small bad matches in order to enable large good matches. A typical distribution of the scores is shown in Figure 3.8. Therein, matches are found on both sides of the $\mu_l - 2\sigma_l$ line which marks the x-ordinate of the cost function. Changing the factor s in equation 3.3 would shift this border towards rewarding or penalizing more matches. Setting the boundaries too tight will eventually result in an optimization which is dominated by the removal of penalized matches with only little influence from rewarded matches. The alphabet size increases to very large numbers if it is not fixed and this leads to very short matches. The matches shown in Figure 3.9 help to get an idea for the implications of the parameter choice made in this work. Match A is a case that should very obviously be rewarded as the backbone stretches are very similar. It would even be beneficial to further extend this, but occasional interruptions of favorable matches due to the borders of the discretization are probably something that one has to live with. Whether match B should be rewarded or penalized may depend on ones point of view or the application. The overall shape is similar while the nucleotides are not superimposed closely. Match C is penalized and therefore a part of the compromise. This may not be harmful. The most important task of the alphabet is to assign corresponding pieces between a pair of structures. In this sense the alphabet only fails if it prefers a bad match over a good one. In practice this means on may have to assess the quality of matches using geometric or statistical measures, for example the RMSD or an E-value [96].

The quality of the matches obviously depends on the set of structures they are searched in. The test set was extracted from the data to check if the alphabets are over fitted to match very specific chains in the training set. We observed that most of the improvements on the training set are reflected by the performance of the test set (Fig. 3.5 C-H). An exception can be observed in Figure 3.5 F. From step 400 onward the cost computed on the training set is still slightly improving while it is not for the test set. However, this improvement is tiny compared to the changes from the steps before. Furthermore, the performance on the test set is also not getting significantly worse. We can therefore conclude that the results can be generalized assuming that the local backbone geometry of future RNA structures will not fundamentally differ from those used here.

The ability to either choose a fixed alphabet size or let the optimization find the best size can be a useful property. Fortunately, the implementation of both approaches is straight forward here. The automatic determination is especially useful because the choice of the letters is directly linked to the performance in finding similar structures. This would be far more complicated if not impossible for unsupervised methods. In a rugged or fuzzy landscape the choice to separate or join different parts of the fragment space is difficult to answer. Constraining the size of the alphabet, on the other hand, can be helpful if one tailors an alphabet for a specific purpose, for example when utilizing existing software that can only cope with a limited alphabet size. A fixed alphabet size also imposes a smaller search space and thereby could enhance the efficiency of the search. Several optimizations were performed with flexible and fixed alphabet size. The runs with fixed alphabet size used five to seven letters. Table 3.1 lists the average scores of the final states. Estimating the best choice from this table would not be very helpful because the ranges of the results are overlapping and there are only three samples for every case. A good example are the costs computed on the training data for the alphabets with size seven. The three cases contain the overall best and the worst result across all optimizations with cost values -10.0 and -7.6 respectively. Chances are good to find an alphabet that performs at least slightly better. To make a confident statement on the perfect alphabet size one would require more optimizations with different random seeds and a wider range of alphabet sizes. The practical implications of the differences may be limited though. As described previously, the choice of the parameters for the cost function is a compromise and to some extend arbitrary. Furthermore, the score is the average over all matches in the training data. As a consequence, an alphabet may perform better for some pairs of structures while another one is superior in other cases.

Besides the alphabet size there are more parameters that could be changed in an attempt to improve the results. The choice for most of these parameters was done based on the performance in some initial trial and error experiments. They were not optimized. The representation of the fragments for example is rather arbitrary. It was chosen to avoid redundancy while capturing a significant amount of the orientation of the local backbone. One could also try different parameters or even different methods for the selection of fragment candidates (sections 3.2.5 and 3.3.2). The method here was chosen in order to reduce the number of fragments but keep enough to roughly cover the whole conformational space. Furthermore, one could change the training and test data and a large number of optimization param-

eters, such as the population size, the number of new alphabets generated in each iteration, the number of iterations, the annealing schedule and the temperatures or the probability distribution used for sampling the next generation. One could even incorporate different moves similar to cross-over moves in genetic algorithms [97] or use a selection scheme [98] instead of sampling. One may spend a lifetime searching for perfect parameters. The practical impact may be limited for the same reasons mentioned before and also because using structural alphabets is a very coarse simplification in the first place.

If one intends to run a larger number of optimizations it would be sensible to think about the runtime of the algorithm. The bottleneck is the computation of the matches in order to assess the alphabet quality. One could try to reduce the number of computations by sampling a subset of chain pairs to compute the cost in every step.

Centroid fragments of an example alphabet are shown in Figure 3.6. One should be careful with the interpretation of this picture. Unlike fragments found with an unsupervised methods, the fragments shown here are not necessarily representative of naturally occurring conformers. Instead, they define a division of the fragment space which has been heuristically optimized with respect to matches found in the training data. However, the region of the fragment space assigned to letter E clearly includes the majority of helix-like backbone pieces. Since a large amount of most RNA structures has a helix-like shape it has a much higher abundance than the other letters (Fig. 3.7). Containing stretches of E leads to many matches between else unsimilar structures. One can exploit the knowledge about this distribution and pay more attention to more significant matches, which have a low probability to be found at random. An example application is described in chapter 5.

As a final remark, it may be interesting to note that the search heuristic described in this chapter shares many properties with genetic algorithms using Boltzmann weighted selection [98] and with population annealing [99]. However, unlike genetic algorithms no recombination moves are implemented and a sampling process is used to generate the next generation instead of a deterministic selection process. In contrast to the method used here, population annealing also re-equilibrates the system after sampling the next generation to ensure a valid canonical ensemble.

3.5. Conclusion

A new approach to the encoding of RNA structures to strings was presented in this chapter. The mechanism described here is specifically designed for the task of finding similarities between different chains. Unlike previous approaches it does not rely on any assumptions about naturally occurring clusters of fragments. Instead, the method is intended to choose the division of the fragment space in a manner that optimizes matches between different chains directly. Alphabets designed this way will be applied in the next chapters.

Chapter 4.

4 RNA structure alignments

4.1. Introduction

Pairwise alignments of macromolecular 3D structures generally serve two functions. The first is the overall assessment of similarity between a pair of structures. This can be used to derive similar function or evolutionary origin in the absence of sequence similarity [39, 100, 101]. Given a structure of interest one can search for similar existing structures in a database [102–105]. One can also take the comparisons one step further and compare all available structures against each other and use clustering methods to explore the space of known molecular structures [106–109]. The second task performed by structural alignment tools is to find the corresponding residues or nucleotides in two molecules. The exact knowledge of matching positions helps to detect regions which are conserved with respect to sequence or structure and those which are not. It can also be used to transfer knowledge, such as the position of binding sites, from one structure to another. Alignment tools may be used on their own followed by a manual inspection of the output. However, they can also be used in larger automated processes. An example for such an application is the modeling tool PRIME, which uses RNA and protein structure alignments to search for templates to use for building models of RNA protein complexes [110]. Recently, it was demonstrated how the alignment quality enhances the success rate of the tool [111].

There are far more experimentally determined 3D structure models available for protein than for RNA molecules[1]. Therefore, it is not surprising that

[1] 156 758 vs. 4684 files containing protein and RNA respectively found in the PDB [12] on February 6, 2020

alignment tools for proteins were developed earlier and are more abundant [112,113]. However, within the last one and a half decades several tools were specifically designed for RNA structure comparison [51,52,73,75–78, 111,114–120]. Most of the approaches adopt methods developed for proteins with specific adjustments made to suite the unique features of RNA.

A common approach for fast alignments is the adaptation of dynamic programming algorithms from the sequence comparison field. The macromolecular backbone is described in terms of small overlapping fragments. These fragments are compared based on some geometric descriptors. A matrix is computed with scores from the comparison of all fragments of one chain with all fragments of the other chain. A dynamic programming procedure such as the Smith-Waterman [35] or the Needleman-Wunsch algorithm [34] is then used to compute the optimal alignment with respect to the position specific scores. Several tools have been developed for RNA structure comparison using this approach [51,52,73,75–77,114]. They mainly differ in the representation of the fragments. The scoring function used in DIAL is a weighted sum of contributions from torsion angles, sequence similarity and a reduced representation of base pairing information [114]. LaJolla [75], PARTS [73] and iPARTS [76] translate either torsion or pseudotorsion angles to structural alphabet letters. iPARTS2 uses a larger alphabet enhanced with sequence information [77]. FRIEs uses a fuzzy clustering method based on torsion angles, distances and H-bond information. Vectors of class probabilities are used to compute position specific scores [51]. SARA describes the backbone in terms of unit vectors. Secondary structure information is optionally incorporated by reducing the backbone to only base-paired nucleotides. Additionally, SARA uses a refinement routine to optimize the alignment with respect to rigid body superposition [52].

A different strategy starts the alignment by searching small and very similar pieces of the two structures to use as alignment seeds, which are subsequently extended. ARTS enumerates all seed matches of two successive base pairs. The structures are superimposed based on the seed match and the alignment is extended with pairs of nucleotides that are now in proximity of each other [115]. Rclick uses a similar approach, but does not consider molecular topology. Instead, local groups of close nucleotides are searched as initial alignment seeds [116]. R3D Align searches for local alignments with high structural similarity and then applies a maximum clique algorithm to merge compatible matches to a global alignment [117]. Other tools build more heavily on the hierarchical relationship between RNA secondary and tertiary structure. SETTER reduces the RNA structure to a set of secondary

structure units and aligns structures based on this representation [118]. STAR3D searches for similar helix pieces first. The tree topology of knot-free RNA secondary structures is then exploited to quickly find a maximum set of compatible matches. Finally, the unpaired loop regions are aligned [119]. RAG-3D uses a graph representation for quick searches for highly similar substructures [121]. The tools RMalign [111] and RNA-align [120] were developed recently and are inspired by the program TM-align [122], which aligns protein structures. These methods try to optimize the alignment with respect to a length independent scoring function. An ensemble of different methods is used to create initial alignments which are refined by iterative rigid body superposition.

In this work, two new tools called CVRRY[2] and ALFONS[3] are introduced. Both implement the dynamic programming alignment approach and a superposition refinement procedure. The largest difference to previous approaches is the use of novel fragment descriptors. ALFONS applies the structural alphabet described in chapter 3 and a simple identity score. CVRRY is based on a set of descriptors previously described [123]. The backbone is represented as a series of overlapping tetrahedra, each spanning seven nucleotides. Five distances are stored along with a scalar reflecting chirality (Fig. 4.2). The fragment description and the specific values of alignment parameters are the only differences between ALFONS and CVRRY. In addition to the geometric description, both tools apply a reduced string representation of base pair information. Alignment parameters are determined using a simplex optimization procedure on a large set of RNA chains. The final step of the two programs is a refinement with respect to rigid body superposition. A fast heuristic is used, which deploys two different algorithms for the superposition of two chains and the alignment from superimposed structures.

To evaluate CVRRY and ALFONS, their performance with respect to maximizing structural overlap is compared with three state of the art tools on three benchmark datasets. SARA, Rclick and STAR3D were chosen for comparison to cover a range of fundamentally different approaches and because either the tool (SARA, STAR3D) or the benchmark data (Rclick) were available. STAR3D is especially interesting because it was reported to be significantly faster than many other available tools [119]. The speed of CVRRY and ALFONS is compared to that of SARA and STAR3D and shown to be substantially faster.

[2]Chiral Volume Rna Ressemblance Yielder
[3]ALphabet For Ordinary Nucleic acid Structures

4.2. Methods

4.2.1. Datasets

Sets of RNA chains were assembled for three different purposes. All chain files were obtained from the PDB [12]. A set of chain pieces was assembled and divided into training and test data for the optimization of alignment parameters. Different benchmark sets found in the literature were used to evaluate the alignment methods and compare them to existing tools. A list of structure pairs was used for a time benchmark. The datasets used for parameterisation and evaluation are described subsequently while the time benchmark set is described separately in section 4.2.7.

Training and test data for parameter optimization

Training and test set for the optimization of alignment parameters (section 4.2.5) were assembled based on the following considerations. A large dataset comprising many different chain pairs will help to find well performing parameters. Unfortunately, the optimization will slow down and eventually become infeasible if the dataset becomes too large. This is especially true if the chains are very long since the runtime scales quadratically with the chain length. Hence, it is sensible to restrict the choice. Chain pairs which are hard to align are definitely useful. However, pairs with no similarity at all will only add noise and will not help to find better parameters. One may also prefer medium-size rather than large chains to avoid very slow comparisons.

The assembly of training and test data for this project basically comprises three steps. Large chains from an initial chain set are cut into smaller pieces. To get an initial estimate of similarity, all possible pairs are aligned multiple times with varying arbitrary parameters. Subsequently, promising chain pairs with at least some similarity are sampled.

Chains from the non-redundant list by Leontis and Zirbel [50] were used. All chains that can also be found in one of the evaluation sets were removed from the list. The base pair annotation was computed for the remaining chains (section 4.2.2). All chains shorter than 50 nucleotides were removed from the dataset. All chains longer than 350 nucleotides were cut into pieces with a maximum size of 350 nucleotides. Gaps in the chain due to missing

nucleotides were ignored. This is actually wanted, because a factor for scaling the scores of fragments close to the margins is optimized in the procedure.

Pairwise CVRRY alignments were used to search for promising candidates. Each pair in the dataset was aligned once for every parameter combination listed in Table A.5. Q-scores (Eq. 4.15) were computed for all alignments. For each chain pair we chose the best Q-score achieved with any parameter combination. All pairs with $Q < 0.6$ were removed from the list. The rest were grouped by the Q-score into bins of size 0.1 from 0.6 to 1.0. 5000 pairs were sampled without replacement from each of the four groups. Subsequently, every group was shuffled and then split 80 % to 20 % for training and test set respectively. Merging the groups finally makes 16 000 pairs in the training set and 4000 pairs in the test set.

Evaluation set

Three established benchmark sets from the literature were used for the evaluation. They have been also used by Nguyen et al. for the evaluation of the program Rclick [116]. The lists of the chain pairs were downloaded from the Rclick server [124]. The NR95-HR comprises a group of high resolution structures found to be non-redundant with respect to sequence (less than 95 % identity) [52]. All structures have a resolution higher than 4 Å and a length between 20 and 320 nucleotides. The second set is the FSCOR set and was originally distributed on the SARA web server [103]. The chains are between 11 and 2774 nucleotides long. The third set contains chains of ribosomal subunits with a length of 117 to 3308 nucleotides. It was originally assembled by Rahrig et al. [125] and will be referred to as the RIBO set. Figure 4.1 shows the length distribution of the chains in the three datasets.

4.2.2. Backbone description and similarity

The alignment algorithm uses similarity scores derived from structural features. CVRRY and ALFONS both combine a description of local backbone geometry with a coarse string representation of base pairing information. The descriptors and the respective scoring functions are described in the following.

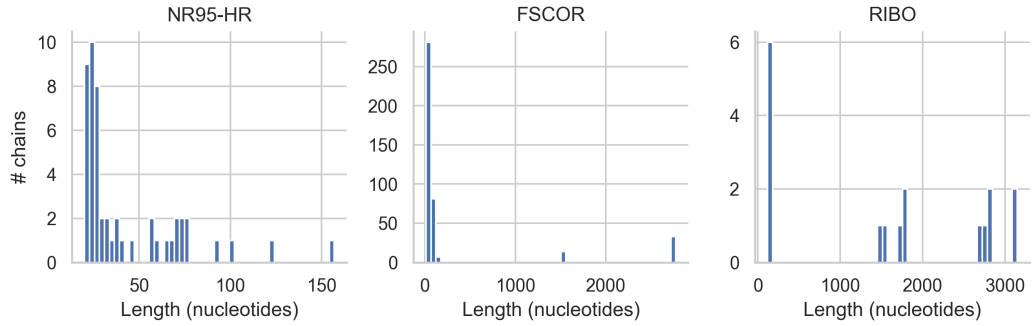

Figure 4.1.: Chain length distributions in the three evaluation sets. The bin widths of the histograms are adapted to the range of chain lengths and thus differ for the three histograms.

ALFONS score

The local backbone structure was translated into a sequence of letters using the methodology described in chapter 3. An alphabet with seven nucleotides scored best (Table 3.1) and was therefore applied here. The similarity score between two letters a and b is

$$s(a,b) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

CVRRY fragments

CVRRY fragments are described by four points in 3D space, the minimum number of points required to capture a bending direction of the backbone. The coordinates of the C4′ atoms were chosen for the description because they are at a central position of the nucleotide. Again, the idea of spaced k-mers is applied, using the same pattern as described in section 3.2.3. The C4′ atom of the nucleotides at positions i, $i+1$, $i+3$ and $i+6$ are considered. They form a tetrahedron, which is fully described by five distances and the chiral volume (Fig. 4.2). The L2-norm of an N-dimensional vector $\vec{u}$ is

$$|\vec{u}| = \sqrt{\sum_{i=1}^{N} u_i^2}, \tag{4.2}$$

where u_i is the element in $\vec{u}$ at position i.

The euclidean distance between two vectors $\vec{u}$ and $\vec{v}$ is

$$d(\vec{u}, \vec{v}) = |\vec{u} - \vec{v}|. \tag{4.3}$$

We will use the notation

$$d_{i,j} = d(\vec{r}_i, \vec{r}_j) \tag{4.4}$$

for the distance between three-dimensional coordinate vectors r_i and r_j. The chiral volume described by Braun et al. [126] is

$$V_{i,j,k,l} = (\vec{r}_i - \vec{r}_l) \cdot [(\vec{r}_j - \vec{r}_l) \times (\vec{r}_k - \vec{r}_l)]. \tag{4.5}$$

The features assembled to a vector are

$$\vec{f} = \begin{pmatrix} d_{i,j} \\ d_{i,k} \\ d_{i,l} \\ d_{j,k} \\ d_{j,l} \\ V_{i,j,k,l} \end{pmatrix} \tag{4.6}$$

with $j = i + 1$, $k = i + 2$ and $l = i + 6$ according to the previously described pattern. To balance the contribution of the features they were normalized to Z-scores [59]. Distributions were collected from the structures in the non-redundant dataset by Leontis and Zirbel [50]. Mean μ and standard deviation σ were computed for each feature. The vector of normalized features is

$$\vec{f}' = \begin{pmatrix} \vdots \\ \frac{f_i - \mu_i}{\sigma_i} \\ \vdots \end{pmatrix}, \tag{4.7}$$

where f_i is the ith element of $\vec{f}$ from equation 4.6 with corresponding mean μ_i and standard deviation σ_i. The euclidean distance $d(\vec{u}, \vec{v})$ between the normalized feature vectors $\vec{u}$ and $\vec{v}$ is used to compute the score

$$s(\vec{u}, \vec{v}) = \frac{1}{1 + d(\vec{u}, \vec{v})}. \tag{4.8}$$

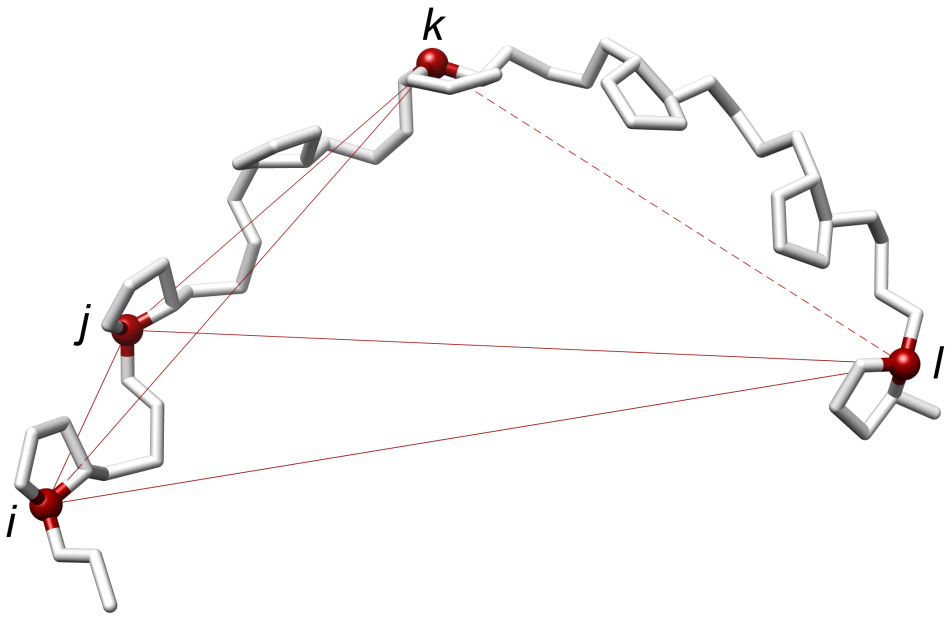

Figure 4.2.: Fragment description used in CVRRY. The chiral volume and five distances are computed for the tetrahedron formed by the highlighted C4′ atoms. Distances used in CVRRY are drawn as solid lines whereas the distance drawn as a dashed line is not explicitly part of the feature vector.

Base pair bonus

DSSR [127] is used to extract base pairing information from the coordinate file. Only canonical base pairs (AU and CG) and wobble base pairs (GU) between two nucleotides of the respective chain are considered. Interactions to other chains in the structure and different types of base pairs are ignored. The information is then encoded into a string as done in DIAL [114]. One of three letters is assigned to every nucleotide in the RNA chain based on its base pairing state. Going from the 5′ to the 3′ end of the chain, a nucleotide can be either unpaired, paired to a nucleotide in the following or in the preceding part of the chain. When comparing two such strings, the score for two letters p and q is

$$s_{basepair}(p,q) = \begin{cases} \beta, & \text{if } p = q, \\ 0, & \text{otherwise,} \end{cases} \tag{4.9}$$

where β is the size of the rewarded bonus.

4.2.3. Alignments

The computation of the alignments basically comprises three steps (Algorithm 4). The fragment descriptors and base pair strings are computed first. In both methods, the local backbone geometry is described in terms of overlapping fragments with a length of seven nucleotides. Therefore, descriptors are computed starting at each nucleotide that is followed by a continuous stretch of at least six nucleotides. This is done separately for both chains R_1 and R_2. Hence, the fragment sets F_1 and F_2 and the base pair strings B_1 and B_2 can be precomputed for a dataset. This saves much time when performing a search or an all against all comparison on a large dataset. The next step is to fill a score matrix. An individual score is assigned to each combination of nucleotide positions between the two chains. For two chains of length m and n we thus get an $m \times n$ score matrix S. The details are described later on in this section. Finally, the alignment A with maximum score is computed using the Gotoh variant [36] of the Needleman-Wunsch algorithm [34]. Penalties g_{open} and g_{extend} are used for opening and extending gaps.

Algorithm 4 Alignment routine

procedure AlignRnaChains(R_1, R_2, g_{open}, g_{extend}, λ, β)
  $m \leftarrow getChainLength(R_1)$
  $n \leftarrow getChainLength(R_2)$
  $F_1 \leftarrow computeFragmentDescriptors(R_1)$
  $F_2 \leftarrow computeFragmentDescriptors(R_2)$
  $B_1 \leftarrow getBasepairString(R_1)$
  $B_2 \leftarrow getBasepairString(R_2)$
  $S \leftarrow computeScoreMatrix(F_1, F_2, B_1, B_2, m, n, \lambda, \beta)$
  $A \leftarrow needlemanWunsch(S, g_{open}, g_{extend})$
  return A
end procedure

Algorithm 5 describes the computation of the score matrix. Two $m \times n$ matrices, the score matrix S and an auxiliary matrix C are initialized to zero at all positions. Subsequently, the scores are computed in three stages. The scores according to the backbone descriptors are added to S first. Afterwards, the score corresponding to pairs covered by fewer fragment comparisons are scaled. Finally, the similarity bonus of the base pair strings is added.

Algorithm 5 Compute the score matrix

```
1: function ComputeScoreMatrix(F_1, F_2, B_1, B_2, m, n, λ, β)
2:     l ← getFragmentLength()
3:     S ← 0_{m,n}
4:     C ← 0_{m,n}
5:     for f_1 in F_1 do
6:         i ← fragmentPosition(f_1)
7:         for f_2 in F_2 do
8:             j ← fragmentPosition(f_2)
9:             s ← calcScore(f_1, f_2)
10:            for k ← 0 to l − 1 do
11:                S_{i+k,j+k} ← S_{i+k,j+k} + s
12:                C_{i+k,j+k} ← C_{i+k,j+k} + 1
13:            end for
14:        end for
15:    end for
16:    for i ← 1 to m do
17:        for j ← 1 to n do
18:            S_{i,j} ← S_{i,j}(1 + λ (l − C_{i,j}) / C_{i,j})
19:        end for
20:    end for
21:    for i ← 1 to m do
22:        for j ← 1 to n do
23:            S_{i,j} ← S_{i,j} + s_{basepair}(B_{1,i}, B_{2,j}, β)
24:        end for
25:    end for
26:    return S
27: end function
```

To score the backbone geometry similarity, all combinations of fragments from both chains are compared. This is the only part where CVRRY and ALFONS differ. The fragments f_i and f_j are described by feature vectors (equation 4.7) in CVRRY and with alphabet letters in ALFONS. The *calcScore* function computes the similarity score given in equation 4.8 for CVRRY or equation 4.1 for ALFONS. Since all fragments span a backbone stretch of $l = 7$ nucleotides, the score is added to all seven positions along the corresponding diagonal of S. Most positions in S will therefore be the sum of the contribution of seven fragment comparisons. The chain positions close to the ends of the chain are not covered by all seven fragments. Furthermore, the backbone can be discontinuous because of missing nucleotides in the structures. Positions close to these chain breaks are also covered by fewer fragments. To account for this, a scaling procedure is implemented. Along with the computation of the scores, the number of fragments covering every position in S is computed and stored in D. Afterwards, each position $S_{i,j}$ is scaled according to the coverage $D_{i,j}$ and a factor λ. Finally, the bonus due to the base pair information strings given in equation 4.9 is added to each position of S.

4.2.4. Alignment evaluation

The following measures were applied to assess alignments.

fDME

Distance matrices can be used to compare two aligned structures without superimposing them. For a given alignment, let a and b denote the substructures comprising the C4' atoms of all nucleotides from one chain which are aligned to a nucleotide in the respective other chain. Distance matrices D^a and D^b are computed on a and b respectively. $D^a_{i,j}$ holds the distance between the coordinates of position i and j in a. The distance matrix error (DME) [128] is

$$DME = \left(\frac{2}{N_{matches}(N_{matches}-1)} \sum_{i=1}^{N_{matches}} \sum_{j=i+1}^{N_{matches}} (D^a_{i,j} - D^b_{i,j})^2\right)^{\frac{1}{2}}. \qquad (4.10)$$

However, this measure is prone to outliers. Parts of the molecules which are different or poorly aligned tend to mask similarities quickly. Furthermore,

a flexible stretch in a molecule may allow two units to arrange differently. The similarity of two molecules with such a hinge-like flexibility may be easily missed using the DME.

This is addressed by the fraction distance matrix error (fDME) [129]. In brief, it captures the share of distances for which the DME falls below a threshold ϵ. The set of residuals summed in equation 4.10 can be written as

$$E = \{(D^a_{i,j} - D^b_{i,j})^2 | i, j \in \mathbb{N} \wedge 1 \leq i, j \leq N_{matches}\}. \tag{4.11}$$

The sorted version of the list is

$$E' = [E_k | E_k \in E \wedge (k = 0 \vee E_k \geq E_{k-1})]. \tag{4.12}$$

The largest number of distances that fits under the threshold ϵ is

$$N_\epsilon = \max(\{N | \frac{1}{N} \sum_{k=1}^{N} E'_k \leq \epsilon^2\}). \tag{4.13}$$

We can now compute

$$fDME = \frac{2N_\epsilon}{N_{matches}(N_{matches} - 1)}. \tag{4.14}$$

In this work ϵ was set to 5 Å.

Q-score

The Q-score [130] weights the fDME by the coverage of the alignment. With $N_{matches}$ non-gap positions in the alignment for two sequences of length m and n we have

$$Q = fDME \frac{N_{matches}}{\min(m, n)}. \tag{4.15}$$

PSI

The Percentage of Structural Identity (PSI) describes the amount of closely superimposed nucleotides in an alignment [52]. Unlike RMSD and fDME it can not be deterministically retrieved from the alignment. Instead, it also depends on a heuristic to find a good superposition [131]. Given an alignment and two superimposed chains we can count the number of

matched nucleotides N_{sup} where the C4′ atoms are within 4 Å distance to each other. For two chains with lengths m and n we get

$$PSI = \frac{N_{sup}}{\min(m,n)}. \tag{4.16}$$

The heuristic to search the best superposition used in CVRRY and ALFONS is described in section 4.2.6.

4.2.5. Parameter optimization

There are four parameters to choose when computing an alignment as described in algorithm 4. These are the gap penalties g_{open} and g_{extend}, the margin factor λ and the base pair bonus β. The Nelder-Mead simplex algorithm [132] was used to optimize parameters as previously [129, 133]. All chain pairs from the training set are aligned repeatedly in the course of an optimization. The Q-score (Eq. 4.15) was used to assess alignments because it rewards both high coverage and structure similarity. However, not all differences are useful improvements. The score difference between two bad alignments is not meaningful and should be considered as noise. To reduce noise, the Q-score is wrapped in a logistic function. Incorporating the constraint $g_{open} \geq g_{extend}$ required by the alignment algorithm, we get the cost function

$$c = \begin{cases} 1 - (1 + e^{-k(Q-Q_0)})^{-1}, & \text{if } g_{open} \geq g_{extend}, \\ 1, & \text{otherwise,} \end{cases} \tag{4.17}$$

with arbitrary parameters k and Q_0 set to 14 and 0.7 respectively. Moves that cause $g_{open} < g_{extend}$ are rejected immediately without computing any alignments. Averaging over all $N_{alignments}$ chain pairs, we get

$$cost = \frac{1}{N_{alignments}} \sum_{i=1}^{N_{alignments}} c_i, \tag{4.18}$$

where c_i is the cost of pair i.

The optimizations were started multiple times with different initial parameters. Three or four initial values were chosen for each of the four parameters (Table A.6). All combinations with $g_{open} > g_{extend}$ were used. This yielded a list with 171 starting points. From each point two optimizations were started using different seeds for the random number generator.

4.2.6. Structure superpositions and refinement

A heuristic was implemented to refine the alignment and find a superposition subject to maximizing the PSI. The procedure combines the MaxSub algorithm [131] and an alignment method which is similar to the approach used in the program ProSup [134]. Algorithm 6 outlines the main steps. The input are the C4′ coordinates of the two chains R_1 and R_2 and an initial alignment A. The chains R_1 and R_2 have length m and n respectively. The alignment A is a set of pairs. Each pair comprises either two indices i and j for positions in R_1 and R_2 respectively or one index and a gap element. Every position of R_1 and R_2 is found in exactly one of the pairs. The first step is to find a good superposition of the nucleotides that are paired in A. This is done using the MaxSub algorithm. MaxSub is a heuristic to optimize the PSI. It tries to find a superposition, such that the coordinates of a maximum number of aligned pairs lie within 4 Å distance of each other. Pseudocode and a detailed explanation of MaxSub were given by Siew et al. [131]. It will be summarized briefly here. Pairs of small, continuous fragments from the two chains are used as seeds. All overlapping seeds of a given length are iterated. A seed length of four nucleotides was used in this work. Each iteration starts with the superposition of the two seed fragments. This is followed by a few iterations of superposition and subsequent selection of pairs with a distance below a threshold. The threshold decreases in each iteration and finally becomes 4 Å. The algorithm has time complexity $O(N^2)$. The result are the superimposed structures R_1' and R_2' and a set $M \subseteq A$. M contains only the pairs that are superimposed within 4 Å distance.

Algorithm 6 Superposition and alignment refinement

1: **procedure** SUPERIMPOSEANDREFINEALIGNMENT(A, R_1, R_2)
2: $M, R_1', R_2' \leftarrow MaxSub(A, R_1, R_2)$
3: $A' \leftarrow refineAlignment(M, R_1', R_2')$
4: $M'', R_1'', R_2'' \leftarrow MaxSub(A', R_1', R_2')$
5: **return** M'', R_1'', R_2''
6: **end procedure**

The removal of pairs from A that do not superimpose close enough can cause unaligned regions in the two chains. One reason for aligned pairs to be excluded from M is that they were not aligned well in A. There may be alternative nucleotide pairs that can be superimposed closely. Therefore, alignments are derived from the superimposed structures R_1' and R_2'.

The basic idea is borrowed from ProSup [134]. Unlike ProSup the method described here only realigns nucleotides which are not part of a pair in M. It also uses a different scoring function. The procedure is described in Algorithm 7. First, one has to collect the non-aligned regions between the matches in M. Let (i, j, k, l) describe a pair of backbone stretches $R_1[i \dots k]$ and $R_2[j \dots l]$. By adding two sentinel index pairs

$$M' = M \cup \{(0,0), (m+1, n+1)\} \tag{4.19}$$

one can define the set of unmatched regions

$$U = \left\{ (i,j,k,l) \left| \begin{array}{l} 1 \leq i \leq k \leq m \wedge 1 \leq j \leq l \leq n \\ \wedge (i-1, j-1), (k+1, l+1) \in M' \\ \wedge (p,q) \notin M' \forall p = i \dots k, q = j \dots l \end{array} \right. \right\}. \tag{4.20}$$

All regions $(i, j, k, l) \in U$ are aligned separately. The procedure is similar to the computation of the original alignment. A scoring matrix is filled and the Needleman-Wunsch algorithm is used to find the optimal path through that matrix. The score for two nucleotides with the coordinates of the C4′ atoms $\vec{r}_i \in R'_1$ and $\vec{r}_j \in R'_2$ is now

$$s_{3D} = \frac{1}{1 + d(\vec{r}_i, \vec{r}_j)^2} \tag{4.21}$$

where $d(\vec{r}_i, \vec{r}_j)$ is the euclidean distance (Equation 4.3). The local alignments are merged with M to a global alignment A'. The MaxSub algorithm is used once more to superimpose the structures based on A' (Algorithm 6). One can now compute the PSI directly from M'' with

$$PSI = \frac{|M''|}{\min(m, n)}. \tag{4.22}$$

4.2.7. Runtime benchmark

The runtimes of CVRRY, ALFONS, STAR3D and SARA were compared. Rclick is only available as a web server and not as standalone version and is therefore not included in the benchmark. Four chain pairs with different lengths were chosen and downloaded from the PDB (Table 4.1). CVRRY, ALFONS and STAR3D were executed with all pairs. SARA only accepts chains with a maximum length of 500 nucleotides by default. It was therefore

Algorithm 7 Alignment refinement

```
1: function alignSuperimposed(R''_1, R''_2)
2:     S ← [|R''_1| × |R''_2|] matrix
3:     for i ← 1 to |R''_1| do
4:         for j ← 1 to |R''_2| do
5:             S_{i,j} ← scoreBy3dDistance(R''_1[i], R''_2[j])
6:         end for
7:     end for
8:     A' ← needlemanWunsch(S)
9:     return A'
10: end function
11:
12: function refineAlignment(M, R'_1, R'_2)
13:     A' ← M
14:     U ← getNotAlignedStretches(M)
15:     for (i, j, k, l) ∈ U do
16:         A' ← A' ∪ alignSuperimposed(R'_1[i ... k], R'_2[j ... l])
17:     end for
18:     return A'
19: end function
```

Chains	Lengths (nucleotides)
1s72_9 & 2qbg_A	122 & 117
2a64_A & 3dhs_A	298 & 215
1fjg_A & 2aw7_A	1507 & 1530
1s72_0 & 3u5h_5	2754 & 3150

Table 4.1.: **Chain pairs used for the runtime benchmark.** Chains are named by PDB-ID and the chain identifier.

only benchmarked on the two shorter chain pairs. Every alignment was computed 30 times by each program. In CVRRY, ALFONS and STAR3D the preparation of each chain and the alignment are two separately executed steps. SARA performs both steps in one call. All programs produce files with an alignment and superimposed coordinates. The tests were done using the openSUSE Leap 15.0 operating system and an Intel Core i5-9500 processor with 3 GHz.

4.2.8. Implementation

CVRRY and ALFONS are implemented in the same framework. The core is written in the C programming language with SWIG [93] interfaces for Python and Perl scripting. An in-house implementation written in Perl was used for the simplex optimization. The GCC C-compiler version 7.4.1 [91], Perl 5.26.1 [135] and Python 3.6.5 [92] were used. Code for the Kabsch algorithm, dynamic programming alignments and fDME calculations was adopted from WURST [133].

4.2.9. Availability

ALFONS and CVRRY are available at `https://gitlab.com/nilspetersen/cvrry`.

4.3. Results

4.3.1. Parameter optimizations

The four alignment parameters gap open (g_{open}) and extend (g_{extend}) penalty, margin scaling factor (λ) and base pair bonus (β) were optimized for both CVRRY and ALFONS. For each tool 171 simplex runs were started with varying initial parameters (section 4.2.5). A test set was used to check for overfitting to the training data. Figure 4.3 shows the cost trajectories of the optimization runs, which found the best optima with respect to the performance on the training data. At each step the lowest cost found by the simplex is plotted. The optimizations took 492 and 428 steps for ALFONS and CVRRY respectively. In both cases, the largest improvement can be observed within the first 50 steps. Therein, the course of the test data follows that of the training data closely. Thereafter, the ALFONS trajectory has a stair like shape with three short periods of improvement. The values computed on the test set follow this course closely. The CVRRY trajectory looks different. The cost only improves once on both datasets simultaneously. This happens around step 120. Subsequently, changes leading to small improvements on the training data do not improve the performance on the test set. Instead, the cost slightly increases around step 210. However, this difference is less

than 0.001 while the total improvement on the test set is ~ 0.06 (arbitrary cost units).

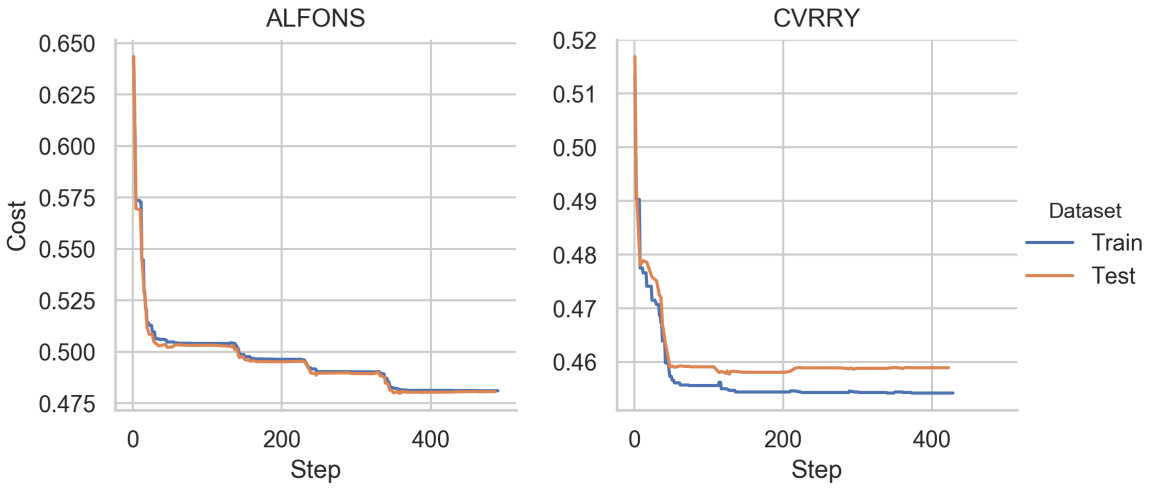

Figure 4.3.: Trajectories of simplex optimizations. The curves show the cost value of the best parameter set found by the simplex so far.

Table 4.2 lists the parameters that achieved the best results. They were used for the benchmarks of the programs in the next section. The parameters for gap costs and base pair bonus are higher for ALFONS than CVRRY. The margin factor, on the other hand, is reduced to 0 for ALFONS, while it is 0.21 for CVRRY. A large number of data points from the optimization trajectories were collected to get an idea about the relationships between parameters and alignment performance. Figure 4.4 and 4.5 show that most of the final states are found close to the best scoring parameters forming a pointy tip. Some optimizations do not reach this area and end with a significantly higher cost. Among the optimizations of ALFONS parameters there are two clearly visible groups (Fig. 4.4). The first is a group with a base pair bonus close to zero. The second group has significantly higher gap open and lower gap extend penalties compared to the best parameters. Looking at the overall distribution of data points from both CVRRY and ALFONS optimizations, one can see that the lower edge has a convex shape. Furthermore, areas on both sides of the final state clusters are explored if possible.

	Margin factor	Base pair bonus	Gap open	Gap extend
ALFONS	0.00	3.27	14.58	3.14
CVRRY	0.21	0.45	4.78	0.56

Table 4.2.: **Best alignment parameters.** The best parameters found for CVRRY and ALFONS with respect to the cost function computed on the training data.

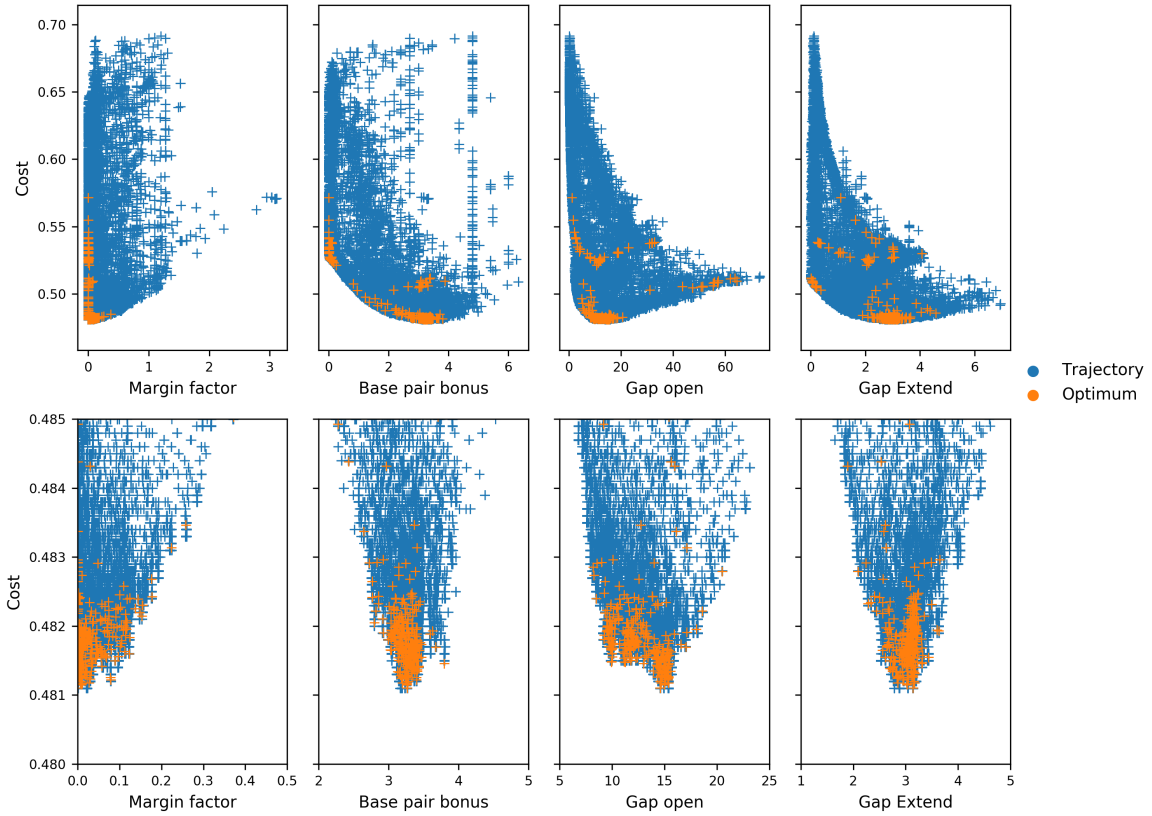

Figure 4.4.: Costs for ALFONS alignments with varying parameter combinations. Data points originate from optimization trajectories and final states. The bottom windows are a zoom onto a cluster of final scores.

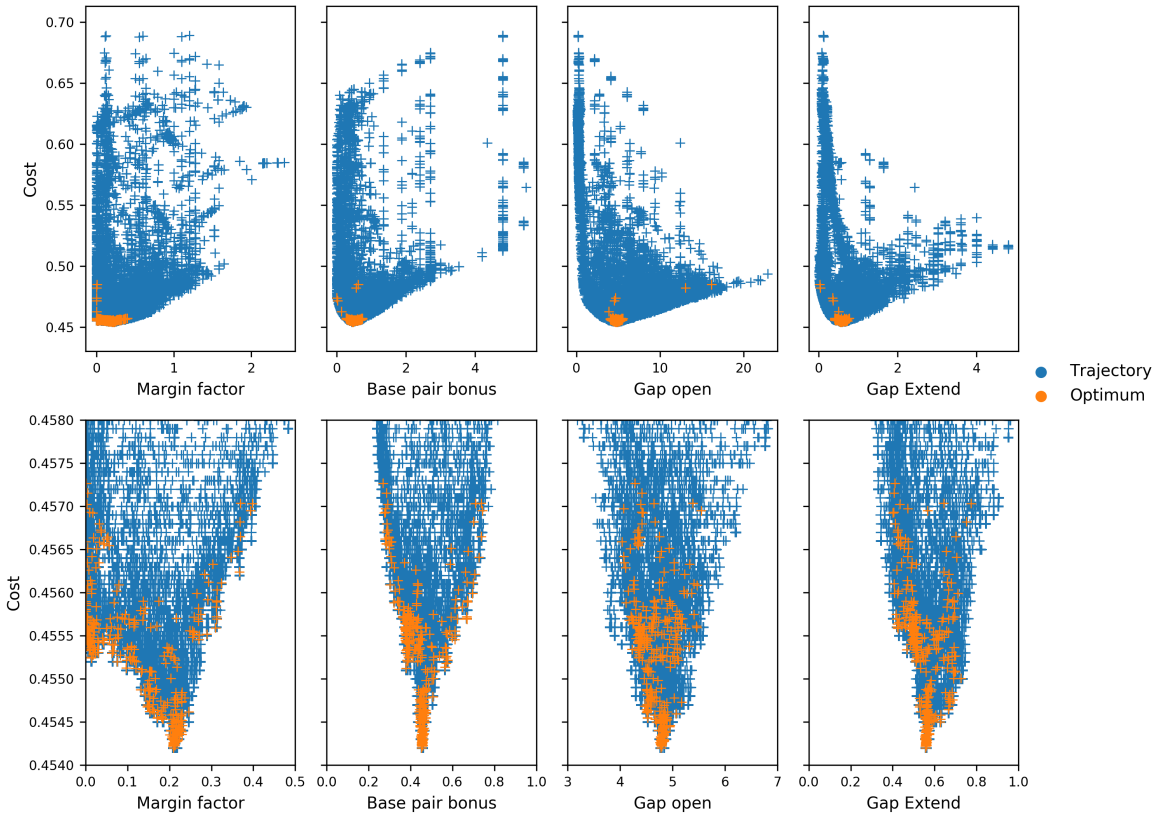

Figure 4.5.: Costs for CVRRY alignments with varying parameter combinations. Data points originate from optimization trajectories and final states. The bottom windows are a zoom onto a cluster of final scores.

Impact of base pair information

Separate optimizations with no base pair bonus were performed to assess its impact on the alignment quality. For the three other parameters the same initial combinations were used as before. Figure 4.6 shows that using the base pair information significantly reduces the cost of the final states. The distributions for optimizations with and without base pair bonus only overlap due to some outliers for which the optimization did not succeed in finding a parameter set close to the other final combinations. One can also see that significantly better scores are reached with CVRRY compared to ALFONS.

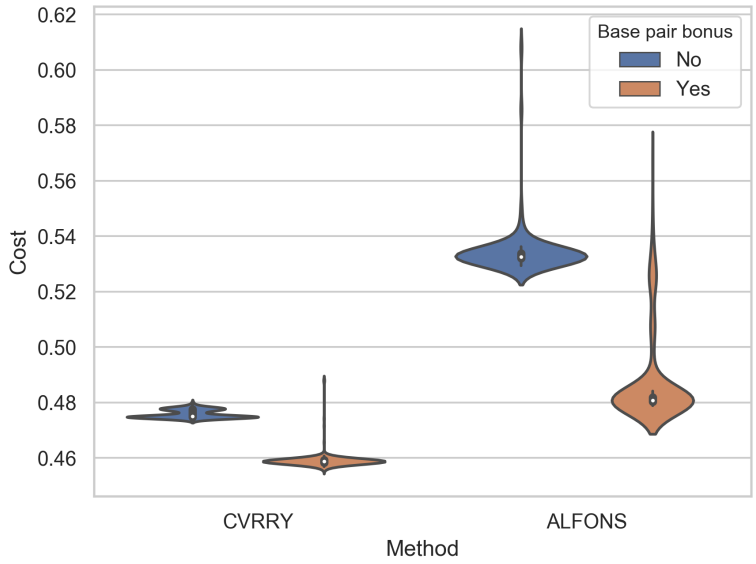

Figure 4.6.: Costs of final states for optimizations with and without base pair bonus.

4.3.2. Superposition refinement

Alignments were computed for the FSCOR dataset with and without superposition based refinement. The success is measured with the PSI. For the alignments without refinement the best superposition is simply searched using MaxSub based on the alignment. In the other case the whole refinement procedure described in section 4.2.6 is applied. Figure 4.7 shows cumulative curves of the PSI. The largest gain can be observed in the middle for PSI values around 0.6. The differences become smaller to the ends of the curve.

4.3.3. Comparison of different alignment programs

Three datasets with RNA chains were used to assess the quality of ALFONS and CVRRY alignments in comparison with state of the art tools. PSI scores were used as the performance measure. Alignments were computed using ALFONS, CVRRY, STAR3D and SARA. The scores achieved by Rclick were taken from the Rclick web server. SARA and STAR3D could not compute alignments for all cases. Both programs require at least some secondary

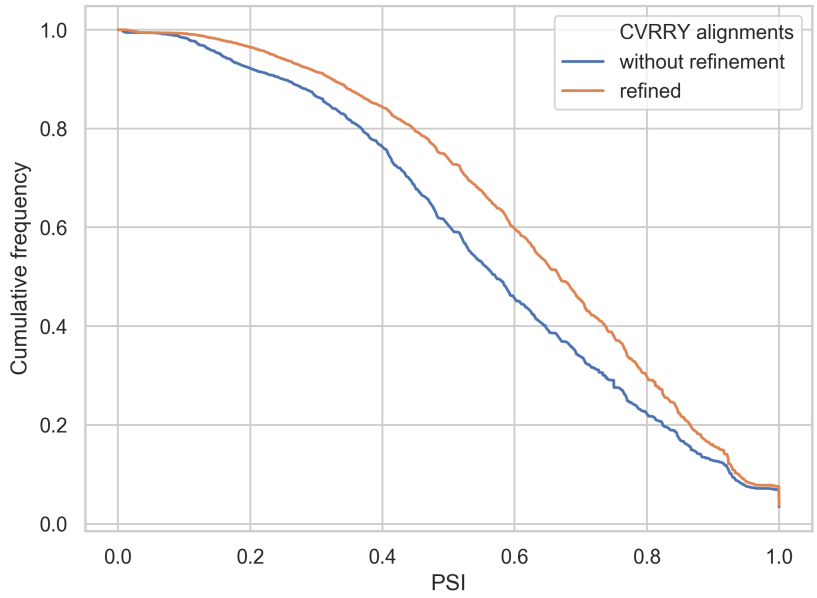

Figure 4.7.: PSI computed for CVRRY alignments on FSCOR dataset with and without alignment refinement.

structure and can not align chains without base pairs. STAR3D needs a minimum of three stacked base pairs. SARA is also limited to align chains that are not longer than 500 nucleotides. The numbers of alignments computed with every tool are listed in Table 4.3. On the NR95-HR dataset, for example, STAR3D computed alignments for less than half of the chain pairs. For the RIBO dataset SARA can only compute the 10 alignments of the shortest chain pairs due to the size limitation. For each tool, only the chain pairs where alignments were computed successfully are taken into account for the comparison against CVRRY and ALFONS.

Cumulative curves of PSI values are shown in Figure 4.8. CVRRY and ALFONS perform nearly equally good with CVRRY producing slightly better results in total.

The largest differences to ALFONS and CVRRY are visible for the Rclick alignments. Rclick outperforms the two programs significantly on the NR95-HR and FSCOR dataset. This is not true for the RIBO dataset. In the Rclick benchmark multiple chains of a complex are superimposed together. This is not yet implemented for any of the other methods and therefore chains are compared separately. Therefore, there are only 30 alignments for comparison

	# of alignments		
	FSCOR	NR95-HR	RIBO
Total	87571	1275	40
Max. 500 Nucleotides	68635	1275	10
ALFONS	87571	1275	40
CVRRY	87571	1275	40
SARA	68628	1275	10
STAR3D	78758	595	40
RCLICK	87571	1275	35

Table 4.3.: **Number of alignments computed successfully with each program.** The second line lists the number of chain pairs where both chains are not longer than 500 nucleotides.

against Rclick. CVRRY and ALFONS both fail in one case to find a good alignment where Rclick succeeds. For the other cases, ALFONS and CVRRY alignments are equally good and sometimes slightly better than Rclick alignments.

The STAR3D results are closer to the PSI scores of ALFONS and CVRRY. While the cumulative frequency is larger for STAR3D across most PSI values, the CVRRY curve is slightly higher for very large PSI values. For the alignment of the large chains in the RIBO dataset, both CVRRY and ALFONS achieve higher cumulative frequencies.

SARA curves are closest to those of CVRRY and ALFONS. The largest difference is observed for very high PSI scores on the FSCOR dataset where CVRRY and ALFONS outperform SARA. The results on the RIBO dataset only comprise the ten shortest alignments. The scores of the three programs are very similar here.

Figure 4.9 shows heat maps of PSI values for the FSCOR dataset comparing CVRRY results to those of each of the other tools. In these plots, the quality of two alignments is more similar the closer they are to the diagonal. For each of the four other programs on can see at least some cases, where CVRRY performs better, and some, where it produces worse results. The most similar results are observed for ALFONS. A large number of alignments is found on or very close to the diagonal here. For ALFONS, STAR3D and SARA the distribution of the alignments around the diagonal is close to symmetric. For the Rclick alignments one can clearly observe a shift of PSI values.

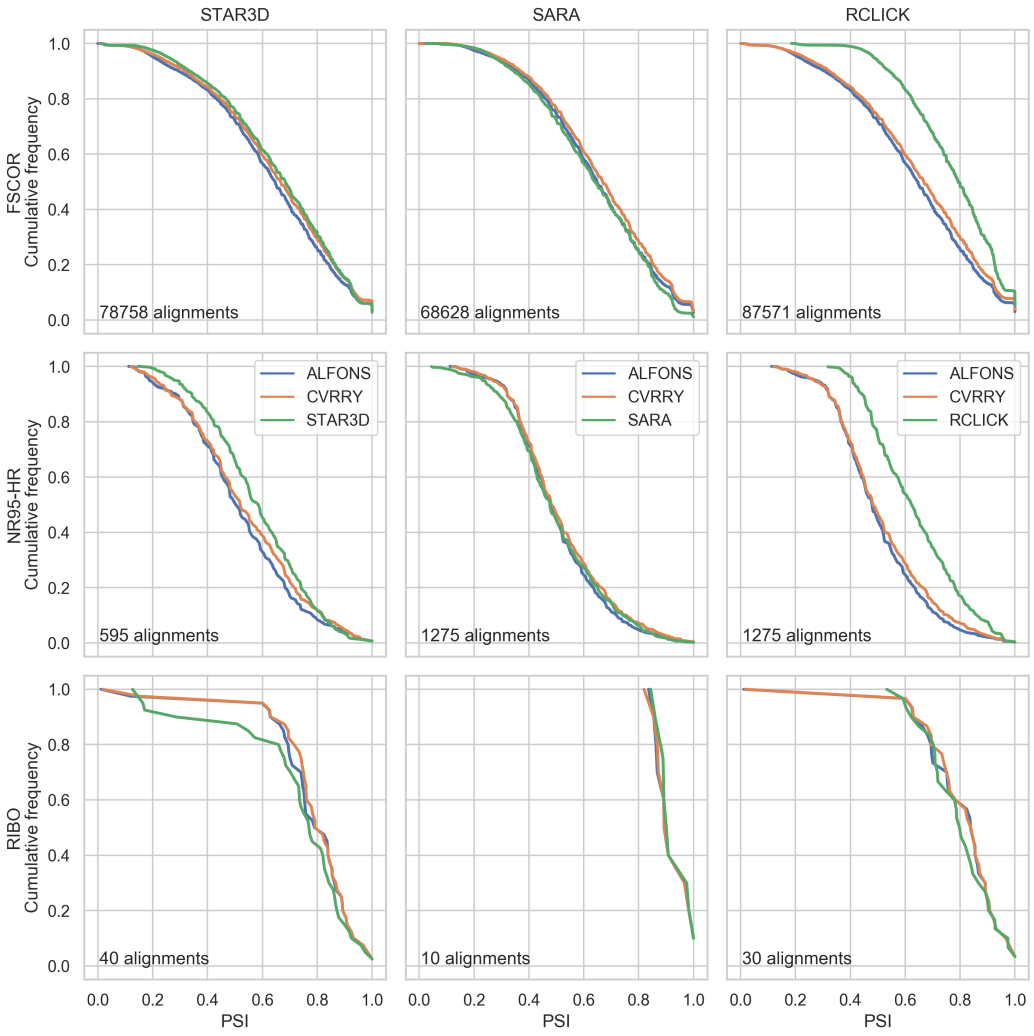

Figure 4.8.: Cumulative frequencies of PSI values for ALFONS, CVRRY, STAR3D and Rclick alignments.

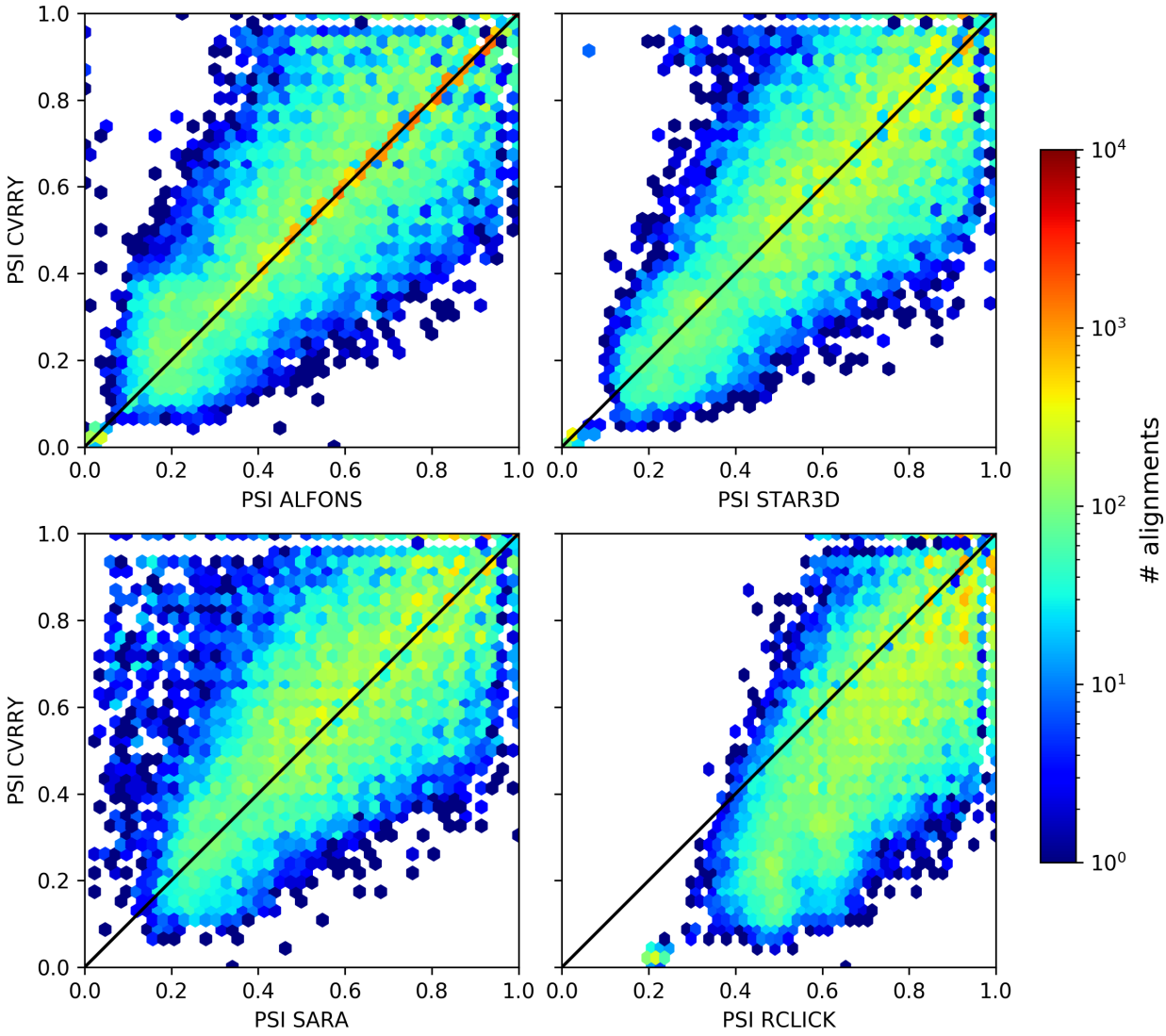

Figure 4.9.: PSI values of CVRRY alignments compared to those computed with ALFONS, STAR3D, SARA and Rclick.

This is most obvious for cases where CVRRY superimposes only very few nucleotides. The minimum PSI value for Rclick is 0.2. Most of the CVRRY alignments with a PSI less than 0.1 have a PSI larger than 0.4 when computed with Rclick.

Figure 4.10 A shows the PSI scores of CVRRY alignments of the chains in the RIBO set compared to those computed with ALFONS. The programs achieve similar PSI scores for most chain pairs. There are two exceptions. In

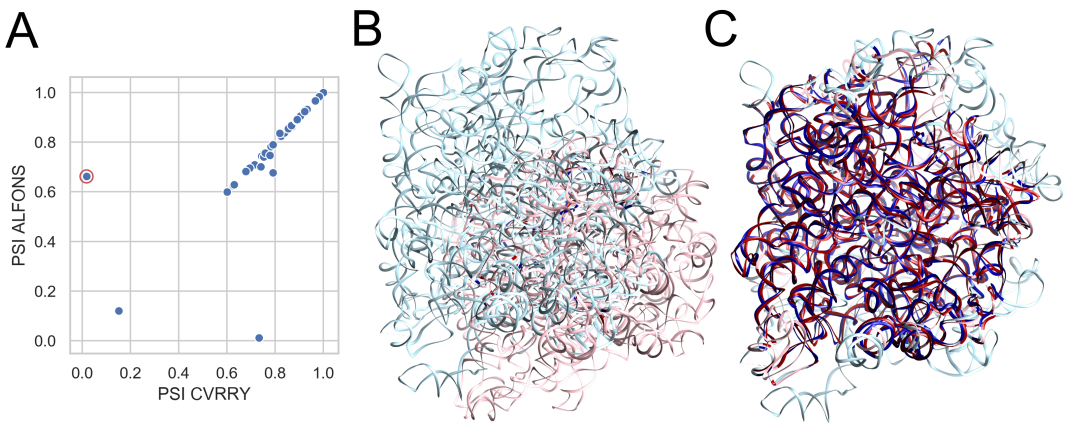

Figure 4.10.: Parameter dependency of alignment between chains 2zjr_X and 4a1b_1. (A) PSI scores of CVRRY and ALFONS on RIBO dataset with original parameters. The alignment between the two chains is highlighted. (B) CVRRY alignment with parameters from original optimization. (C) CVRRY alignment after parameters were specifically optimized for this chain pair.

one case CVRRY fails to compute a proper alignment and in the other case ALFONS fails. In both cases, the other tool achieves a score higher than 0.6. Hence, there is evidence that a much better alignment is possible. CVRRY fails aligning the two chains 2zjr_X and 4a1b_1 (Fig. 4.10 B). To check if the two chains can be aligned with different parameters, the four alignment parameters were optimized again. The parameters given in Table 4.2 were used as starting point. Only this chain pair was used as training set. The chains were not cut into pieces as done for the training set. The optimization yielded margin factor 0.28, base pair bonus 0.53, gap open penalty 1.96 and gap extend penalty 0.16. Using these parameters CVRRY achieves a PSI score of 0.78. Figure 4.10 C shows the new superposition.

4.3.4. Runtimes

Figure 4.11 depicts the results of the runtime benchmark. The average runtimes are also listed in Table A.7, A.8 and A.9. For the comparison of ALFONS, CVRRY and STAR3D the precomputations and the alignments are analyzed separately. The precomputations for ALFONS and CVRRY are faster for six of eight chains by a factor between 13 for chain 1fjg_A and 132 for chain 2qbg_A. For the chains 2a64_A and 3dhs_A STAR3D is about 1.2

and 1.6 times faster than ALFONS and CVRRY. The alignment computations are always faster for ALFONS and CVRRY compared to STAR3D. The most extreme case is the alignment of chains 1s72_9 and 2qbg_A. The chains are the shortest chains in the dataset. They are 112 and 117 nucleotides long. ALFONS and CVRRY computations require on average 0.008 and 0.007 seconds respectively, which is roughly 63 and 71 times faster than STAR3D. The longest chains in the dataset are 1s72_0 and 3u5h_5 with length 2754 and 3150 nucleotides. ALFONS needs 0.3 and CVRRY 0.4 seconds to compute the alignment. They are therefore about 5 and 4 times faster than STAR3D in this case. For the comparison against SARA the precomputations and alignment calculation are measured together. For the shorter chains 1s72_9 and 2qbg_A ALFONS and CVRRY are both about 2 times faster than SARA. Aligning the longer chains 2a64_A and 3dhs_A takes SARA roughly 8 times longer than ALFONS and CVRRY. The differences between CVRRY and ALFONS are much smaller than those to SARA and STAR3D. Looking at the alignments only one can see that CVRRY performs slightly better in one case while ALFONS performs better in the three other cases.

4.4. Discussion

The following considerations highlight the strengths, difficulties and perspectives of the new programs CVRRY and ALFONS in comparison to the three established tools SARA, STAR3D and Rclick. This should support further developments in this field and also help to choose a tool for a given application.

Figure 4.9 shows how the tools produce alignments with varying quality for the different chain pairs. Neither does CVRRY perform better nor does it perform worse on all alignments compared to any of the other tools. It is therefore obvious that none of the tools always finds the perfect alignment with respect to PSI. Thus, there is, at least theoretically, room for improvement for each of the tools.

A naive approach to improve the search for a good alignment would be to employ several programs. For a pair of chains one simply chooses the best alignment according to some measure like the PSI. Since different tools are superior for different chains pairs, there is obviously an improvement in the overall accuracy of the alignments compared to the use of any single tool. The downside is that the required computational resources are the sum of

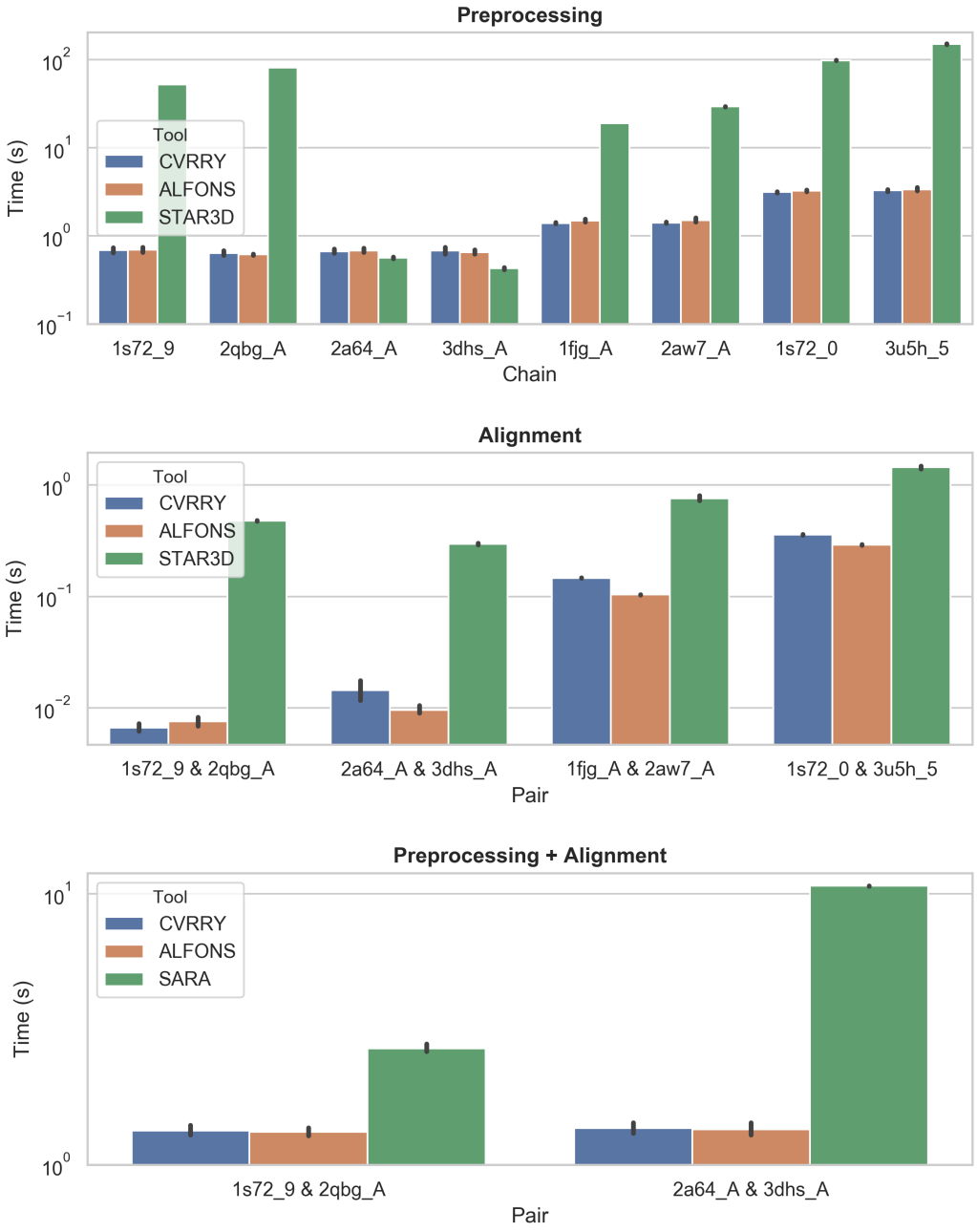

Figure 4.11.: Average times required to compute chain descriptors and alignments. Error bars represent 95 % bootstrap confidence intervals.

all programs applied. This may be suitable if one is interested in an accurate comparison of only few chain pairs.

If one specific tool is to be chosen, one should consider the purpose of the application. Sometimes an exact mapping of corresponding nucleotides is essential. Better mappings can for example improve results from homology modeling [136, 137] or the assignment of interaction sites [110]. For other applications it may be enough to distinguish similar from unsimilar chains. Examples of such cases would be the search for homologs or clustering. In this work, the quality of the nucleotide mapping was assessed with the PSI. It was chosen because it is a state of the art measure for benchmarking RNA alignments and has been widely used [52, 116, 119]. Its application is built on the assumption that related nucleotides can be superimposed closely. The mapping that produces the largest number of closely superimposed nucleotides is expected to be the most accurate alignment. This may be reasonable for most cases but there are some limitations. If there are one or more hinge movements in the molecule, only a fraction of the molecules can be superimposed. The authors of Rclick addressed this issue by computing alignments iteratively. After each alignment the superimposed parts are removed from the two chains and the remaining nucleotides are used for the next iteration. This procedure could easily be adopted by any of the other tools. The program SupeRNAlign uses an even more sophisticated approach. The method is built on top of existing alignment tools. In multiple iterations matches that can be superimposed very closely are selected from alignments produced by an external alignment tool [138]. Unfortunately, there is yet no single score to assess and compare the quality of such alignments. Distance matrix based methods such as the fDME score could be an alternative. The challenge remains to balance local and global similarity in order to produce meaningful results. Another drawback of using the PSI is the fixed threshold. The threshold of 4 Å is probably appropriate for small to medium size RNA molecules such as tRNAs. For large ribosomal subunits this threshold may be too small and exclude corresponding nucleotide pairs from the superposition. Recently, two scores were proposed as a size independent alternative to the RMSD [111, 120]. Both measures were inspired and are similar to the TM-score for protein structures [139]. They could be useful for future benchmarks and applications of the RNA structure alignment tools presented here.

From Figures 4.8 and 4.9 one can clearly see that on average Rclick alignments achieve higher PSI scores than CVRRY and ALFONS on the FSCOR dataset. This may not be too surprising. The Rclick heuristic approaches the

objective of maximizing the PSI most directly of all tools tested here. The algorithm searches the largest number of nucleotides that can be superimposed closely irrespective of the molecules topology. This is different to the other tools, which are bound to the linear order of the chains. The largest differences can be observed for chain pairs where CVRRY superimposes very few nucleotides (PSI < 0.1) while Rclick alignments have medium PSI values around 0.5. However, it is debatable whether these superpositions are helpful. Many pairs in the dataset are between unrelated pairs and superimposing a larger amount of these structures will not help to get much biological insight. Although the differences between the results from STAR3D compared to CVRRY and ALFONS are smaller than those described for Rclick, one can see a similar trend here. STAR3D reaches higher cumulative scores for low to medium PSI values on the NR95-HR and the FSCOR dataset (Fig. 4.8). These differences can be explained to a large extend with a closer look at the STAR3D algorithm. Searching for the most similar helices by superposition, the algorithm explicitly starts with finding a group of superimposable nucleotides. Given two small chains with low overall similarity and a significant proportion of helix structure, finding the best matching helix pair may well be the best alignment one can get with respect to PSI. There are probably many such cases since the distribution of chain lengths is very skewed to small values in the two datasets (Fig. 4.1). Again, it is debatable whether these improvements are helpful. The downside of relying on secondary structure is, that STAR3D fails to align structures without or with only few base pairs. This happened for more than half of the chain pairs in the NR95-HR dataset (Table 4.3). The picture changes for very similar chains with PSI scores close to one. Here, the cumulative frequency for CVRRY is slightly higher than for STAR3D. This improvement is probably due to the refinement step. Unlike in CVRRY, ALFONS and SARA, STAR3D does not make a final refinement of the global superposition. SARA is the method in the benchmark which is methodologically most similar to CVRRY and ALFONS. It also achieves the most similar overall performance with respect to cumulative PSI curves (Fig. 4.8). Unfortunately, it can not compute some alignments for the FSCOR dataset and is computationally too inefficient to compute alignments between most of the structures in the RIBO set in an acceptable time. Furthermore, CVRRY and ALFONS perform better for very similar structure pairs with PSI values close to one in the FSCOR dataset. When choosing between SARA and CVRRY one is probably better off using CVRRY. The most encouraging results for the application of ALFONS and CVRRY are the alignments of the large homologous chains in the RIBO set. PSI values are often similar or

higher compared to those achieved by STAR3D or Rclick (Fig. 4.8). Unfortunately, there are also cases where CVRRY and ALFONS fail to produce a useful alignment (Fig. 4.10).

The risk to misalign backbone stretches and the sensitivity of the alignment to the parameters are consequences of using limited local information. Figure 3.7 in chapter 3 shows how the largest fraction of fragments have a helix-like shape and are therefore assigned to the same letter. This leads to different choices with similar scores. Nucleotides within long helices, for example, share a very similar local surrounding. Therefore, it becomes difficult to correctly assign residues that are in long helices of different size. Additionally, large gaps increase the number of alternative choices for an alignment and thus make the problem more difficult. Unfortunately, the gaps are not only insertions and deletions from the evolutionary history of the molecules but due to missing backbone pieces that could not be modeled from the experimental data. Furthermore, gaps differ very much in their length. There are insertions and deletions of single nucleotides, but there are also pairs of large ribosome chains that differ by the existence of whole helices. The superposition shown in Figure 4.10 C comprises all these different kinds of gaps. The alignment of the two depicted chains was found to be very sensitive to the used parameters. A large number of the nucleotides in the cores of the chains are superimposed closely. At the edges there are large unmatched pieces from both chains. Some of them are from structural differences while others are due to missing pieces. There are also several small insertions throughout the structure. Hence, this example demonstrates the difficulties of the dynamic programming algorithm quite well. Base pairing information was incorporated to add some information about non-local contacts. The simple linear representation was used to stay within the quadratic runtime of the dynamic programming algorithm and avoid computationally more expensive tree comparisons. Figure 4.6 shows that this simple extension leads to a significant improvement. A possible improvement in the future could be to incorporate more different interactions such as different non-canonical base pairs, base triples or interaction between bases and backbone.

Simplex optimization was used to search for sensible parameters. This method is well suited for functions with a convex shape, and it does not require the computation of gradients. But there is no guarantee to find the global optimum if the landscape of the target function is rugged. One may end in local minima and not even get close to a good parameter choice. In this work we did not need to find the global optimum but wished to find

parameters that perform as good as possible. The target function is obviously not perfectly smooth since some optimizations end in local optima with significantly higher cost and different parameters compared to optimizations with a better outcome (Fig. 4.4 and 4.5). However, most optimizations end in a valley of the target function and have very similar final parameters. This is especially remarkable since the optimizations started from a wide range of parameters. Furthermore, the optima for ALFONS and CVRRY are very different (Table 4.2) but are well reached using the same set of initial conditions. The ability to converge to this regions in parameter space from a wide range of starting points shows that the overall shape of the target function is very close to convex and that the minima found are likely to be close to a global optimum. The observed cost values computed on the test set during the optimizations followed those for the training set for most of the improvement (Fig. 4.3). We thus conclude that the final parameters should perform well when applied to new structures. One should also keep in mind that this parameter set performs well for a large number of chain pairs but is not the best in every single case. An example is the chain pair depicted in Figure 4.10.

The refinement algorithm was shown to significantly improve the superposition with respect to PSI (Fig. 4.7). The bottleneck here is the limitation of the performance by the initial alignment. If the largest superimposed piece found by the MaxSub procedure is a good fit more nucleotides which are now superposed closely are added to the alignment. In order to enhance the performance one could try to use multiple parameter sets and choose the best alignment in the end. The cost of doing this would be a longer runtime. A drawback of the superposition refinement is that it is limited to structures without hinge movements, which is not the case for the pure backbone alignment. As pointed out previously, an iterative approach of superimposing rigid subunits could work here.

Time efficiency is the major strength of ALFONS and CVRRY. The two programs are significantly faster than SARA and STAR3D. Runtime is critical when many structure pairs are aligned. Chain specific features can be precomputed and used multiple times for applications like database searches or all against all comparisons. In these cases, the alignment speed is much more important than the preprocessing. Therefore, the runtimes of preprocessing and alignment were compared separately for CVRRY, ALFONS and STAR3D. ALFONS and CVRRY are clearly faster than STAR3D when computing alignments. They are also faster in most of the precomputations. However, there are two outliers. Interestingly, the precomputations for the

two chains 1s72_9 and 2qbg_A take STAR3D much longer than those for 2a64_A and 3dhs_A, even though the latter are much longer. This is probably due to the calculation of base pairs. 1s72_9 and 2qbg_A are chains from structure files with multiple long RNA chains. STAR3D passes those files to MC-Annotate [140,141] which searches for base pairs in all chains. This takes significantly longer for more and longer chains. The files of the chains 2a64_A and 3dhs_A contain only one chain and the computation is therefore much faster. CVRRY and ALFONS do not have this problem because the respective chain is extracted from the file before it is passed to DSSR for base pair annotation. A comparison to Rclick was not done because the program is only available as a web server. Therefore, only some coarse speculations are made here. The clique searching algorithm deployed by Rclick is computationally more complex than the quadratic runtime algorithms used in CVRRY and ALFONS. The measured time reported by Nguyen et al. [116] for small RNA chains is substantially higher than those measured in this study. Furthermore, the authors write that they had to reduce the number of potential seed matches by only considering nucleotides with identical bases when comparing large structures in order to make the calculation feasible. Unfortunately, this means that large structures with very low sequence identity can not be aligned.

4.5. Conclusion

This chapter introduced ALFONS and CVRRY, two new tools for RNA 3D structure alignment. The greatest strength of these tools is that they are significantly faster than state of the art tools while producing alignments with similar quality measured in terms of structural overlap (PSI) for pairs of structurally similar chains. A site for future work is the sensitivity to parameters which sometimes leads to poor alignments in cases where a different parameter choice clearly succeeds. ALFONS and CVRRY are thus recommended to use in cases where computational resources are limited while rare mis-alignments do not have lethal consequences.

5 Chapter 5. Alignment-free RNA structure comparison

5.1. Introduction

In the previous chapter two new time efficient tools for structure alignments with quadratic time complexity were introduced ($O(mn)$ for chains with length m and n). However, for some applications there is a demand for even faster comparisons. In cases where the exact residue or nucleotide mapping is not important, alignment-free heuristics may help to quickly get an estimate of the overall similarity of two molecules. Quick comparison may, for example, enable fast database scans for related structures or the clustering of large numbers of macromolecular structures. In both scenarios, the score from the alignment-free comparison can be used directly or applied as a filter to select candidate pairs that are subsequently aligned using a more expensive tool.

In a biological context, alignment-free methods were first developed and are now well established for the comparison of biological sequences [142]. They are most relevant when dealing with large genomic sequences [142], but are also used to build guide trees for multiple alignments of protein sequences [143, 144]. The list of methods and available tools is long [142]. Many tools rely on the sequence composition of words with a fixed length, also called k-mers [142]. The average common substring method extends this idea by utilizing enhanced suffix arrays to account for substring matches with different lengths. A distance is defined based on the average length of the longest common substrings shared by the two sequences [145].

Approaches were also made to implement alignment-free comparisons of macromolecular structures. As for alignment methods, most methods were designed to compare protein structures [146–152]. Few methods were developed for quick comparison of RNA structures. Most of them compare RNA molecules on the secondary structure level [153–155]. Alignment-free comparison of 3D RNA structures can be conducted using the program FRASS. Therein, the structures are represented as Gauss integrals which are compared to get a distance estimate. The comparison based on this representation is fast and the speed is independent of the size of the molecules ($O(1)$). The computation of the Gauss integrals, on the other hand, has cubic ($O(N^3)$) time complexity. Therefore, the Gauss integrals have to be precomputed and stored for a set of structures in order to get time efficient processes [156].

Herein, new methods for fast RNA 3D structure comparison are presented. They are based on substring matches between strings of the structural alphabet described in section 3. Two simple attempts were implemented first and are presented as a baseline for the evaluation of the more sophisticated approaches. The first one is the computation of the longest common substring (LCS). The length of the LCS is used to rank structure pairs. The second method is closely related to the average common substring measure for genome comparison. It sums the length of all longest substring matches and is referred to as the sum of longest common substrings (LCS_SUM). The tools URSULA[1] and FREEDOLIN[2] are based on LCS and LCS_SUM respectively. Unlike their predecessors, these methods take the uneven distribution of the different alphabet letters found in experimentally determined structure models (Fig. 3.7) into account. Instead of just using the match length, the substring matches are weighted by an estimate of significance. To achieve fast computations, an enhanced suffix array is utilized and extended with two additional arrays for each of the two input strings. Using these data structures, the comparison requires only $O(m+n)$ time for two chains with m and n nucleotides length.

A benchmark is conducted by simulating a database scenario based on annotations from the Rfam database [157]. The four new tools and the program FRASS are compared with respect to their ability to find structures of homologous molecules. Additionally, the speed of FREEDOLIN, URSULA and FRASS is compared by measuring the time required to perform a

[1] Ultrafast Rna Search Using Letters from a structural Alphabet
[2] (alignment) FREE Detection Of Large Interesting Nucleic acid structures

database search. The runtimes of FREEDOLIN and URSULA are also shown in comparison to ALFONS, the alignment tool described in the previous chapter.

5.2. Methods

5.2.1. RNA structure alphabet strings, suffix arrays and the longest common substring

RNA structures were translated to strings as described in section 3.2.3. The six letter alphabet presented in section 3.3.3 was used. As mentioned earlier (section 3.2.1), the structure models of RNA chains found in the PDB are often discontinuous with some nucleotides missing. To prevent string matches across these chain breaks and across the molecule ends, sentinel characters were inserted at the respective positions. The strings were concatenated and subsequently a generalized suffix array (GSA), a longest common prefix (LCP) array and the longest common substring (LCS) were computed as described in section 3.2.4.

5.2.2. Longest substring matches

Let $S = (s_1, \ldots, s_m)$ and $T = (t_1, \ldots, t_n)$ be two strings with length m and n respectively. For every position i in S, let $lcs(S, i)$ be the size of the longest substring $S[i \ldots i + lcs(S, i) - 1]$ that exactly matches a substring $T[j \ldots j + lcs_{a,i} - 1]$ at some position j in T. Vice versa, there is a longest exact substring match of size $lcs(T, j)$ starting at position j in T and some position i in S. Note that $lcs(S, i)$ and $lcs(T, j)$ also depend on T and S respectively. However, the respective other string is omitted from the function parameters in the notation used here to achieve a better readability and avoid confusion between $lcs(S, i)$ and $lcs(T, j)$. In the following, the computation of two arrays $LCS_S = (lcs(S, 1), \ldots, lcs(S, m))$ and $LCS_T = (lcs(T, 1), \ldots, lcs(T, n))$ is described.

Pseudocode for the procedure is given in Algorithm 8. The computation gets the GSA GSA and the LCP array LCP as input. GSA contains the indices of all suffixes of the concatenated string $U = (s_1, \ldots, s_m, t_1, \ldots, t_n)$ in lexical order. U, GSA and LCP have the same length l. Let $U_j = U[j \ldots k]$

be the substring of U that starts at position j and ends with the nearest following sentinel at position k. We can consider GSA as a sorted list of sentinel-terminated substrings of U. Since the sentinel characters are unique, they set a limit to the length of the LCP between any successive entries in GSA. $LCP[i]$ holds the size of the LCP shared by substrings $U_{GSA[i]}$ and $U_{GSA[i-1]}$. The length of the LCP of two substrings $U_{GSA[i]}$ and $U_{GSA[j]}$ with $i < j$ is

$$lcp(U_{GSA[i]}, U_{GSA[j]}) = \min(LCP[i+1 \dots j]). \tag{5.1}$$

In the following, we have to distinguish between substrings from sequences S and those from T. $U_{GSA[i]}$ is a substring of S, if $GSA[i] <= m$, and a substring of T, otherwise. We will now search for the longest exact substring match for each $1 \leq i \leq l$ between $U_{GSA[i]}$ and the other chain. Let

$$lcs_{GSA}(i) = \begin{cases} lcs(S, GSA[i]), & \text{if } GSA[i] \leq m, \\ lcs(T, GSA[i] - m), & \text{else.} \end{cases} \tag{5.2}$$

$lcs(S, GSA[i])$ is equal to the LCP of either the nearest preceding ($j < i$) or the next succeeding ($i < j$) entry at position j in GSA for which $U_{GSA[j]}$ is a substring of T. This is due to the lexical ordering of GSA. Using equation 5.1, the size of the LCP to the closest preceding entry from the respective other string is computed for each position $1 \leq i \leq l$ as

$$lcs^{-}(i) = \begin{cases} 0, & \text{if } i = 1, \\ LCP[i], & \text{if } GSA[i] \leq m < GSA[i-1] \\ & \text{ or } GSA[i-1] \leq m < GSA[i], \\ \min(LCP[i], lcs^{-}(i-1)), & \text{otherwise.} \end{cases} \tag{5.3}$$

For the next successor from the respective other string, we have

$$lcs^{+}(i) = \begin{cases} 0, & \text{if } i = m + n, \\ LCP[i+1], & \text{if } GSA[i] \leq m < GSA[i+1] \\ & \text{ or } GSA[i+1] \leq m < GSA[i], \\ \min(LCP[i+1], lcs^{+}(i+1)), & \text{otherwise.} \end{cases} \tag{5.4}$$

Now, we can compute

$$lcs_{GSA}(i) = \max(lcs^{-}(i), lcs^{+}(i)). \tag{5.5}$$

The first three loops in Algorithm 8 compute equations 5.3, 5.4 and 5.5 for $i = 1 \dots l$ using dynamic programming. The fourth loop sorts the match

lengths according to the order in which the suffixes occur in S and T. Each of the four loops takes l steps. Since $l = m + n$, the overall time complexity is $O(m + n)$.

5.2.3. Sum of longest common substrings

Using the notation of section 5.2.2, the sum of longest common substrings for a string S with respect to another string T is

$$LCS\_SUM(S) = \sum_{i=1}^{m} lcs(S, i). \tag{5.6}$$

Hence, we can compute LCS_S and LCS_T as described previously and calculate the sum of each array to get the values for S with respect to T and vice versa.

5.2.4. FREEDOLIN

The FREEDOLIN score is, like LCS_SUM (Eq. 5.6), a sum of contributions from matches of string S found in T. Instead of taking a sum over their lengths, the matches are weighted by an estimate of significance.

For a character a we say $p(a)$ is the probability of a. $p(a)$ is estimated using the measured frequencies of the letters depicted in Figure 3.7. If we make the simplifying assumption that the probabilities to observe two specific characters at two distinct positions in a string $V = (v_1, \ldots, v_l)$ of length l are independent from each other,

$$p(V) = p(v_1, \ldots, v_l) = p(v_1) \ldots p(v_l) = \prod_{i=1}^{l} p(v_i) \tag{5.7}$$

becomes the probability of observing V.

Given V and another string $T = (t_1, \ldots, t_n)$ with length n and $n > l$, we say that V is a substring of T, if V equals $T[j..j + l - 1]$ for some j. Let $N(T, l)$ be the number of sentinel-free substrings of length l in T and, therefore, the number of substrings that could potentially match V. We now estimate the probability that V is a substring of T as

$$p(V \text{ is a substring of } T) = N(T, l) \prod_{i=1}^{l} p(v_i). \tag{5.8}$$

Algorithm 8 Compute longest matches between two strings

```
1: function longestMatchEachPosition(GSA, LCP, m)
2:     l ← getSize(GSA)
                                          ▷ forward traversal of the GSA
3:     LCS⁻ ← newArray(l)
4:     LCS⁻[0] ← 0
5:     for i ← 2 to l do
6:         if sameString(GSA[i], GSA[i − 1]) then
7:             LCS⁻[i] ← min(LCP[i], LCS⁻[i − 1])
8:         else
9:             LCS⁻[i] ← LCP[i]
10:        end if
11:    end for
                                          ▷ backward traversal of the GSA
12:    LCS⁺ ← newArray(l)
13:    LCS⁺[l] ← 0
14:    for i ← l − 1 to 1 do
15:        if sameString(GSA[i], GSA[i + 1]) then
16:            LCS⁺[i] ← min(LCP[i + 1], LCS⁺[i + 1])
17:        else
18:            LCS⁺[i] ← LCP[i + 1]
19:        end if
20:    end for
                                          ▷ merge forward and backward values
21:    LCS_GSA ← newArray(l)
22:    for i ← 1 to l do
23:        LCS_GSA[i] ← max(LCS⁻[i], LCS⁺[i])
24:    end for
                                          ▷ suffix array to string order
25:    LCS_str ← newArray(l_GSA)
26:    for i ← 1 to l_GSA do
27:        LCS_str[GSA[i]] ← LCS_GSA[i]
28:    end for
29:    LCS_S ← LCS_str[1 . . . m]
30:    LCS_T ← LCS_str[m + 1 . . . l]
31:    return LCS_S, LCS_T
32: end function
```

The negative logarithm

$$-\log p(V \text{ is a substring of } T) = -\log(N(T,l)) - \sum_{i=1}^{l} \log p(v_i) \tag{5.9}$$

is used to weight substring matches in FREEDOLIN. Let us assume we have computed LCS_S for string S with length m as described in section 5.2.2. We therefore know length $lcs(S,i)$ of the longest substring match for each position i with $1 \leq i \leq m$. For every position i in S we define the weight

$$w(S,T,i) = -\log(N(T, lcs(S,i))) - \sum_{j=i}^{i+lcs(S,i)-1} \log p(s_i). \tag{5.10}$$

Taking the sum over all matches in LCS_S, we get

$$s_{FREEDOLIN}(S,T) = \sum_{i=1}^{m} w(S,T,i). \tag{5.11}$$

It may be helpful to point out that $s_{FREEDOLIN}(S,T) \neq s_{FREEDOLIN}(T,S)$.

In order to compute Equation 5.11 in linear time $O(m)$, two additional arrays are constructed. The first array is $N_T = (\log N(T,1), \ldots, \log N(T,n))$ and holds $\log N(T,l)$ in T for each length l with $1 \leq l \leq n$. The calculation for counting the substrings in $O(n)$ time is described in Algorithm 9. The second array is used to compute the sum on the right hand side of Equation 5.10 in constant time. Let us define the cumulative sum of log-letter-frequencies for any i with $0 \leq i \leq m$ as

$$c(S,i) = \begin{cases} \sum_{j=1}^{i} \log p(s_j), & \text{if } i > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.12}$$

We can tabulate these values to get an array $C = (c(S,0), \ldots, c(S,m))$ and then compute the sum in Equation 5.10 in constant time using

$$\log P(s_i, \ldots, s_j) = \sum_{k=i}^{j} \log P(s_i) = c(S,j) - c(S,i-1). \tag{5.13}$$

Algorithm 9 Count sentinel-free substrings

```
1: function countSubstrings(S)
2:     m ← getLength(S)
3:     Counts[1...m] ← 0
4:     k ← 0
5:     for i ← 1 to l do
6:         if isSentinel(S[i]) then
7:             k ← 0
8:         else
9:             k ← k + 1
10:            Counts[k] ← Counts[k] + 1
11:        end if
12:    end for
13:    for i ← m − 1 to 1 do
14:        Counts[i] ← Counts[i] + Counts[i + 1]
15:    end for
16:    N_S ← newArray(m)
17:    for i ← 1 to m do
18:        N_S[i] ← log(Counts[i])
19:    end for
20:    return N_S
21: end function
```

Algorithm 10 describes a procedure to compute $s_{FREEDOLIN}(S, T)$ and $s_{FREEDOLIN}(T, S)$ simultaneously. Line 2 to 5 describe the computation of the sequence specific arrays N_S, N_T, C_S and C_T. N_S and C_S are independent from T and N_T and C_T are independent from S. Hence, these arrays can be computed beforehand and stored in order to be re-used. In line 6 S and T are concatenated to compute GSA and LCP, which are then used to compute $s_{FREEDOLIN}(S, T)$ and $s_{FREEDOLIN}(T, S)$.

The overall time complexity of the algorithm is $O(m + n)$. The computations of the structural alphabet string, the array of substring counts and the cumulative log-frequencies each require $O(m)$ and $O(n)$ steps for S and T respectively. Computing the generalized suffix array and the LCP array on the concatenated string requires $O(m + n)$ steps. The joint construction of the arrays LCS_S and LCS_T as described in section 5.2.2 also requires $O(m + n)$. With the previously described data structures available, the FREEDOLIN score can be computed in $O(m)$ and $O(n)$ steps for the two chains respectively.

Algorithm 10 FREEDOLIN comparison of RNA structure alphabet strings

1: **function** FREEDOLIN(S, T)
2: $N_S \leftarrow countSubstrings(S)$
3: $N_T \leftarrow countSubstrings(T)$
4: $C_S \leftarrow cumulativeLogFrequency(S)$
5: $C_T \leftarrow cumulativeLogFrequency(T)$
6: $U \leftarrow concatenate(S, T)$
7: $GSA \leftarrow inducedSort(U)$
8: $LCP \leftarrow kasaiLCP(GSA)$
9: $m \leftarrow getLength(S)$
10: $LCS_S, LCS_T \leftarrow longestMatchEachPosition(GSA, LCP, m)$
11: $score_S \leftarrow computeFreedolinScore(LCS_S, C_S, N_T)$
12: $score_T \leftarrow computeFreedolinScore(LCS_T, C_T, N_S)$
13: **return** $score_S, score_T$
14: **end function**

5.2.5. URSULA

The URSULA score uses the same weights for matches as the FREEDOLIN score (Eq 5.10). The score for S given T is equal to the largest weight among all substrings of S found in T. Analogous to equation 5.11 we can write

$$s_{URSULA}(S, T) = \max_{i=1...m} w(S, T, i). \tag{5.14}$$

The routine implemented here is very similar to that of FREEDOLIN, which is described in algorithm 10. The only difference is found in line 11 and 12, where the maxima in LCS_S and LCS_t are identified instead of taking the sums.

Implementation remark

The routine described in algorithm 10 computes LCS_S and LCS_T as described in algorithm 8. For further optimization, one may consider that the substring match with the largest weight will always be between suffixes, which are adjacent in GSA. Thus, one could save a few computing steps. The four loops in algorithm 8 and the loop required to find the maximum in LCS_S and LCS_T can be reduced to one. However, the overall runtime complexity stays in $O(m + n)$. In this work, the routines from FREEDOLIN were reused for the implementation of URSULA for convenience.

5.2.6. Evaluation

Dataset

The classification of ncRNAs provided by the Rfam database version 14.1 [157] was utilized for evaluation. We follow the nomenclature of RNA families used in Rfam. Coordinate files were downloaded from the PDB [12] for all Rfam entries for which a structure model is available. Chains that are only represented by a coarse grained chain of P-atoms and chains that are shorter than seven nucleotides were removed from the set, because no strings can be computed for these cases. Some chains are assigned to more than one family. These chains were also removed from the dataset. Eleven chains were removed from the dataset because FRASS failed to process the structure files. Eventually, the dataset included 4909 chains.

Performance assessment

The performance of the four tools was compared in a database search scenario. Every chain from each family with at least two members was compared to all other chains in the dataset. A search is considered successful if a large amount of chains from the same family (positives) score higher than most chains that belong to another family (negatives). This can be visualized using a receiver operating characteristic (ROC) curve [158]. Figure 5.1 shows some example curves. One has to keep in mind here, that a curve considers only the comparison of the query chain with the database and not, as it is often done, all pairwise scores. The area under the curve value (AUC) [158] was used to get the performance of a search as a single number.

Runtime benchmark

All runtime measurements were computed on the machine described in section 4.2.7. As before, every measurement was repeated 30 times. Runtimes were measured in two different scenarios to compare FREEDOLIN and URSULA to the alignment tools from chapter 4 and to FRASS. To compare FREEDOLIN and URSULA to the alignment programs, they were executed using the chain pairs listed in Table 4.1. The preparation of the chain features and the comparison were executed separately as done previously for most of the alignment tools. A different runtime benchmark was performed to

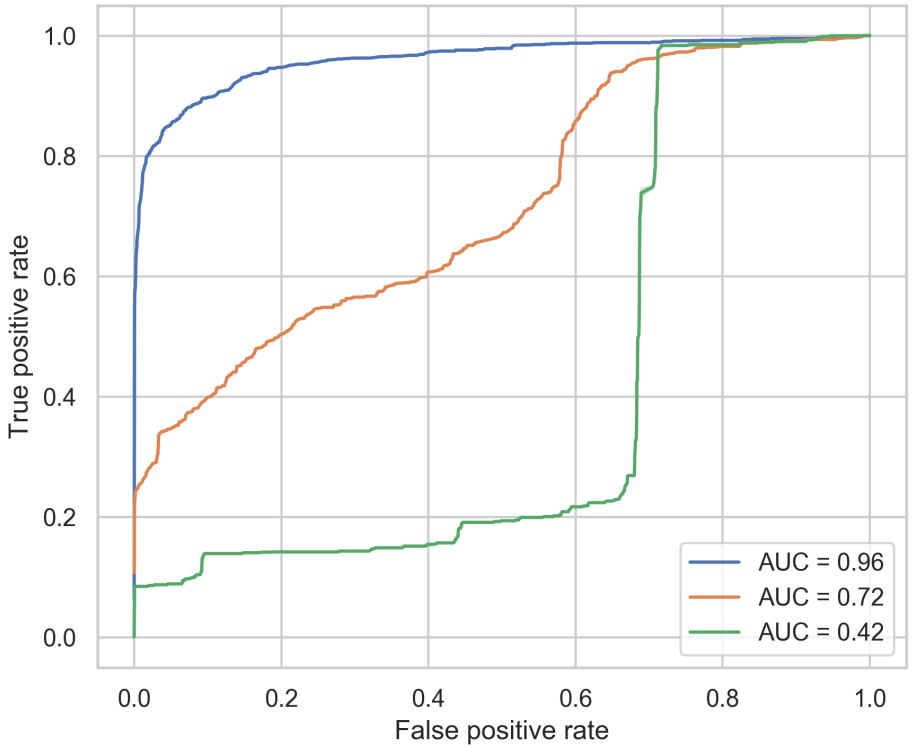

Figure 5.1.: Example ROC curves of Rfam database searches with varying AUC values.

compare the time efficiency of FREEDOLIN, URSULA and FRASS. The time required to search the Rfam dataset was measured. The first chain of each pair listed in Table 4.1 was chosen as a query. Hence, there are four different queries for chains with a length of 122 to 2754 nucleotides. The FRASS software package contains tools to set up a database and to compare a query chain against the database. To set up the database, the Gaussian integrals for a given list of structures are computed and stored. When a query chain is compared against the database, the Gaussian integrals are computed for that chain and subsequently compared to all entries in the database. In order to get a fair comparison, a similar procedure was implemented for FREEDOLIN and URSULA. A database with string descriptors for all chains from the Rfam set was prepared. Each database search includes the computation of the string description for the query chain and the comparison to all chains in the database.

5.2.7. Implementation

The alignment-free string comparison methods were implemented into the same framework described in section 3.2.8. The same tools were used. The ROC curves and AUC values were computed using the function from scikit-learn [95].

5.2.8. Availability

FREEDOLIN and URSULA are available at `https://gitlab.com/nilspetersen/cvrry`.

5.3. Results

5.3.1. Database scan

Database searches were conducted on the Rfam dataset and the results were assessed computing AUC values as described in section 5.2.6. Figure 5.2 and 5.3 show the distributions of AUC values for the different methods grouped by the Rfam family of the queries. One can see that FREEDOLIN and URSULA perform significantly better than their simple counterparts LCS_SUM and LCS for most of the families. When comparing FREEDOLIN and FRASS, multiple cases where one of the method performs significantly better than the other are easily found for both directions. The same is true for the comparison of URSULA and FRASS.

An average score is required to assess and compare the overall performance of the tools. However, the families differ a lot in size. The largest family contains 1292 structures while the smallest families for which queries were run have only two members. A simple average AUC over all query structures would be dominated by the largest families. Therefore, the mean is computed separately for every family first. One gets 68 mean values for 68 families. Taking the average of these 68 values yields the overall score. The results for the five tools are shown in Figure 5.4. Note that the shown 95 % bootstrap confidence intervals are also computed on the 68 mean values. The average AUC of FREEDOLIN, URSULA and FRASS are close to each other with overlapping confidence intervals. FREEDOLIN scores marginally higher than URSULA which performs slightly better than FRASS. The three tools all perform significantly better than LCS_SUM and LCS.

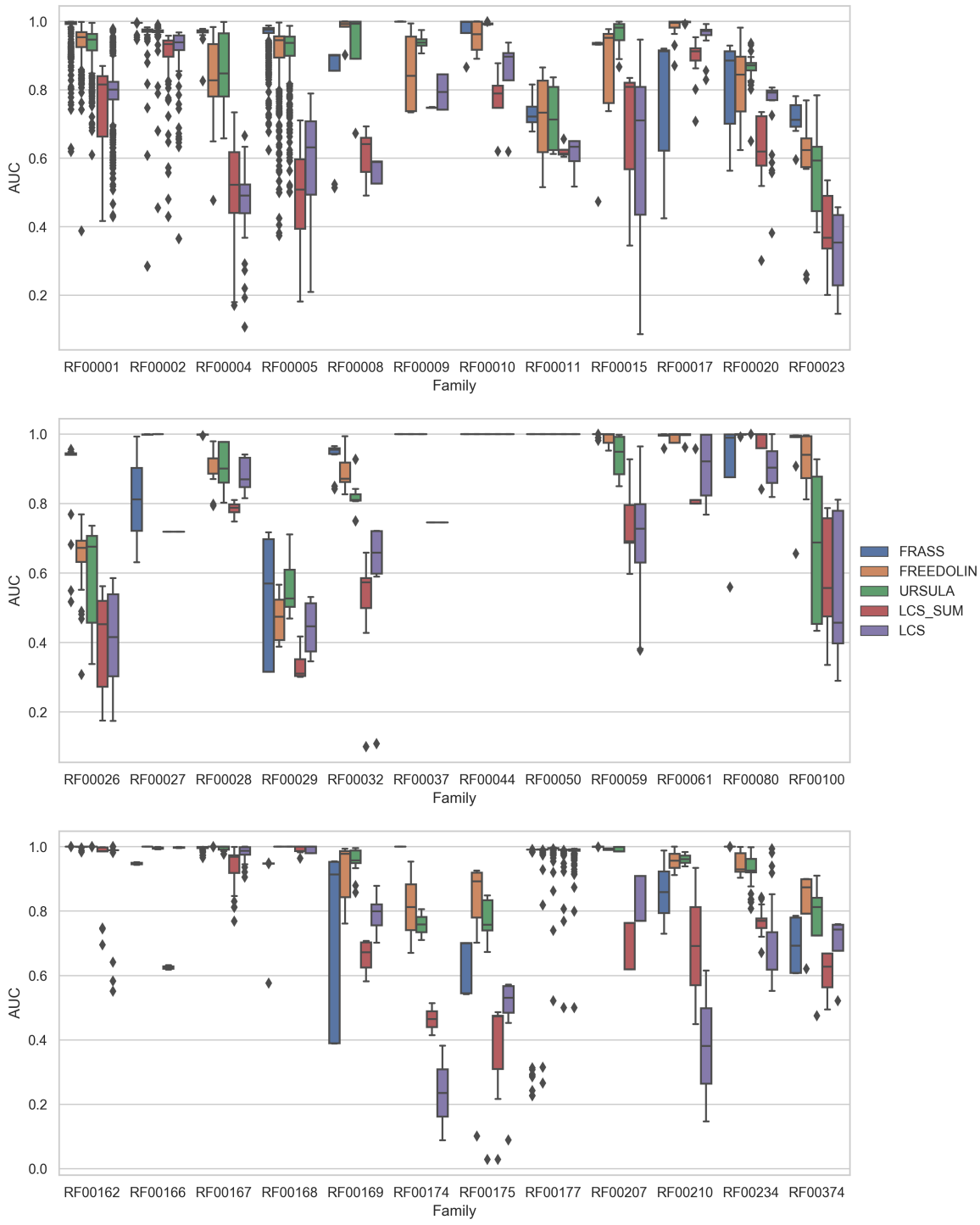

Figure 5.2.: AUC values for comparisons of single RNA chains against the Rfam dataset. Boxes show medians, upper and lower quartiles. Whiskers stretch out to 1.5 interquartile ranges. Outliers are drawn as circles. Figure 5.3 shows the results for the other 32 families.

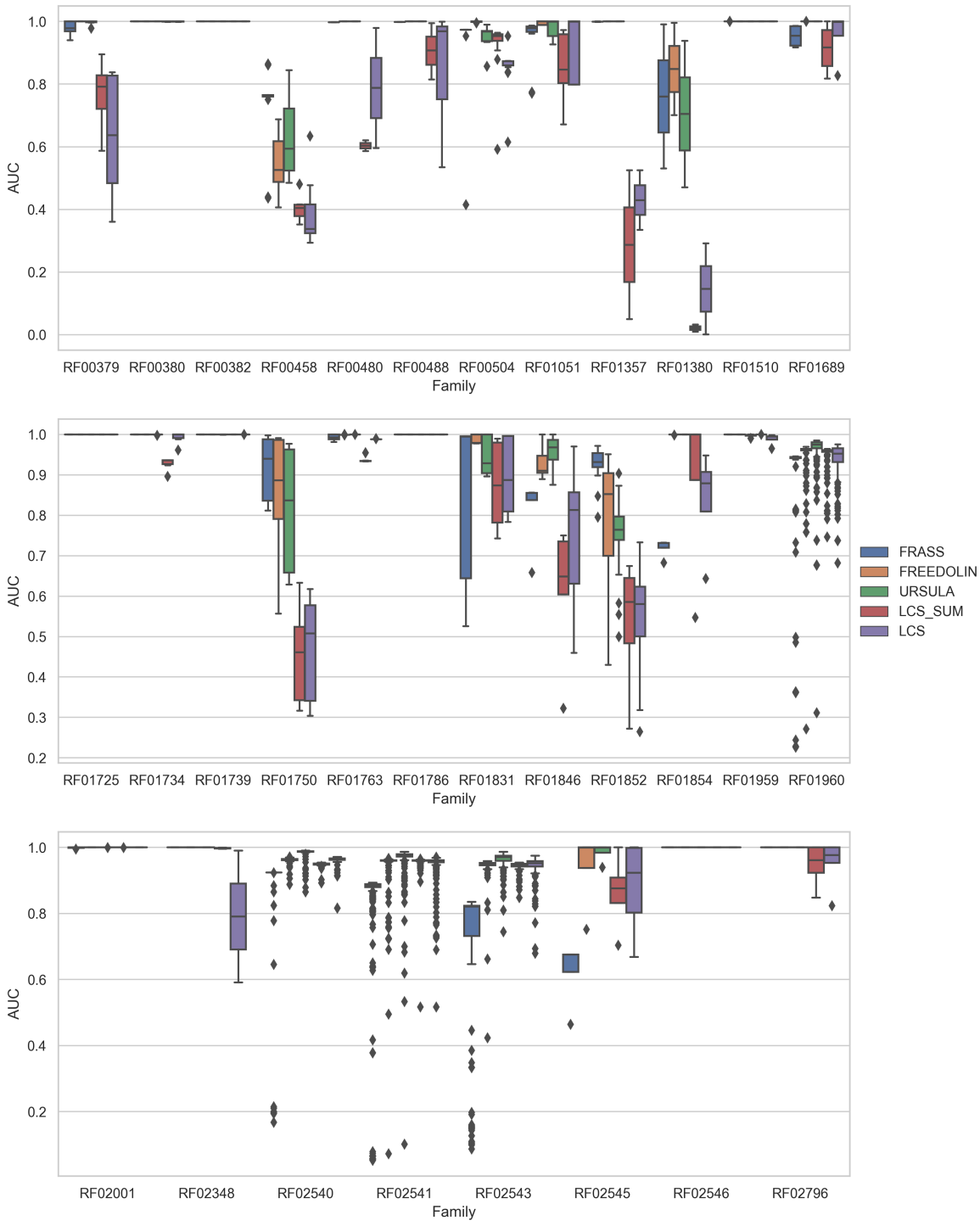

Figure 5.3.: AUC values for comparisons of single RNA chains against the Rfam dataset. Boxes and whiskers as in Figure 5.2, which shows the results for the other 36 families.

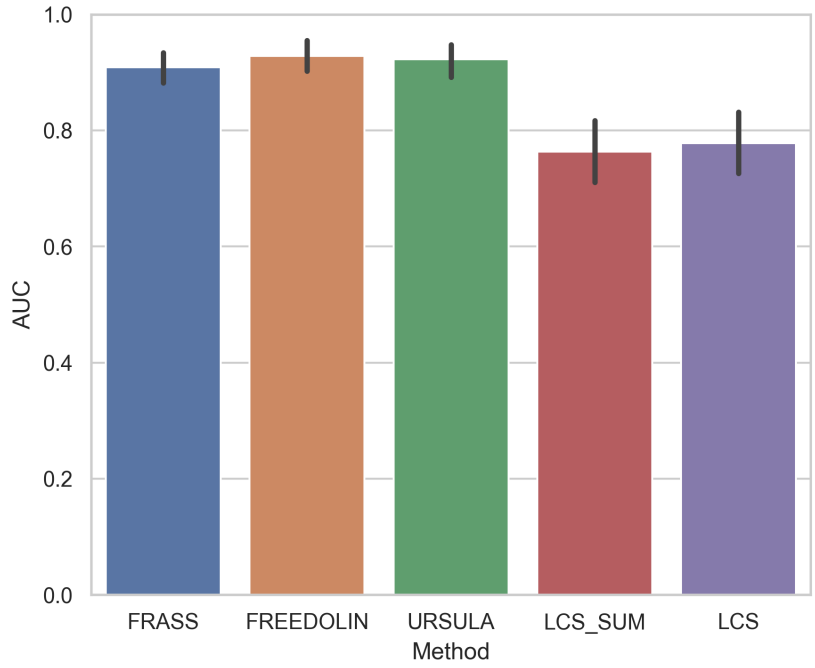

Figure 5.4.: AUC scores of database searches averaged over all families. Error bars represent 95 % bootstrap confidence intervals computed on per family averages.

5.3.2. Runtimes

Runtimes were measured following the description in section 5.2.6. The results are depicted in Figure 5.5. Figure 5.5 A and B show the runtimes of the alignment tool ALFONS and the alignment-free tools FREEDOLIN and URSULA. The average runtimes are also listed in Table A.7 and A.8. Figure 5.5 A shows the time required for the preprocessing of the chains. Since the preprocessing step is the same for FREEDOLIN and URSULA, it is only depicted once. The preprocessing step conducted by the alignment-free tools is significantly faster in all cases compared to the procedure of ALFONS. Figure 5.5 B depicts the average times required for a comparison of two chains. The runtimes of FREEDOLIN and URSULA are very similar with a difference of less than 10 %. FREEDOLIN and URSULA are always faster than ALFONS. The runtimes of all programs increase with the length of the chains, but the runtime of ALFONS increases to a substantially larger extent compared to that of FREEDOLIN and URSULA. The comparison of the shortest two chains 1s72_9 and 2qpg_A takes ALFONS 1.1 times as long as FREEDOLIN. This factor grows to 29.5 for the comparison of the longest two chains 1s72_0 and 3u5h_5.

Figure 5.5 C shows the duration of database scans performed with FRASS, FREEDOLIN and URSULA. Again, the speed of FREEDOLIN and URSULA is very similar. FRASS is faster than FREEDOLIN and URSULA for the searches with the shorter query chains. For the shortest query chain 1s72_9 FRASS is only marginally faster and requires 0.97 times the time FREEDOLIN requires. The difference is a bit larger for chain 2a64_A where the runtime of FRASS is 0.8 times the runtime of FREEDOLIN. FRASS becomes substantially slower than the string based methods when longer chains are used as queries. For the query chains 1s72_9 and 2a64_A scanning the dataset takes FRASS 6.7 and 38.3 times as long as FREEDOLIN.

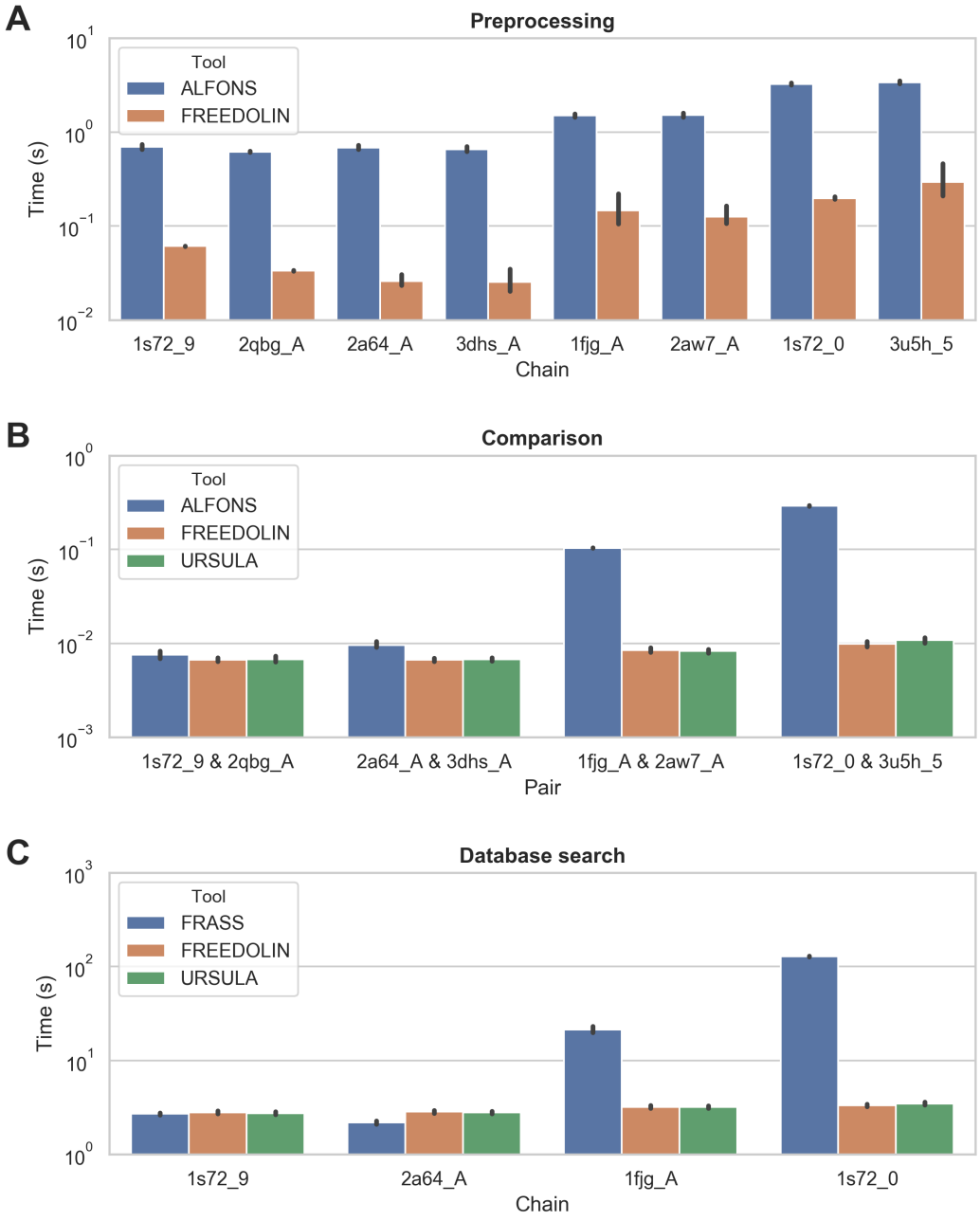

Figure 5.5.: Average times required to (A) compute structural alphabet strings, (B) compare two chains and (C) search a database. The preprocessing step is the same for FREEDOLIN and URSULA and therefore only depicted once. Error bars represent 95 % bootstrap confidence intervals.

5.4. Discussion

There are two obvious objectives for the design of an alignment-free structure comparison method. The method should be fast and it should provide useful information about the similarity of two structures. The comparison to the program FRASS was herein used to evaluate how far these goals are reached by two new tools FREEDOLIN and URSULA. Furthermore, the comparison to the more naive attempts LCS and LCS_SUM is used to demonstrate the benefit of the substring match weighting used in FREEDOLIN and URSULA. The speed of the alignment-free tools is also compared to that of the alignment program ALFONS in order to get an idea about how much the alignment-free comparisons could help to speed up time consuming processes.

Let us first compare FREEDOLIN and URSULA to LCS and LCS_SUM. Methodologically, FREEDOLIN differs from LCS_SUM only by the weights assigned to substring matches. The same is true for URSULA and LCS. LCS and LCS_SUM weight the substring matches simply by their length as done by the methods for genomic data [145] that inspired the work presented here. FREEDOLIN and URSULA, on the other hand, assign weights to substring matches based on a simple estimate of statistical significance. Figures 5.2 to 5.4 show that FREEDOLIN and URSULA clearly outperform LCS and LCS_SUM. One can conclude that the significance estimate developed here brings a substantial improvement in the reliability of similarity detected between similar structures.

The weight function used in FREEDOLIN and URSULA was described by Equation 5.10. Substring matches that are estimated to be unlikely found by chance are weighted higher than those which are assumed to be more likely. There are two contributions. The first is the letter composition of the substring. Letters that are rarely seen in general contribute more weight than the more abundant ones. The second contribution is due to the size of the target string in which the given substring was found. The chance to find a given substring by chance increases with the size of the structure it is searched in. Hence, the significance and thereby the weight assigned to the match is smaller for larger target structures. However, the estimate is just a rough approximation. It is based on the simplifying assumption that the probabilities to find certain letters at different positions are independent from each other (Eq. 5.7). It is easy to see that this is not true though. Typical RNA strands comprise helix and loop regions. Hence, a letter that represents

a typical helix shape is likely to be in the proximity of other helix letters. Likewise, certain loop shapes will be translated to specific sequences of letters and therefore some combinations of neighboring letters will occur more frequently than others. The accuracy of the estimate could be improved using a model with conditional probabilities. A little bit more sophisticated than the current approach but still easy to implement would be the use of a Markov chain [159]. Therein, the probability of a character at a certain position in the string depends on the character at the preceding position.

Another part of this work with room for improvement is the combination of the weighted longest common substrings to a single score. Taking the sum, as done by the FREEDOLIN score, is a rather ad-hoc solution. It was used because it is a simple way to account for multiple stretches that are common between the backbones of the two compared structures. However, the score can not be interpreted in terms of probability since the matches listed by the suffix array are certainly not independent of each other. An interesting development for the future would be a statistically rigorous estimate of the joint probability of all substring matches.

To compare FREEDOLIN, URSULA and FRASS one can have a look at the average AUC values computed for the database scans and observe that FREEDOLIN and URSULA achieve marginally higher scores than FRASS (Fig. 5.4). More insight is gained observing the performances of the families separately (Fig. 5.2 and 5.3). Here, one can see that the string based methods and FRASS are superior for different families. This means that the winner on the overall performance is likely to change with the composition of the dataset used for evaluation. It also means that FRASS and our string based methods can complement each other. A combination of the different approaches may well lead to a method with enhanced sensitivity.

Runtimes of URSULA and FREEDOLIN were compared to those of ALFONS (Fig. 5.5 A and B) and FRASS (Fig. 5.5 C). Let us first consider the comparison to ALFONS. Figure 5.5 A shows that the preprocessing of the chains is slower for ALFONS than for FREEDOLIN. This difference is due to the computation of base pair information used by ALFONS. However, in many applications the preprocessing step is executed only once and the computed features are stored to be reused many times afterwards. It is therefore more interesting to look at the runtimes of the chain comparisons. Only small differences were observed between the alignment-free methods and ALFONS for pairs of short chains while substantially larger differences were observed for long chains (Fig. 5.5 B). Hence, using the alignment-free

methods is especially useful for the comparison of large structures while their usefulness is limited for large numbers of small chains. To compare the computational speed of URSULA, FREEDOLIN and FRASS, the time required for a database query was measured as described in section 5.2.6. The times plotted in Figure 5.5 C comprise the processing of the query chain and the comparison to each chain in the Rfam set. FRASS is slightly faster for small query chains than the string based methods but substantially slower for large query chains. This is easily understood looking at the runtime complexities of the steps performed by FRASS. FRASS comparisons run in constant time ($O(N)$) while the computation of the Gaussian integrals has time complexity $O(N^3)$. For large structures, FRASS spends most of the time computing the Gaussian integrals for the query. The runtime performance of FRASS thus depends on the context that the program is used in. If Gaussian integrals for most chains are precomputed and stored, FRASS profits from the fast comparisons. If the descriptors have to be computed on the fly, however, this benefit is lost. Given the times measured for FREEDOLIN and URSULA, the two tools should be fast enough for most tasks that require rapid structure comparisons.

Additionally to the previously described runtime comparison with ALFONS, it would be interesting to benchmark FREEDOLIN and URSULA against an alignment tool in the database search scenario. One could run and evaluated the database scans on the Rfam set using ALFONS. However, the results will not only depend on the alignment and superposition of the nucleotides but also on the scoring function used to rank the alignments. The size-independent scores proposed by Zheng et al. [111] and Gong et al. [120] could be adequate choices. Another scoring function based on distance matrices is currently designed and tested in our group.

The results presented in this chapter concern database searches. Clustering a large number of structures is another application where alignment-free methods can be very useful. This task is a bit more difficult since there are more requirements for the similarity score than in the database search scenario. The score function should be size independent and therefore normalized in a sensible way. Many clustering algorithms also need a symmetric score. Both requirements are not yet met by the FREEDOLIN and URSULA scores. The normalization used for the average common substring method for genome sequences [145] may be a good starting point for a normalization of the FREEDOLIN score. However, there are differences between the comparison of the RNA structure alphabet strings and the comparison of genome sequences. One of them is the range of

string lengths. The PDB contains many small structures like simple hairpins for which there are many similar substructures found in large molecules like ribosomal subunits. Additionally, there are many structure files that contain only a piece of the chain of a larger molecule. It is very difficult and sometimes impossible to distinguish automatically between those two cases. Another feature of RNA structures which is specifically important for the string representation used here is that the chain models are often discontinuous because some nucleotides are missing. In FREEDOLIN and URSULA sentinel characters are inserted at the respective string positions to avoid meaningless matches. The weight function accounts for this by incorporating the number of sentinel-free substrings of the given length. Both features, the variations in chain lengths and the discontinuities, should be kept in mind when formulating a normalization scheme.

Another way to perform a fast clustering of RNA chains is to combine an alignment-free and an alignment method. The alignment-free method can be used to find next neighbor candidates for each chain. Subsequently, alignments are computed for all neighbor candidate pairs. The result is a sparse graph with weighted edges. A graph clustering algorithm such as spectral [160], modularity [161] or affinity propagation clustering [88] can then be used to group the chains based on that graph. The FREEDOLIN and URSULA scores could be used in this scenario without further modification. A similar approach has been implemented previously to cluster protein structures [130].

FREEDOLIN and URSULA both rely on substring matches and thus use local structural similarities to quickly get an idea about the overall similarity of two structures. Different methods to find similar substructures in RNA chains have been proposed [78, 121, 162] and could be added to the benchmark described in this chapter. One could either use the size of the largest match, as done by URSULA, or add contributions of multiple substructure matches, as done by FREEDOLIN.

The string representation used by FREEDOLIN and URSULA encodes only local information of the RNA backbone. A lot of available information about the structure is ignored this way. Incorporating at least some information about non-local interactions may help to improve the specificity of the scores. A simple approach would be to incorporate the base pair string encoding described in section 4.2.2. Each letter in the structural alphabet could be combined with each of the three base pair states yielding a new alphabet with 18 letters. While increasing the specificity, this may

decrease the sensitivity of the score function. In order to improve the sensitivity, one could adapt approaches that account for common substrings with mismatches [163,164]. More inspiration on how to incorporate non-local interactions into alignment-free RNA structure comparison can be taken from methods for protein comparison. For example, Cui et al. described contact libraries that include both local and remote contacts within protein chains [150]. Recently, two studies presented the use of deep neural networks to automatically extract the features used for alignment-free structure comparison [151,152].

5.5. Conclusion

The idea of longest common substrings was applied to strings of an RNA structure alphabet in order to develop alignment-free methods for the fast comparison of RNA 3D structures. A weight function for substring matches was introduced to adapt the methodology to the properties of the structure alphabet strings. This led to the implementation of the tools FREEDOLIN and URSULA. Both tools are fast enough to compare thousands of structure pairs within seconds. In the database search benchmark, FREEDOLIN and URSULA achieved results similar to slightly better than the state of the art program FRASS. In future applications, they should be useful for quick extraction of the most similar structures from large datasets. The presented approaches are also promising as seeds for the development of more sophisticated scoring methods.

Chapter 6.

6 Conclusions and outlook

This work addressed the computational prediction and comparison of RNA 3D structure models. This led to the development of the following tools and frameworks:

- The structure prediction tool NASTI, which was developed removing numerical instabilities from the program NAST.
- A new framework to create structural alphabets for RNA.
- The alignment program ALFONS, which uses one of the new structural alphabets.
- Another structural alignment program CVRRY, that applies novel backbone descriptors.
- Two tools FREEDOLIN and URSULA for fast alignment-free structure comparison that also make use of a structural alphabet designed with the new method.

Example results produced by these tools were shown and discussed in the respective chapters. The usefulness of the tools was demonstrated in comparison to state of the art tools. NASTI simulations and samples were compared to those of its predecessor NAST, ALFONS and CVRRY were compared to three state of the art alignment programs and FREEDOLIN and ALFONS were benchmarked against the alignment-free comparison tool FRASS. All approaches performed as well or better than their competitors from the literature.

In retrospect one can clearly see how structure prediction and comparison are related. Both tasks rely heavily on the representation of the molecule.

All tools described in this work use some piece wise geometric description of the backbone. NASTI, CVRRY and ALFONS additionally incorporate information about base pairs, and the treatment of helix and non-helix parts of the RNA backbone was a relevant issue in all chapters of this thesis. Another common principle used by all approaches is the extraction of knowledge from the PDB. Experimentally determined structure files were used to parameterise the NASTI energy terms, to search for an optimal structure alphabet representation and to find parameters for CVRRY and ALFONS alignments. In FREEDOLIN and ALFONS, substring matches were weighted by the distribution of structure alphabet letters found in PDB structures. The choice of the dataset used for the parameterisation, however, is not a trivial task. Approaches to solve different problems can face similar problems here. In comparison to protein structures the number of available folds for RNAs seems rather limited. There is also the problem of redundancy within the set of known structures. The large number of tRNA structures or ribosomes do not provide much independent information. Another problem is the uneven distribution of chain lengths. On the one side, there is a large number of small (< 200 nucleotides) structures. On the other side, there is a large number of ribosomal structures ($> 10^3$ nucleotides). Data between these extremes is rather sparse. These problems were handled in multiple ways in this work. The NASTI parameters were obtained only from ribosomal subunits relying on the belief that their substructures are representative for the whole RNA fold space. A different approach was used for the optimization of structural alphabets and the search of alignment parameters. Here, ribosomal chains were cut to smaller pieces in order to enable effective optimization procedures. In the future, more sophisticated solutions could help to improve both comparison and prediction of RNA structures. Finally one could state that the choices on how to pick, design and combine the molecule representation, the algorithms and the parameters are in all cases subject to computational speed and accuracy. In most cases, there is a compromise to make which one would like to make as smart as possible.

There are multiple ways in which each of the developed methods may contribute to future developments in the field. Herein, the contributions are roughly grouped in three categories.

First, each of the tools can be downloaded and applied as it is. The respective tool may either be used on its own or it could be incorporated into some data processing pipeline or interactive tool. It seems sensible, for example, to replace NAST with NASTI in the pipeline described by Weinreb et al. [33].

One could also integrate ALFONS into chimera [165], a tool for interactive analysis of macromolecules, to superimpose chains and interactively analyze conserved and non-conserved sites between a pair of chains. Alternatively, the structural alphabet and enhanced suffix arrays could be used to highlight similar backbone stretches in an interactive environment.

The second type of contributions provided by the developments of this work are the insights about the developed methods and the transfer of that knowledge to other research areas. The description of the numerical instabilities found in the NAST force field and the provided solution, for example, should be helpful for future low resolution force field developments. Chapter 3 describes a new approach for the design of a structural alphabet which is fundamentally different to the established unsupervised clustering approach. It is obvious that this method can be applied to protein structures as well. One can even take this a step further and adapt the method to any type of data, for which a string representation is useful when searching for patterns. This will, however, require the formulation of an appropriate target function.

The third way in which the described methods and tools contribute to the future of RNA structure prediction and comparison is to use them as seed-ideas for further developments. Publicly available code is very beneficial here. Further efforts can be made to refine and enhance the existing tools. The tools can also be combined with each other or with different existing approaches. Several suggestions for improvements and research directions were already made in the respective chapters. A few more ideas are added in the following.

6.1. Ideas for further developments

Models of large RNA structures are often missing parts of the chain. One could design a pipeline which combines NASTI and either CVRRY, ALFONS or both alignment tools in order to model the missing parts. Given a structure model with missing nucleotides, one first performs a database search to find similar structures. The search could be conducted with the alignment tool or with either FREEDOLIN or URSULA. Afterwards, the query structure is aligned to the most similar hits from the database search. An alignment with superposition refinement is used to assign the corresponding residues. If the missing nucleotides are present in one or more of the structures

found in the database, the respective coordinates can be used to generate position or distance and angle restraints. NASTI could then be used to sample possible backbone conformations of the missing stretches. The program C2A [166] could be used afterwards to add the fine grained atom coordinates. This approach is closely related to homology modeling [167] and loop modeling [21]. In contrast to homology modeling, only structural similarity is required and sequence similarity is not necessary. An advantage compared to other loop modeling approaches is the use of knowledge from similar structures.

Section 4.4 described how missing pieces in structure models are problematic for alignment tools like CVRRY and ALFONS, which are based on backbone descriptors and dynamic programming methods as used in sequence alignments. An approach to enhance the performance of the alignment algorithms could be implemented using NASTI. In a first step, the missing regions are modeled with NASTI. Information about base pairs and tertiary interactions is required for some regions in order to get realistic results. Multiple alignments can help to get information about such interactions [33]. After the structure model is completed, the chains are aligned. This is likely to improve the result even if the modeled regions are not 100 % correct. There are two reasons. First, some regions will be similar to the real structure and may guide the alignment. For example, modeled helix regions are very likely to take on a helical shape due to the helix restraints used in NASTI and therefore match their counterparts. Second, gaps now only serve one function, which is to be inserted at positions of real insertions, instead of compensating missing residues.

It was mentioned earlier in this chapter that our new method for structural alphabet design could be adapted to find string representations for data types other than RNA 3D structures. The most obvious case is the application to protein structures. This may even turn out more successful than the RNA alphabet due to the larger number of chains available which can be used in the optimization. Although there are already many structural alphabets for proteins available, this may still be interesting since the new optimization approach is fundamentally different from most methods. A somewhat related method was presented by Ku and Hu [168]. This approach starts with unsupervised clustering to generate the alphabet. Subsequently, a scoring matrix is optimized through iterative database searches. The target function is based on database annotations and evaluates whether structures from the same classes are found. In contrast to our method, no structure superpositions are made. A detailed comparison between this approach and

our method would be interesting. With some modification one could also use our new method to optimize an alphabet for the reduced representation of amino acid sequences [169]. One can use the same target function as before and hence still optimize with respect to the similarity of 3D backbone stretches. Instead of mapping fragments defined by backbone coordinates one defines a mapping of either single residues or k-mers to a new set of letters. It was also mentioned before that in principle this methodology could be applied to a large variety of data. This requires a meaningful representation of the data in sequence form and a useful target function to assess the quality of matches. The detection of patterns in music should be a suitable task. Music has already a natural sequential order along the time axis and methods developed for biological sequences were applied in this research field before [170]. After designing an alphabet for music pattern recognition, the FREEDOLIN methodology could be adapted for fast alignment-free comparisons.

Multiple alignments of macromolecular sequences [143,144,171] or structures [73,138,172–175] can provide useful information about structural and functional relationships between molecules. Especially sequence based methods gained a lot of attention in the past. Many tools were developed and optimized with respect to speed and accuracy in order to cope with large datasets [143,144,171]. So far, only few tools exist for multiple alignments of RNA 3D structures [73,138,172,173]. The new structural alphabet from this work is well suited for the application of multiple sequence alignment methods. Such an approach could be especially useful to align large groups of long chains. Many multiple sequence alignment tools implement the progressive alignment method [143,144,171]. Therein, a guide tree determines the order in which the sequences are assembled to build the multiple alignment. Building the guide tree requires the comparison of all sequence pairs. An alignment method can be used here, but is too expensive for large datasets. Alignment-free methods help to speed up the process [143,144]. To quickly align many large RNA structures using our new structural alphabet representation, a size independent and symmetric version of the FREEDOLIN score would be helpful to build the guide tree.

Further inspiration for the application of structural alphabets can be found in the literature. Here are two examples. Several approaches use seeded alignment methods on structural alphabet strings [68,74,78]. These heuristics compute alignments much faster than the classical dynamic programming alignment methods. Typically, significant exact substring matches are searched first and extended subsequently [176]. Another application, for

which a structural alphabet is useful, is the detection of conserved pieces in a large set of structures. Wu et al. described a method to find structural motifs that occur in different protein families [177]. Both approaches are easily adapted to be used on RNA structures with our new alphabet.

Bibliography

[1] Cech TR, Steitz JA. 2014. The noncoding RNA revolution - Trashing old rules to forge new ones. *Cell* **157**: 77–94.

[2] Laing C, Schlick T. 2011. Computational approaches to RNA structure prediction, analysis, and design. *Curr Opin Struct Biol* **21**: 306–318.

[3] Dans PD, Gallego D, Balaceanu A, Darré L, Gómez H, Orozco M. 2019. Modeling, simulations, and bioinformatics at the service of RNA structure. *Chem* **5**: 51–73.

[4] Terwilliger TC, Adams PD, Afonine PV, Sobolev OV. 2018. A fully automatic method yielding initial models from high-resolution cryo-electron microscopy maps. *Nat Methods* **15**: 905–908.

[5] Voet D, Voet J, Pratt C. 2017. *Fundamentals of Biochemistry: Life at the Molecular Level*. John Wiley & Sons, Hoboken.

[6] Tinoco I, Bustamante C. 1999. How RNA folds. *J Mol Biol* **293**: 271–281.

[7] Wang Z, Gerstein M, Snyder M. 2009. RNA-Seq: a revolutionary tool for transcriptomics. *Nat Rev Genet* **10**: 57–63.

[8] Merino EJ, Wilkinson KA, Coughlan JL, Weeks KM. 2005. RNA structure analysis at single nucleotide resolution by Selective 2'-Hydroxyl Acylation and Primer Extension (SHAPE). *J Am Chem Soc* **127**: 4223–4231.

[9] Mortimer SA, Weeks KM. 2007. A fast-acting reagent for accurate analysis of RNA secondary and tertiary structure by SHAPE chemistry. *J Am Chem Soc* **129**: 4144–4145.

[10] Felden B. 2007. RNA structure: experimental analysis. *Curr Opin Microbiol* **10**: 286–291.

[11] Bai XC, McMullan G, Scheres SH. 2015. How cryo-EM is revolutionizing structural biology. *Trends Biochem Sci* **40**: 49–57.

[12] Berman H, Henrick K, Nakamura H. 2003. Announcing the worldwide Protein Data Bank. *Nat Struct Biol* **10**: 980.

[13] Ding F, Sharma S, Chalasani P. 2008. Ab initio RNA folding by discrete molecular dynamics: from structure prediction to folding mechanisms. *RNA* **14**: 1164–1173.

[14] Jonikas MA, Radmer RJ, Laederach A, Das R, Pearlman S, Herschlag D, Altman RB. 2009. Coarse-grained modeling of large RNA molecules with knowledge-based potentials and structural filters. *RNA* **15**: 189–199.

[15] Xia Z, Gardner DP, Gutell RR, Ren P. 2010. Coarse-grained model for simulation of RNA three-dimensional structures. *J Phys Chem B* **114**: 13497–506.

[16] Bernauer J, Huang X, Sim AY, Levitt M. 2011. Fully differentiable coarse-grained and all-atom knowledge-based potentials for RNA structure evaluation. *RNA* **17**: 1066–1075.

[17] Šulc P, Romano F, Ouldridge TE, Doye JPK, Louis AA. 2014. A nucleotide-level coarse-grained model of RNA. *J Chem Phys* **140**: 235102.

[18] Cragnolini T, Laurin Y, Derreumaux P, Pasquali S. 2015. Coarse-grained HiRE-RNA model for ab initio RNA folding beyond simple molecules, including noncanonical and multiple base pairings. *J Chem Theory Comput* **11**: 3510–3522.

[19] Kerpedjiev P, Hoener zu Siederdiessen C, Hofacker IL. 2015. Predicting RNA 3D structure using a coarse-grain helix-centered model. *RNA* **21**: 1110–1121.

[20] Boniecki MJ, Lach G, Dawson WK, Tomala K, Lukasz P, Soltysinski T, Rother KM, Bujnicki JM. 2016. SimRNA: A coarse-grained method for RNA folding simulations and 3D structure prediction. *Nucleic Acids Res* **44**: e63.

[21] Li J, Zhang J, Wang J, Li W, Wang W. 2016. Structure prediction of RNA loops with a probabilistic approach. *PLoS Comput Biol* **12**: e1005032.

[22] Boudard M, Barth D, Bernauer J, Denise A, Cohen J. 2017. GARN2: coarse-grained prediction of 3D structure of large RNA molecules by regret minimization. *Bioinformatics* **33**: 2479–2486.

[23] Bell DR, Cheng SY, Salazar H, Ren P. 2017. Capturing RNA folding free energy with coarse-grained molecular dynamics simulations. *Sci Rep* **7**: 45812.

[24] Poblete S, Bottaro S, Bussi G. 2018. A nucleobase-centered coarse-grained representation for structure prediction of RNA motifs. *Nucleic Acids Res* **46**: 1674–1683.

[25] Cao S, Chen SJ. 2011. Physics-based de novo prediction of RNA 3D structures. *J Phys Chem B* **115**: 4216–26.

[26] Frellsen J, Moltke I, Thiim M, Mardia KV, Ferkinghoff-Borg J, Hamelryck T. 2009. A probabilistic model of RNA conformational space. *PLoS Comput Biol* **5**: e1000406.

[27] Wang Z, Parisien M, Scheets K, Miller WA. 2011. The cap-binding translation initiation factor, eIF4E, binds a pseudoknot in a viral cap-independent translation element. *Structure* **19**: 868–880.

[28] Das R, Karanicolas J, Baker D. 2010. Atomic accuracy in predicting and designing noncanonical RNA structure. *Nat Methods* **7**: 291–294.

[29] Mathews DH, Turner DH. 2006. Prediction of RNA secondary structure by free energy minimization. *Curr Opin Struct Biol* **16**: 270–278.

[30] Rivas E. 2013. The four ingredients of single-sequence RNA secondary structure prediction. A unifying perspective. *RNA Biol* **10**: 1185–1196.

[31] Hamada M, Sato K, Asai K. 2011. Improving the accuracy of predicting secondary structure for aligned RNA sequences. *Nucleic Acids Res* **39**: 393–402.

[32] Puton T, Kozlowski LP, Rother KM, Bujnicki JM. 2013. CompaRNA: A server for continuous benchmarking of automated methods for RNA secondary structure prediction. *Nucleic Acids Res* **41**: 4307–4323.

[33] Weinreb C, Riesselman AJ, Ingraham JB, Gross T, Sander C, Marks DS. 2016. 3D RNA and functional interactions from evolutionary couplings. *Cell* **165**: 963–975.

[34] Needleman SB, Wunsch CD. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* **48**: 443–453.

[35] Smith TF, Waterman M. 1981. Identification of common molecular subsequences. *J Mol Biol* **147**: 195–197.

[36] Gotoh O. 1982. An improved algorithm for matching biological sequences. *J Mol Biol* **162**: 705–708.

[37] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. 1990. Basic local alignment search tool. *J Mol Biol* **215**: 403–410.

[38] Illergård K, Ardell DH, Elofsson A. 2009. Structure is three to ten times more conserved than sequence - A study of structural response in protein cores. *Proteins* **77**: 499–508.

[39] Capriotti E, Marti-Renom MA. 2010. Quantifying the relationship between sequence and three-dimensional structure conservation in RNA. *BMC Bioinformatics* **11**: 322.

[40] Petersen NP, Ort T, Torda AE. 2019. Improving the numerical stability of the NAST force field for RNA simulations. *J Chem Theory Comput* **15**: 3402–3409.

[41] Kmiecik S, Gront D, Kolinski M, Wieteska L, Dawid AE, Kolinski A. 2016. Coarse-grained protein models and their applications. *Chem Rev* **116**: 7898–7936.

[42] Kar P, Feig M. 2014. Recent advances in transferable coarse-grained modeling of proteins. *Adv Protein Chem Struct Biol* **96**: 143–180.

[43] Sen A, Thakur S, Bothra AK, Sur S, Tisa LS. 2012. Identification of TTA codon containing genes in Frankia and exploration of the role of tRNA in regulating these genes. *Arch Microbiol* **194**: 35–45.

[44] Jung S, Schlick T. 2013. Candidate RNA structures for domain 3 of the foot-and-mouth-disease virus internal ribosome entry site. *Nucleic Acids Res* **41**: 1483–1495.

[45] Mitrasinovic PM, Tomar JS, Nair MS, Barthwal R. 2011. Modeling of HIV-1 TAR RNA-ligand complexes. *Med Chem* **7**: 301–308.

[46] Chen C, Mitra S, Jonikas M, Martin J, Brenowitz M, Laederach A. 2013. Understanding the role of three-dimensional topology in determining the folding intermediates of group I introns. *Biophys J* **104**: 1326–1337.

[47] Zeng Y, Larson SB, Heitsch CE, McPherson A, Harvey SC. 2012. A model for the structure of satellite tobacco mosaic virus. *J Struct Biol* **180**: 110–116.

[48] Winger M, Trzesniak D, Baron R, van Gunsteren WF. 2009. On using a too large integration time step in molecular dynamics simulations of coarse-grained molecular models. *Phys Chem Chem Phys* **11**: 1934–1941.

[49] Zemla A, Venclovas C, Moult J, Fidelis K. 1999. Processing and analysis of CASP3 protein structure predictions. *Proteins* **S3**: 22–29.

[50] Leontis NB, Zirbel CL. 2012. Nonredundant 3D structure datasets for RNA knowledge extraction and benchmarking. In *RNA 3D Structure Analysis and Prediction* (eds. NB Leontis, E Westhof). Springer, Berlin, Heidelberg, pp. 281–298.

[51] Wiegels T, Bienert S, Torda AE. 2013. Fast alignment and comparison of RNA structures. *Bioinformatics* **29**: 588–596.

[52] Capriotti E, Marti-Renom MA. 2008. RNA structure alignment by a unit-vector approach. *Bioinformatics* **24**: i112–i118.

[53] Yang H, Jossinet F, Leontis N, Chen L, Westbrook J, Berman H, Westhof E. 2003. Tools for the automatic identification and classification of RNA base pairs. *Nucleic Acids Res* **31**: 3450–3460.

[54] Scott DW. 1979. On optimal and data-based histograms. *Biometrika* **66**: 605–610.

[55] Dierckx P. 1975. An algorithm for smoothing, differentiation and integration of experimental data using spline functions. *J Comput Appl Math* **1**: 165–184.

[56] SciPy 0.19.1 `http://www.scipy.org`.

[57] Mardia KV, Jupp PE. 2008. *Directional Statistics*. John Wiley & Sons, Chichester.

[58] NAST 1.0 `https://simtk.org/projects/nast`.

[59] Moore D, McCable G, Craig B. 2009. *Introduction to the Practice of Statistics*. W. H. Freeman and Company, New York.

[60] Cao Y, Petzold L. 2006. Accuracy limitations and the measurement of errors in the stochastic simulation of chemically reacting systems. *J Comput Phys* **212**: 6–24.

[61] Eastman P, Swails J, Chodera JD, McGibbon RT, Zhao Y, Beauchamp KA, Wang LP, Simmonett AC, Harrigan MP, Stern CD, et al. 2017. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS Comput Biol* **13**: e1005659.

[62] Python 2.7.13 `https://www.python.org`.

[63] Thomas PD, Dill KA. 1996. Statistical potentials extracted from protein structures: How accurate are they? *J Mol Biol* **257**: 457–469.

[64] Bayrak CS, Kim N, Schlick T. 2017. Using sequence signatures and kink-turn motifs in knowledge-based statistical potentials for RNA structure prediction. *Nucleic Acids Res* **45**: 5414–5422.

[65] Deschavanne P, Tufféry P. 2009. Enhanced protein fold recognition using a structural alphabet. *Proteins* **76**: 129–137.

[66] Etchebest C, Benros C, Hazout S, De Brevern AG. 2005. A structural alphabet for local protein structures: Improved prediction methods. *Proteins* **59**: 810–827.

[67] Guyon F, Camproux AC, Hochez J, Tufféry P. 2004. SA-Search: A web tool for protein structure mining based on a structural alphabet. *Nucleic Acids Res* **32**: 545–548.

[68] Tung CH, Huang JW, Yang JM. 2007. Kappa-alpha plot derived structural alphabet and BLOSUM-like substitution matrix for rapid search of protein structure database. *Genome Biol* **8**: R31.

[69] Gelly JC, Joseph AP, Srinivasan N, De Brevern AG. 2011. iPBA: A tool for protein structure comparison using sequence alignment strategies. *Nucleic Acids Res* **39**: 18–23.

[70] Camproux AC, Tuffery P, Chevrolat JP, Boisvieux JF, Hazout S. 1999. Hidden Markov model approach for identifying the modular framework of the protein backbone. *Protein Eng* **12**: 1063–1073.

[71] Allam I, Flatters D, Caumes G, Regad L, Delos V, Nuel G, Camproux AC. 2018. SAFlex: A structural alphabet extension to integrate protein structural flexibility and missing data information. *PLoS One* **13**: e0198854.

[72] Tyagi M, Gowri VS, Srinivasan N, Brevern AG, Offman B. 2006. A substitution matrix for structural alphabet based on structural alignment of homologous proteins and its applications. *Proteins* **65**: 32–39.

[73] Chang YF, Huang YL, Lu CL. 2008. SARSA: a web tool for structural alignment of RNA using a structural alphabet. *Nucleic Acids Res* **36**: W19–W24.

[74] Liu YC, Yang CH, Chen KT, Wang JR, Cheng ML, Chung JC, Chiu HT, Lu CL. 2011. R3D-BLAST: A search tool for similar RNA 3D substructures. *Nucleic Acids Res* **39**: W45–W49.

[75] Bauer RA, Rother K, Moor P, Reinert K, Steinke T, Bujnicki JM, Preissner R. 2009. Fast structural alignment of biomolecules using a hash table, n-grams and string descriptors. *Algorithms* **2**: 692–709.

[76] Wang CW, Chen KT, Lu CL. 2010. iPARTS: An improved tool of pairwise alignment of RNA tertiary structures. *Nucleic Acids Res* **38**.

[77] Yang CH, Shih CT, Chen KT, Lee PH, Tsai PH, Lin JC, Yen CY, Lin TY, Lu CL. 2016. iPARTS2: an improved tool for pairwise alignment of RNA tertiary structures, version 2. *Nucleic Acids Res* **44**: W328–W332.

[78] Yen CY, Lin JC, Chen KT, Lu CL. 2017. R3D-BLAST2: an improved search tool for similar RNA 3D substructures. *BMC Bioinformatics* **18**: 574.

[79] Wadley LM, Keating KS, Duarte CM, Pyle AM, Haven N, Haven N. 2007. Evaluating and learning from RNA pseudotorsional space: quantitative validation of a reduced representation for RNA structure. *J Mol Biol* **372**: 942–957.

[80] Henikoff S, Henikoff JG. 1992. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci USA* **89**: 10915–10919.

[81] Murray LJ, Arendall WB, Richardson DC, Richardson JS. 2003. RNA backbone is rotameric. *Proc Natl Acad Sci USA* **100**: 13904–13909.

[82] Richardson JS, Schneider B, Murray LW, Kapral GJ, Immormino RM, Headd JJ, Richardson DC, Ham D, Hershkovits E, Williams LD, et al. 2008. RNA backbone: Consensus all-angle conformers and modular string nomenclature (an RNA Ontology Consortium contribution). *RNA* **14**: 465–481.

[83] Ma B, Tromp J, Li M. 2002. PatternHunter: Faster and more sensitive homology search. *Bioinformatics* **18**: 440–445.

[84] Kasai T, Lee G, Arimura H, Arikawa S, Park K. 2001. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching* (ed. A Amir). Springer, Berlin, Heidelberg, pp. 181–192.

[85] Nourani Y, Andresen B. 1998. A comparison of simulated annealing cooling strategies. *J Phys A Math Gen* **31**: 8373–8385.

[86] sais-lite 2.4.1. `https://sites.google.com/site/yuta256/sais`.

[87] Nong G, Zhand S, Chan WH. 2011. Two efficient algorithms for linear time suffix array construction. *IEEE Transactions on Computers* **60**: 1471–1484.

[88] Dueck D, Frey BJ. 2007. Clustering by passing messages between data points. *Science* **315**: 972–976.

[89] Hotelling H. 1933. Analysis of a complex of statistical variables into principal components. *J Educ Psychol* **6**: 417–441.

[90] Kabsch W. 1976. A solution for the best rotation to relate two sets of vectors. *Acta Cryst A* **32**: 922–923.

[91] The GNU Compiler Collection `https://gcc.gnu.org/`.

[92] Python 3.6.5 `https://www.python.org`.

[93] SWIG 4.0. `http://swig.org`.

[94] OpenMP 4.5. `https://www.openmp.org`.

[95] scikit-learn 0.21.2 `https://scikit-learn.org`.

[96] Karlin S, Altschul SF. 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc Natl Acad Sci USA* **87**: 2264–2268.

[97] Gen M, Lin L. 2008. Genetic algorithms. In *Wiley encyclopedia of computer science and engeneering* (ed. BW Wah). John Wiley & Sons, New York.

[98] de la Maza M, Tidor B. 1991. Boltzmann weighted selection improves performance of genetic algorithms. Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA.

[99] Hukushima K, Iba Y. 2003. Population annealing and its application to a spin glass. *AIP Conference Proceedings* **690**: 200–206.

[100] Chothia C, Lesk A. 1986. The relation between the divergence of sequence and structure in proteins. *EMBO J* **5**: 823–826.

[101] Gan HH, Perlow RA, Roy S, Ko J, Wu M, Huang J, Yan S, Nicoletta A, Vafai J, Sun D, et al. 2002. Analysis of protein sequence/structure similarity relationships. *Biophys J* **83**: 2781–2791.

[102] Dror O, Nussinov R, Wolfson HJ. 2006. The ARTS web server for aligning RNA tertiary structures. *Nucleic Acids Res* **34**: W412–W415.

[103] Capriotti E, Marti-Renom MA. 2009. SARA: A server for function annotation of RNA structures. *Nucleic Acids Res* **37**: W260–W265.

[104] Margraf T, Schenk G, Torda AE. 2009. The SALAMI protein structure search server. *Nucleic Acids Res* **37**: W480–W484.

[105] Holm L. 2020. DALI and the persistence of protein shape. *Protein Sci* **29**: 128–140.

[106] Orengo C. 1994. Classification of protein folds. *Curr Opin Struct Biol* **4**: 429–440.

[107] Ouzounis CA, Coulson RM, Enright AJ, Kunin V, Pereira-Leal JB. 2003. Classification schemes for protein structure and function. *Nat Rev Genet* **4**: 508–519.

[108] Kolodny R, Petrey D, Honig B. 2006. Protein structure comparison: implications for the nature of 'fold space', and structure and function prediction. *Curr Opin Struct Biol* **16**: 393–398.

[109] Abraham M, Dror O, Nussinov R, Wolfson HJ. 2008. Analysis and classification of RNA tertiary structures. *RNA* **14**: 2274–2289.

[110] Zheng J, Kundrotas PJ, Vakser IA, Liu S. 2016. Template-based modeling of protein-RNA interactions. *PLoS Comput Biol* **12**: e1005120.

[111] Jinfang Z, Xie J, Hong X, Liu S. 2019. RMalign: an RNA structural alignment tool based on a size independent scoring function. *BMC Genomics* **20**: 276.

[112] Hasegawa H, Holm L. 2009. Advances and pitfalls of protein structural alignment. *Curr Opin Struct Biol* **19**: 341–348.

[113] Ma J, Wang S. 2014. Algorithms, applications, and challenges of protein structure alignment. In *RNA 3D Structure Analysis and Prediction* (ed. R Donev). Academic Press, Cambridge, pp. 121–175.

[114] Ferrè F, Ponty Y, Lorenz WA, Clote P. 2007. DIAL: A web server for the pairwise alignment of two RNA three-dimensional structures using nucleotide, dihedral angle and base-pairing similarities. *Nucleic Acids Res* **35**: W659–W668.

[115] Dror O, Nussinov R, Wolfson H. 2005. ARTS: Alignment of RNA tertiary structures. *Bioinformatics* **21**: ii47–ii53.

[116] Nguyen MN, Sim AY, Wan Y, Madhusudhan MS, Verma C. 2017. Topology independent comparison of RNA 3D structures using the CLICK algorithm. *Nucleic Acids Res* **45**: e5.

[117] Rahrig RR, Leontis NB, Zirbel CL. 2010. R3D align: Global pairwise alignment of RNA 3D structures using local superpositions. *Bioinformatics* **26**: 2689–2697.

[118] Hoksza D, Svozil D. 2012. Efficient RNA pairwise structure comparison by SETTER method. *Bioinformatics* **28**: 1858–1864.

[119] Ge P, Zhang S. 2015. STAR3D: A stack-based RNA 3D structural alignment tool. *Nucleic Acids Res* **43**: e137.

[120] Gong S, Zhang C, Zhang Y. 2019. RNA-align: quick and accurate alignment of RNA 3D structures based on size-independent TM-score$_{\text{RNA}}$. *Bioinformatics* **35**: 4459–4461.

[121] Zahran M, Bayrak CS, Elmetwaly S, Schlick T. 2015. RAG-3D: a search tool for RNA 3D substructures. *Nucleic Acids Res* **43**: 9474–9488.

[122] Zhang Y, Skolnick J. 2005. TM-align: A protein structure alignment algorithm based on the TM-score. *Nucleic Acids Res* **33**: 2302–2309.

[123] Schawo J. 2017. *Fragment-based Structural Alignments of RNA Molecules*. Master thesis, University Hamburg.

[124] Nguyen MN, Verma C. 2015. Rclick: A web server for comparison of RNA 3D structures. *Bioinformatics* **31**: 966–968.

[125] Rahrig RR, Petrov AI, Leontis NB, Zirbel CL. 2013. R3D Align web server for global nucleotide to nucleotide alignments of RNA 3D structures. *Nucleic Acids Res* **41**: W15–W21.

[126] Braun W, Bösch C, Brown LR, Nobuhiro G, Wüthrich K. 1981. Combined use of proton-proton overhauser enhancements and a distance geometry algorithm for determination of polypeptide conformations. Application to micelle-bound glucagon. *Biochim Biophys Acta* **667**: 377–396.

[127] Lu XJ, Bussemaker HJ, Olson WK. 2015. DSSR: An integrated software tool for dissecting the spatial structure of RNA. *Nucleic Acids Res* **43**: e142.

[128] Levitt M, Huber R. 1983. Molecular dynamics of native protein. II. Analysis and nature of motion. *J Mol Biol* **168**: 621–657.

[129] Russell AJ, Torda AE. 2002. Protein sequence threading: Averaging over structures. *Proteins* **47**: 496–505.

[130] Margraf TA. 2012. *Applications of fast protein structure alignments*. Ph.D. thesis, University Hamburg.

[131] Siew N, Elofsson A, Rychlewski L, Fischer D. 2000. MaxSub: An automated measure for the assessment of protein structure prediction quality. *Bioinformatics* **16**: 776–785.

[132] Press W, Teukolsky S, Vetterling W, Flannery B. 1992. *Numerical Recipes*. Cambridge University Press, Cambridge.

[133] Schenk G, Margraf T, Torda AE. 2008. Protein sequence and structure alignments within one framework. *Algorithms Mol Biol* **3**: 4.

[134] Lackner P, Koppensteiner WA, Sippl MJ, Domingues FS. 2000. ProSup: a refined tool for protein structure alignment. *Protein Eng* **13**: 745–752.

[135] Perl 5.26.1 `https://www.perl.org`.

[136] Schwede T, Kopp J, Guex N, Peitsch MC. 2003. SWISS-MODEL: An automated protein homology-modeling server. *Nucleic Acids Res* **31**: 3381–3385.

[137] Nielsen M, Lundegaard C, Lund O, Petersen TN. 2010. CPHmodels-3.0-remote homology modeling using structure-guided sequence profiles. *Nucleic Acids Res* **38**: W576–W581.

[138] Piątkowski P, Jabłónska J, Zyła A, Niedziałek D, Matelska D, Jankowska E, Waleń T, Dawson WK, Bujnicki JM. 2017. SupeRNAlign: A new tool for flexible superposition of homologous RNA structures and inference of accurate structure-based sequence alignments. *Nucleic Acids Res* **45**: e150.

[139] Zhang Y, Skolnick J. 2004. Scoring function for automated assessment of protein structure template quality. *Proteins* **57**: 702–710.

[140] Gendron P, Lemieux S, Major F. 2001. Quantitative analysis of nucleic acid three-dimensional structures. *J Mol Biol* **308**: 919–936.

[141] Lemieux S, Major F. 2002. RNA canonical and non-canonical base pairing types: a recognition method and complete repertoire. *Nucleic Acids Res* **30**: 4250–4263.

[142] Zielezinski A, Vinga S, Almeida J, Karlowski WM. 2017. Alignment-free sequence comparison: Benefits, applications, and tools. *Genome Biol* **18**: 186.

[143] Edgar RC. 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res* **32**: 1792–1797.

[144] Edgar RC. 2004. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics* **5**: 113.

[145] Ulitsky I, Burstein D, Tuller T, Chor B. 2006. The average common substring approach to phylogenomic reconstruction. *J Comput Biol* **13**: 336–350.

[146] Rogen P, Faint B. 2003. Automatic classification of protein structure by using Gauss integrals. *Proc Natl Acad Sci USA* **100**: 119–124.

[147] Choi IG, Kwon J, Kim SH. 2004. Local feature frequency profile: A method to measure structural similarity in proteins. *Proc Natl Acad Sci USA* **101**: 3797–3802.

[148] Shyu CR, Chi PH, Scott G, Xu D. 2004. ProteinDBS: a real-time retrieval system for protein structure comparison. *Nucleic Acids Res* **32**: W572–W575.

[149] Budowski-Tal I, Nov Y, Kolodny R. 2010. FragBag, an accurate representation of protein structure, retrieves structural neighbors from the entire PDB quickly and accurately. *Proc Natl Acad Sci USA* **107**: 3481–3486.

[150] Cui X, Li SC, He L, Li M. 2014. Fingerprinting protein structures effectively and efficiently. *Bioinformatics* **30**: 949–955.

[151] Liu Y, Ye Q, Wang L, Peng J. 2018. Learning structural motif representations for efficient protein structure search. *Bioinformatics* **34**: i773–i780.

[152] Min Y, Liu S, Lou C, Cui X. 2018. Learning Protein Structural Fingerprints under the Label-Free Supervision of Domain Knowledge. In *Proc. - 2018 IEEE Int. Conf. Bioinforma. Biomed. BIBM 2018*. IEEE, pp. 69–74.

[153] Liu N, Wang T. 2006. A method for rapid similarity analysis of RNA secondary structures. *BMC Bioinformatics* **7**: 493.

[154] Heyne S, Costa F, Rose D, Backofen R. 2012. Graphclust: Alignment-free structural clustering of local RNA secondary structures. *Bioinformatics* **28**: i224–i232.

[155] Ma Q, Xie J, Zhang Y, Li Y, Shi X, Liang Y. 2017. RNA-TVcurve: a Web server for RNA secondary structure comparison based on a multi-scale similarity of its triple vector curve representation. *BMC Bioinformatics* **18**: 51.

[156] Kirillova S, Tosatto SC, Carugo O. 2010. FRASS: the web-server for RNA structural comparison. *BMC Bioinformatics* **11**: 327.

[157] Kalvari I, Nawrocki EP, Argasinska J, Quinones-olvera N, Finn D, Bateman A, Petrov AI. 2019. Non-coding RNA analysis using the Rfam database. *Curr Protoc Bioinformatics* **62**: e51.

[158] Hajian-Tilaki K. 2013. Receiver operating characteristic (ROC) curve analysis for medical diagnostic test evaluation. *Casp J Intern Med* **4**: 627–635.

[159] Gagniuc P. 2017. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, Hoboken.

[160] Ng AY, Jordan MI, Yair W. 2002. On spectral clustering: analysis and an algorithm. In *Adv. Neural Inf. Process. Syst.* (eds. T Ditterich, Z Ghahramani). Cambridge, pp. 849–856.

[161] Newman ME. 2006. Finding community structure in networks using the eigenvectors of matrices. *Phys Rev E* **74**: 36104.

[162] Lai CE, Tsai MY, Liu YC, Wang CW, Chen KT, Lu CL. 2009. FASTR3D: A fast and accurate search tool for similar RNA 3D structures. *Nucleic Acids Res* **37**: 287–295.

[163] Leimeister CA, Morgenstern B. 2014. kmacs: The k-mismatch average common substring approach to alignment-free sequence comparison. *Bioinformatics* **30**: 2000–2008.

[164] Thankachan SV, Chockalingam SP, Liu Y, Krishnan A, Aluru S. 2017. A greedy alignment-free distance estimator for phylogenetic inference. *BMC Bioinformatics* **18**: 238.

[165] Pettersen EF, Goddard TD, Huang CC, Couch GS, Greenblatt DM, Meng EC, Ferrin TE. 2004. UCSF Chimera - A visualization system for exploratory research and analysis. *J Comput Chem* **25**: 1605–1612.

[166] Jonikas MA, Radmer RJ, Altman RB. 2009. Knowledge-based instantiation of full atomic detail into coarse-grain RNA 3D structural models. *Bioinformatics* **25**: 3259–3266.

[167] Rother M, Rother K, Puton T, Bujnicki JM. 2011. ModeRNA: A tool for comparative modeling of RNA 3D structure. *Nucleic Acids Res* **39**: 4007–4022.

[168] Ku SY, Hu YJ. 2008. Protein structure search and local structure characterization. *BMC Bioinformatics* **9**: 349.

[169] Melo F, Marti-Renom MA. 2006. Accuracy of sequence alignment and fold assessment using reduced amino acid alphabets. *Proteins* **63**: 986–995.

[170] Martin B, Brown DG, Hanna P, Ferraro P. 2012. Blast for audio sequences alignment: a fast scalable cover identification. In *13th Int. Soc. Music Inf. Retr. Conf.* ISMIR, Canada, pp. 529–534.

[171] Chatzou M, Magis C, Chang JM, Kemena C, Bussotti G, Erb I, Notredame C. 2016. Multiple sequence alignment modeling: Methods and applications. *Brief Bioinform* **17**: 1009–1023.

[172] Kemena C, Bussotti G, Capriotti E, Marti-Renom MA, Notredame C. 2013. Using tertiary structure for the computation of highly accurate multiple RNA alignments with the SARA-Coffee package. *Bioinformatics* **29**: 1112–1119.

[173] Hoksza D, Svozil D. 2015. Multiple 3D RNA structure superposition using neighbor joining. *IEEE/ACM Trans Comput Biol Bioinformatics* **12**: 520–530.

[174] Léonard S, Joseph AP, Srinivasan N, Gelly JC, De Brevern AG. 2014. MulPBA: An efficient multiple protein structure alignment method based on a structural alphabet. *J Biomol Struct Dyn* **32**: 661–668.

[175] Dong R, Peng Z, Zhang Y, Yang J. 2018. mTM-align: An algorithm for fast and accurate multiple protein structure alignment. *Bioinformatics* **34**: 1719–1725.

[176] Sun Y, Buhler J. 2006. Choosing the best heuristic for seeded alignment of DNA sequences. *BMC Bioinformatics* **7**: 133.

[177] Wu CY, Chen YC, Lim C. 2010. A structural-alphabet-based strategy for finding structural motifs across protein families. *Nucleic Acids Res* **38**: e150.

A Appendix A. Supplemental Data

This section contains additional figures and tables. Figures are shown first and tables afterwards.

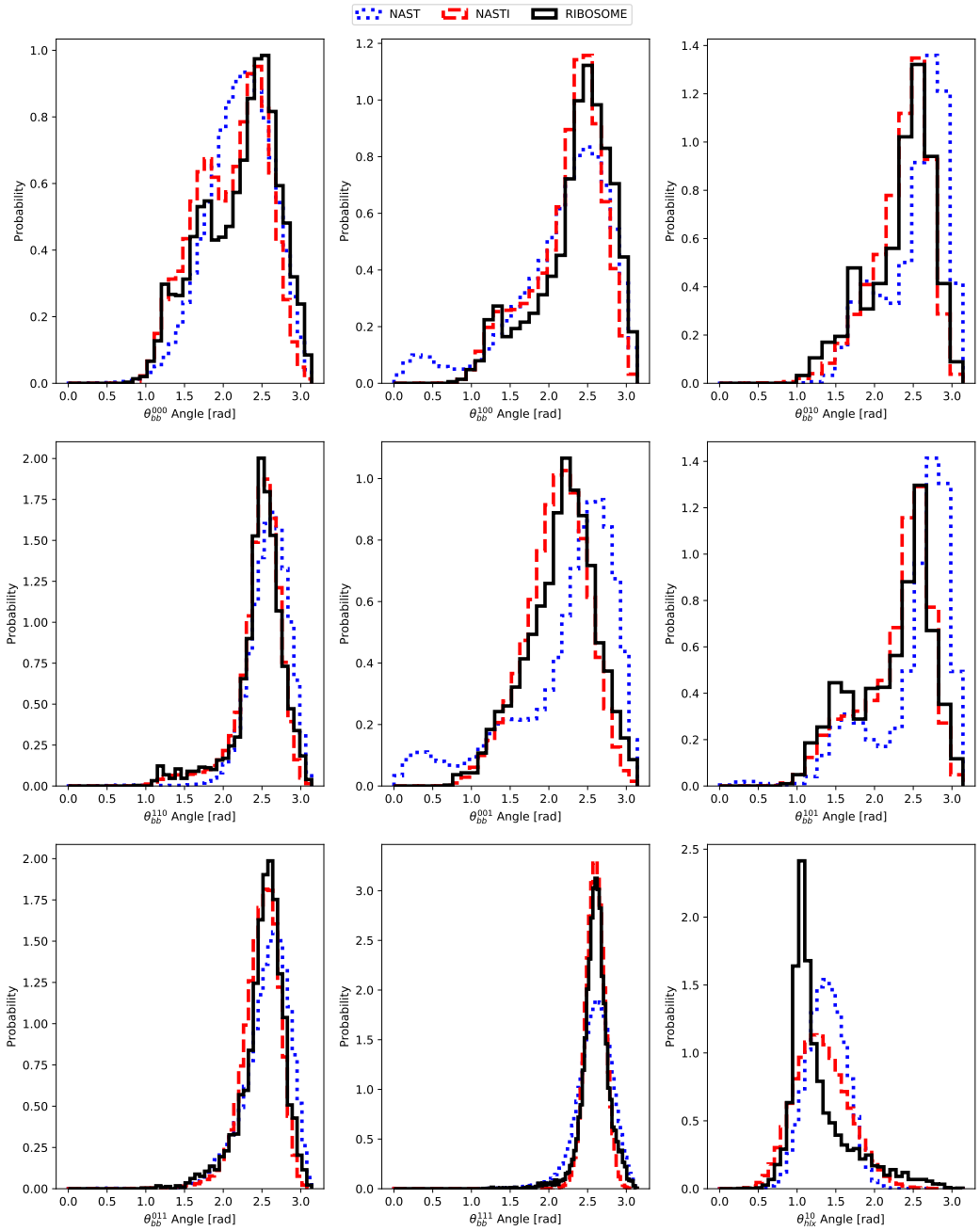

Figure A.1.: Bond angles were collected from ribosome substructures and from samples of simulations of the five structures in the test set.

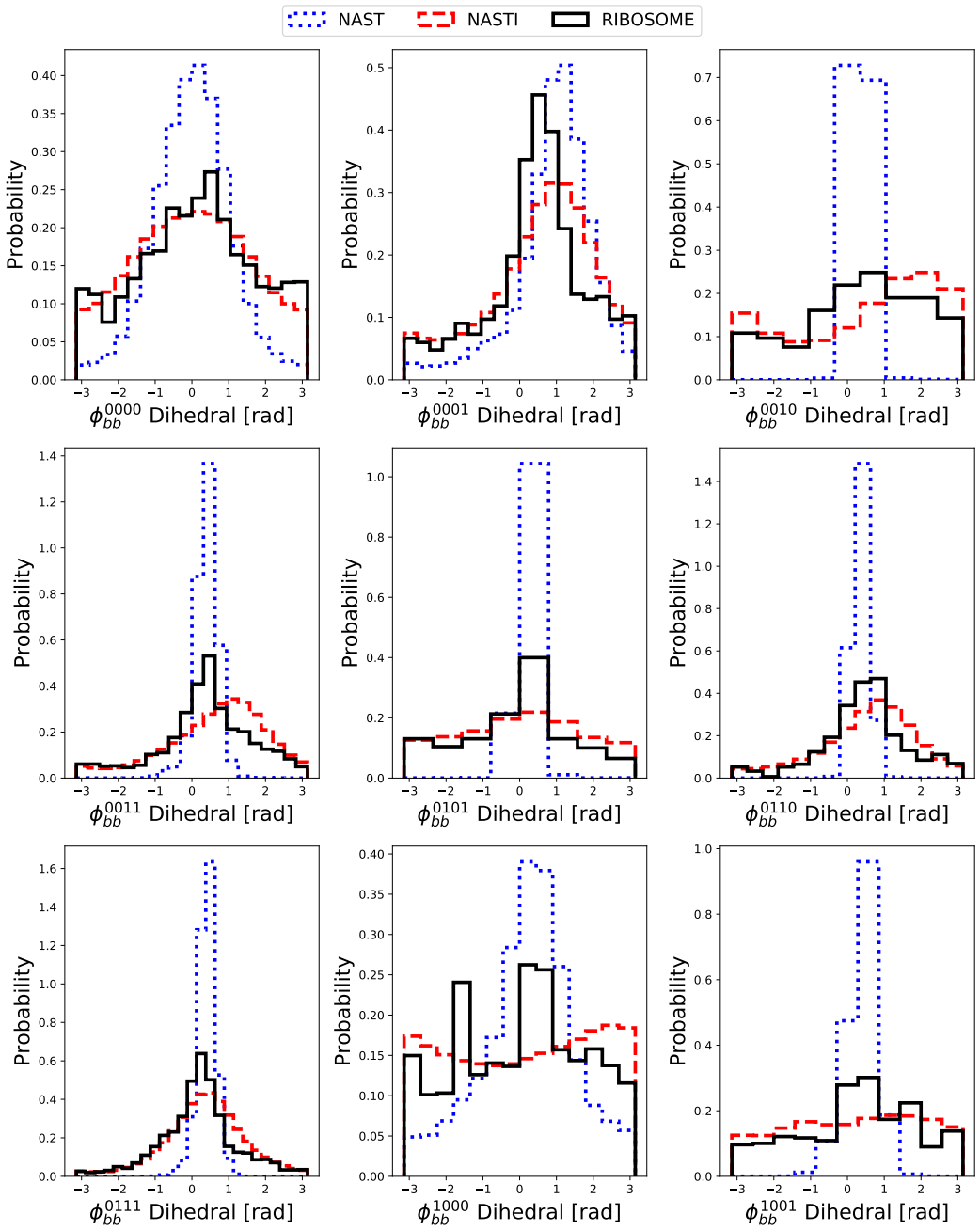

Figure A.2.: Dihedrals were collected from ribosome substructures and from samples of simulations of the five structures in the test set. This figure shows distributions for half of the dihedral types. The other distributions are shown in Figure A.3.

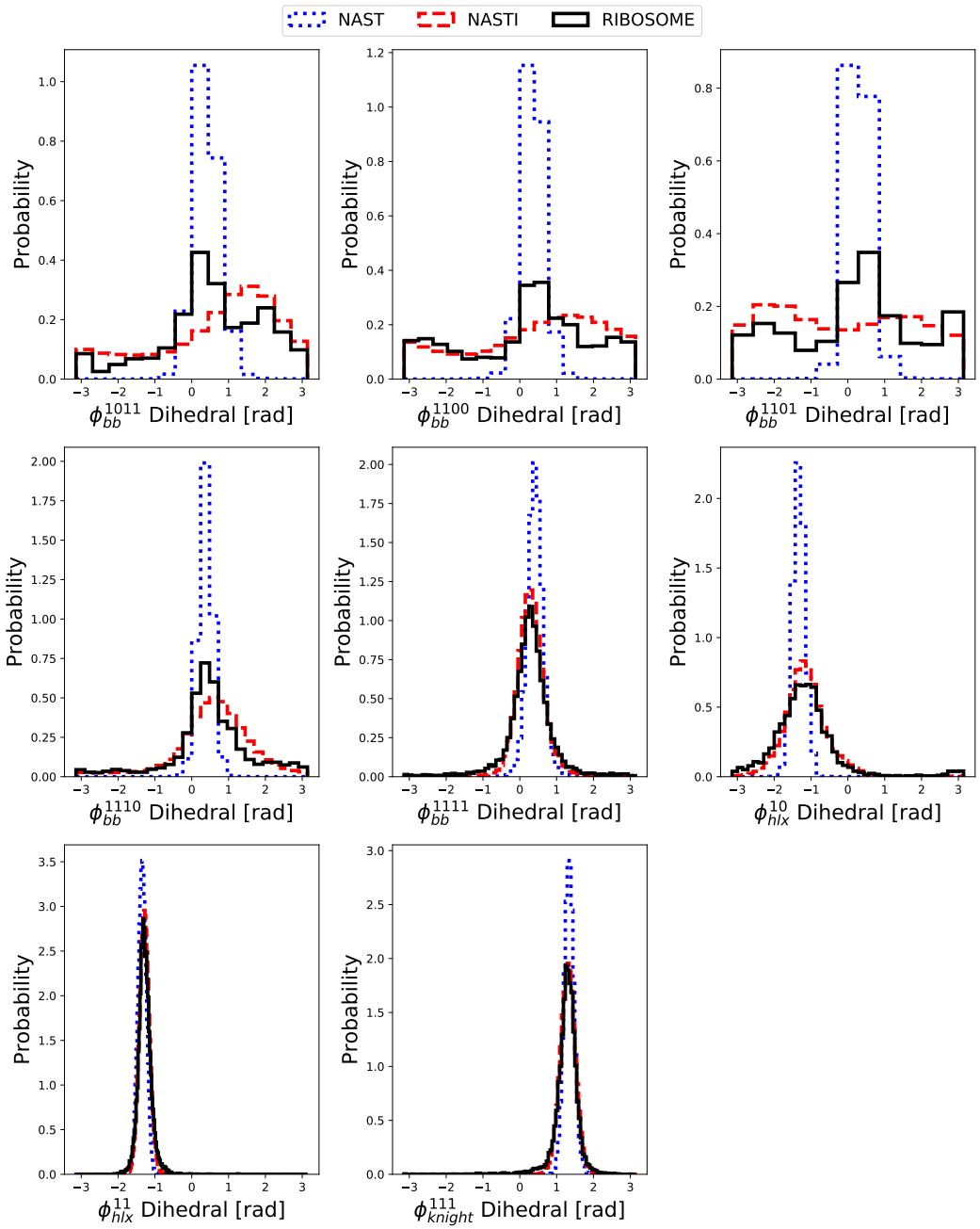

Figure A.3.: Dihedrals were collected from ribosome substructures and from samples of the five structures in the test set. This figure shows distributions for half of the dihedral types. The other distributions are shown in Figure A.2.

PDBID	CHAINID
4r0d	A
3jcs	3
4bts	DA
4v4q	CA
5g2x	A
4r4v	A
5o60	A
4gxy	A
4v19	A
5lj3	Z
5jup	EC
4v92	AZ
5mrc	A
5mrc	aa
2a64	A
3p49	A
3jan	4
5a2q	3
4gma	Z
1u6b	B
1u9s	A
5t2a	2
5t2a	E
5t2a	A
3pdr	A
3j7p	5
5lzt	9
3j79	A
1nbs	B
5it9	i
5aj3	A
3j9w	AA
5tc1	R
3dir	A
4p8z	A
4v88	A6
4v88	A8
5j01	A
1y0q	A
3q1q	B

Table A.1.: **Parameterization set.** List of RNA chains used for parameterizing the energy terms of NASTI.

		Bonds	
		k (kJ mol^{-1} nm^{-2})	r' (nm)
r_{bb}	00	544.966	0.584
	01	802.825	0.585
	10	1072.806	0.579
	11	3455.842	0.577
r_{hlx}	11	651.475	1.362

		Harmonic angles	
		k (kJ mol^{-1} rad^{-2})	θ' (rad)
θ_{hlx}	10	7.214	1.297
	11	125.8	1.079

		Dihedrals	
		k (kJ mol^{-1})	ϕ' (rad)
ϕ_{bb}	0000	-1.063	0.216
	0001	-2.432	0.669
	0010	-1.194	0.887
	0011	-2.751	0.536
	0100	-1.037	0.118
	0101	-1.697	0.121
	0110	-3.35	0.552
	0111	-3.912	0.196
	1000	-0.629	0.322
	1001	-1.271	0.644
	1010	-2.237	0.6
	1011	-2.225	0.811
	1100	-1.157	0.913
	1101	-0.904	0.534
	1110	-4.01	0.572
	1111	-8.587	0.292
ϕ_{hlx}	10	-6.276	-1.278
	11	-77.17	-1.275
ϕ_{knight}	111	-26.6	1.278

	Non-bonded	
	ϵ (kJ mol^{-1})	σ (nm)
helical	8.69e-15	11.92
non-helical	4.32e-14	9.13

Table A.2.: **Force field parameters.** Non-bonded parameters are from NAST release 1.0. Parameters for the backbone angles are listed separately.

		Spline parameters			
		t (rad)	a (kJ mol^{-1} rad^{-2})	b (kJ mol^{-1} rad^{-1})	c (kJ mol^{-1})
θ_{bb}	000	0.835642220348	41.2493261429	-111.519635643	78.4298524533
		1.19641165318	43.3048495624	-116.438139988	81.3721304106
		1.3767963696	-56.3900635912	158.081049007	-107.606380986
		1.46698872781	9.37088520903	-34.8600922328	33.9148586786
		1.73756580243	21.1872201163	-75.9234111226	69.5899679972
		1.91795051885	-38.414143288	152.701520608	-149.65568522
		2.00814287706	-11.5291235548	44.7235988545	-41.2381379957
		2.27871995168	20.1396124305	-99.604762214	123.203819985
		2.54929702631	11.7464096906	-56.8112286423	68.6571060448
		3.09045117556	-154.397010413	970.105027302	-1518.16016915
	001	0.71674428218	40.4517363906	-92.854049802	59.0548295825
		1.03682938522	5.53654906861	-20.4518653898	21.520473406
		1.35691448827	0.190251467418	-5.94292804248	11.6767797581
		1.89038966001	7.62869335059	-34.0660352875	38.2585953298
		3.06403503784	-81.7658719152	513.750125046	-801.005359449
	010	1.15274583282	2.33500229336	-12.6434992694	17.6795114237
		1.64850749398	5.87841513593	-24.3261845199	27.3090085164
		1.97901526809	-12.0347796106	46.5747872874	-42.848044348
		2.14426915514	-1.3911728219	0.92927181444	6.09009110154
		2.47477692925	24.4694061235	-127.069056488	164.473746033
		2.6400308163	30.9133393423	-161.093421039	209.386431494
		2.97053859041	-65.9590117006	414.432693193	-645.424834563
	011	1.03361260372	-4.93738094208	3.92180768195	16.5720301616
		2.09729086327	15.2480767512	-80.7477443	105.360369046
		2.59784533835	41.1004163289	-215.068504014	279.832648779
		3.09839981342	-458.670237982	2881.91010013	-4518.00631585

Table A.3.: **Spline parameters A.** Parameters for the spline terms used as the potential for backbone angles. The parameter t defines the intervals. a, b and c are the coefficients for the quadratic, linear and constant terms of the polynomial respectively. This table contains parameters for half of the angle terms, the rest is found in the next table.

		Spline parameters			
		t (rad)	a (kJ mol^{-1} rad^{-2})	b (kJ mol^{-1} rad^{-1})	c (kJ mol^{-1})
θ_{bb}	100	0.819005179406	28.4455845602	-75.2990333901	53.46268156
		1.17382457513	24.3161347165	-65.6045339737	47.7728607307
		1.41037083894	-8.49645880246	26.9513161247	-17.4961752454
		1.76519023466	-2.56175804537	5.9995644806	0.995738455229
		2.00173649848	-11.6191079996	42.2604204461	-35.2966009708
		2.23828276229	16.757397084	-84.7688639188	106.867127779
		2.3565558942	20.1987314755	-100.988257608	125.978081678
		2.59310215801	-1.25872325317	10.2944867173	-18.3056805511
		2.82964842182	40.0855039941	-223.684768046	312.733833939
		3.06619468564	-146.788774224	922.30106946	-1444.17400845
	101	0.932073178516	19.1680539409	-58.801570524	47.2829154262
		1.57626308338	-2.79819769235	10.4476125353	-7.29454997909
		2.05940551203	-13.4238120767	54.212510199	-52.3593857201
		2.38150046446	26.6682438316	-136.745989335	175.024491946
		2.54254794068	29.5436318644	-151.367613178	193.612581742
		2.70359541689	8.25836319151	-36.2741035148	38.0294391224
		2.86464289311	19.7925585732	-102.356805171	132.681109951
		3.02569036932	-75.130109205	472.056398284	-736.317138902
	110	0.892551425953	46.9197733702	-121.834889026	85.5991731809
		1.19658762546	31.5082236782	-84.9523497245	63.5325781191
		1.34860572522	3.30628640917	-8.88576159818	12.2406599966
		1.50062382497	-10.2274970068	31.7324740738	-18.2356860922
		2.03268717411	-7.60606387796	21.0753670763	-7.40440373878
		2.33672337362	32.5724150438	-166.69661455	211.98118545
		2.56475052326	28.4253835358	-145.42441209	184.702239255
		3.0968138724	-342.035124883	2149.0700712	-3368.10893374
	111	1.1627287967	-12.4936435252	21.7061139463	18.8916203581
		2.14649579563	6.30532446316	-58.9976975524	105.506816395
		2.39243754537	67.6033293277	-352.300994141	456.361725864
		2.6383792951	58.775521792	-305.718784894	394.910957666
		2.88432104484	51.753786062	-265.212904619	336.494976208
		3.10293593349	-723.864572183	4548.17524433	-7131.32254839

Table A.4.: **Spline parameters B.** Parameters for the spline terms used as the potential for backbone angles. The parameter t defines the intervals. a, b and c are the coefficients for the quadratic, linear and constant terms of the polynomial respectively. This table contains parameters for half of the angle terms, the rest is found in the previous table.

Gap open	Gap extend	Margin scaling	Base pair bonus
2.5	0.1	0	0
2.5	0.1	0	4
2.5	0.1	1	0
2.5	0.1	1	4
2.5	0.3	0	0
2.5	0.3	0	4
2.5	0.3	1	0
2.5	0.3	1	4
5.0	0.1	0	0
5.0	0.1	0	4
5.0	0.1	1	0
5.0	0.1	1	4
5.0	0.3	0	0
5.0	0.3	0	4
5.0	0.3	1	0
5.0	0.3	1	4

Table A.5.: **Parameter combinations used in the search for candidate chains for training and test set.**

Margin factor	0.1	0.500	1.00		
Base pair bonus	0.1	3.000	6.00		
Gap open	0.2	2.150	4.10	6.050	8.0
Gap extend	0.1	1.075	2.05	3.025	4.0

Table A.6.: **Parameters used at the beginning of simplex optimization.** A list of all combinations where gap open is larger than gap extend was assembled and yielded 171 start points.

Chain	CVRRY	ALFONS	STAR3D	FREEDOLIN
1s72_9	6.91×10^{-1}	6.98×10^{-1}	5.24×10^{1}	6.10×10^{-2}
2qbg_A	6.42×10^{-1}	6.18×10^{-1}	8.15×10^{1}	3.34×10^{-2}
2a64_A	6.71×10^{-1}	6.86×10^{-1}	5.66×10^{-1}	2.59×10^{-2}
3dhs_A	6.82×10^{-1}	6.54×10^{-1}	4.28×10^{-1}	2.54×10^{-2}
1fjg_A	1.40	1.50	1.90×10^{1}	1.47×10^{-1}
2aw7_A	1.42	1.51	2.94×10^{1}	1.26×10^{-1}
1s72_0	3.15	3.26	9.86×10^{1}	1.98×10^{-1}
3u5h_5	3.29	3.38	1.51×10^{2}	2.94×10^{-1}

Table A.7.: **Average time in seconds required for chain wise preparations.**

Pair	CVRRY	ALFONS	STAR3D	FREEDOLIN	URSULA
1s72_9 & 2qbg_A	6.67×10^{-3}	7.57×10^{-3}	4.76×10^{-1}	6.70×10^{-3}	6.73×10^{-3}
2a64_A & 3dhs_A	1.44×10^{-2}	9.60×10^{-3}	2.96×10^{-1}	6.67×10^{-3}	6.73×10^{-3}
1fjg_A & 2aw7_A	1.46×10^{-1}	1.04×10^{-1}	7.56×10^{-1}	8.47×10^{-3}	8.27×10^{-3}
1s72_0 & 3u5h_5	3.59×10^{-1}	2.90×10^{-1}	1.43	9.83×10^{-3}	1.08×10^{-2}

Table A.8.: **Average time in seconds required for a comparison (alignment or alignment-free) excluding the preparation of chain features.**

Pair	ALFONS	CVRRY	SARA
1s72_9 & 2qbg_A	1.32	1.34	2.69
2a64_A & 3dhs_A	1.35	1.37	10.7

Table A.9.: **Average time in seconds required for an alignment including the preparation of chain features.**

Chain	FRASS	FREEDOLIN	URSULA
1s72_9	2.71	2.80	2.75
2a64_A	2.19	2.86	2.79
1fjg_A	21.5	3.20	3.20
1s72_0	128	3.34	3.48

Table A.10.: **Average time in seconds required for a database search with a given query chain.**

B Appendix B. Scientific contributions

This appendix lists my scientific contributions during the time of my PhD.

Publication

Petersen NP, Ort T, Torda AE. 2019. Improving the numerical stability of the NAST force field for RNA simulations. *J Chem Theory Comput* **15**: 3402-3409

Talk

Petersen NP. 2016. Mixing RNA scoring functions and 3D models. *14th Bioinformatik Herbstseminar*, Doubice, Czech Republic.

Poster presentations

Petersen NP, Torda AE. 2016. RNA secondary structure prediction with 3D constraints. *German Conference on Bioinformatics*, Berlin, Germany.

Petersen NP, Torda AE. 2015. RNA secondary structure prediction with pseudoknots using Newtonian dynamics simulations. *23rd Annual International Conference on Intelligent Systems for Molecular Biology and 14th European Conference on Computational Biology*, Dublin, Ireland.

Anhang C.

C Danksagung

Zuerst bedanke ich mich bei Andrew Torda für die Betreuung dieser Arbeit und Unterstützung während dieser Zeit, angefangen bei der Bewerbung um ein Stipendium und später auch durch die Anstellung in der Arbeitsgruppe. Die Tür war stets offen für Fragen und die Laune immer gut. Insbesondere seine optimistische Einstellung war für mich oft sehr ermutigend. Außerdem konnte ich bei der Arbeit tun, was und wie ich es wollte. Auch das weiß ich sehr zu schätzen. Abschließend kann ich sagen, dass ich eine gute Zeit und viel Spaß hatte und eine Menge gelernt habe.

Bei Daniel Wilson bedanke ich mich für das Interesse sowie die Zeit und Mühe diese Arbeit zu begutachten.

Besonders herzlich möchte ich mich auch bei Anette Schade für die ständige Hilfsbereitschaft bedanken.

Thomas Ort, Jan Schawo und Irina Maiber haben durch die Arbeit an ihren Abschlussprojekten zur Entwicklung der in diesem Schriftstück vorgestellten Software beigetragen. Thomas hat einen Prototypen für NASTI geschrieben. Jan hat die erste CVRRY-Version implementiert und dem Projekt seinen Namen gegeben. Irina hat die ersten Experimente zur Verwendung von Basenpaaren in CVRRY gemacht und entwickelt derzeit eine neue Bewertungsfunktion für Strukturalignments. Die Betreuung der Projekte und die Zusammenarbeit mit allen dreien hat mir ausgesprochen gut gefallen und viel Spaß gemacht.

Wenn man nette Kollegen hat, kommt man doppelt so gerne in die Uni. Deshalb vielen Dank an die Crew (aka. die supercoole Mensa-Gang), das sind Björn, Marco, Martin, Heiner, Tobias, Timur, Giorgio, Massoud, Simon, Kira, Savitha, Jan und Jan. Martin wünsche ich noch eine gute Besserung und Björn teile ich auf diesem Wege mit, dass der Nachtwächter ihn vermisst.

Bei Corinna, Marie, Lara, Basti und Randolf bedanke ich mich allerherzlichst fürs Lesen und die Korrekturvorschläge.

Mein größter Dank gilt Corinna für die viele Unterstützung in den letzten Jahren - insbesondere in den letzten Monaten - insbesondere in den letzten Wochen - insbesondere in den letzten Tagen. Ich bin dann wohl die nächsten Wochen mit Putzen, Einkaufen und Kochen dran.

Das Leben ist am besten, wenn man liebe Menschen kennt. Insbesondere danke ich meinen Eltern, die immer da sind, meinen Schwestern und dem Rest meiner Familie sowie meinen supercoolen Freunden.

Finanzierung

Die vorgestellte Arbeit wurde über die ersten eineinhalb Jahre durch ein Stipendium nach dem “Hamburgischen Gesetz zur Förderung des wissenschaftlichen und künstlerischen Nachwuchses” und anschließend über eine Stelle als wissenschaftlicher Mitarbeiter in der Arbeitsgruppe “Biomolekulare Modellierung” finanziert.

Anhang D.

D Gefahrstoffe und KMR-Substanzen

Für die vorliegende Arbeit wurden keinerlei Laborexperimente mit chemischen oder biologischen Materialien durchgeführt. Aus diesem Grund sind hier keine Gefahrstoffe, krebserzeugende, erbgutverändernde oder fortpflan-zungsgefährdende (KMR) Stoffe angegeben.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, die vorliegende Dissertation selbst verfasst und keine anderen als die angegebenen Hilfsmittel benutzt zu haben. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium. Ich versichere, dass diese Dissertation nicht in einem früheren Promotionsverfahren eingereicht wurde.

Hamburg, den 3.4.2020

_____

Nils Peter Petersen